Generation and Analysis of Attack Graphs on Computer Networks

Alastair Janse van Rensburg

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas Term 2018

Abstract

The complexity of computer network attacks requires a sophisticated understanding of network security. Attackers combine seemingly inconsequential vulnerabilities into damaging attacks. Attack graphs compactly represent the possible ways exploits can be combined in the network by attackers. Armed with an accurate, up-to-date and sufficiently-detailed attack graph model, network defenders could straightforwardly select the most critical vulnerabilities and weaknesses in their network, allowing them to perform the necessary actions to optimally mitigate the threat.

But attack graphs suffer from a number of problems that stand in the way. They rely heavily on data sources that were not intended to be used for this purpose, and have not been demonstrated to be reliable enough. Graphical models frequently suffer from complexity problems, and attack graphs are no exception. Once an attack graph is constructed, it requires analysis methods that are hard to verify, making any conclusions hard to justify.

In this thesis, I address each of these problems through a variety of theoretical methods. I begin by establishing a clear definition of attack graph, based on template-matching methods. This is used to provide a set of comparisons by which attack graph analysis techniques can be verified theoretically, demonstrating that some common analysis methods perform unreasonable assessments. From this basis, I examine the assumptions that underlie attack graph models, and propose two novel assumptions, the single-precondition assumption and the partitioned-preconditioned assumption. I also provide a motivating smart home example, together with a dataset of vulnerabilities.

I provide two independent contributions towards the generation and analysis of attack graphs; the first is a method to construct attack graphs without vulnerability data, making them easier to construct and allowing them to model zero-day vulnerabilities. The second is a method to parametrise attack graphs, enabling analysis to incorporate characteristics of the attacker and decisions of the defender, so that analysis can be performed across the full spectrum of possible attackers and defender decisions. Finally, these techniques are combined and presented with a collection of possible analysis methods, and applied to the smart home use case through a software implementation.

Statement of Originality

This thesis is written in accordance with the regulations for the degree of Doctor of Philosophy. The thesis has been composed by myself and has not been submitted in any previous application for any degree. The work presented in this thesis is my own.

An early form of the work in Chapter 7 was presented in [JvRNG16]. The contribution of this paper was my own, supervised by the co-authors.

Acknowledgements

This research was supervised by Michael Goldsmith and Jassim Happa, and I would like to thank them for their continued support, guidance and advice throughout the project. I would also like to thank Jason Nurse, who supervised the initial project which began this research. Aside from these direct supervisors, I am very grateful to the rest of our research group, who have all provided advice throughout. Thank you, particularly, to Jo, for your feedback and comments.

I would also like to thank Ivan Flechais, Thomas Gibson-Robinson, and Margaret Varga, who, through their comments at my Transfer and Confirmations of Status, provided me with invaluable feedback and direction, with which this thesis was much improved.

Thank you especially to Richard, Claire, Louise and all the other friends who have made this process both more manageable and more enjoyable. Finally, I would like to thank my parents for everything they have done.

Contents

Chapter 1

Introduction

The complexity of computer network attacks requires a sophisticated understanding of network security. Attackers combine seemingly inconsequential vulnerabilities into damaging attacks. To mitigate this, defenders require an understanding that encompasses the many ways in which different weaknesses can be combined and exploited.

Defensive techniques that protect components in isolation are necessary tools with which network defenders can provide protection. Cryptographic analysis can verify the security of protocols; operating system countermeasures mitigate memory-modifying attacks; input sanitisation prevents database exploitation.

But none of these solutions are perfect: in each implementation there is a flaw; in each protection there is a workaround. As these are discovered, exploited and eventually patched, the landscape of vulnerability shifts – and in a modern, networked system, keeping up is difficult.

In practice applying patches is not free, but entails costly downtime, breaks dependencies and faces user resistance. Over time, systems become perforated with known-yet-unpatched weaknesses, allowing a malicious user to move through the network.

The connections between these weaknesses become a complex map of vulnerability. For an attacker, this is a wealth of potential paths to their goals. For a defender, understanding the topology of their vulnerabilities is key to controlling and blocking those paths.

This map of vulnerabilities is formalised as an *attack graph*.

1.1 Context

Cyber security is an increasingly important area of research, as the number of devices and our dependency on them rapidly grows. The cost of cyber crime is difficult to

estimate, but is certainly considerable, with some estimates as high as $2tn [Sha16]. Cyber attackers are a threat to a wide variety of areas, from infrastructure such as transport, emergency departments, and electricity providers, to homes through the Internet of Things (IoT) [TO18]. The IoT, in particular, is emerging as a popular target for attackers, particularly as an easy entry point [Sym19].

The challenges of applying patches [CCZ08] mean that known vulnerabilities pose a significant threat to security, with attackers able to utilise toolkits of existing exploits (e.g. Metasploit [Met]) to readily perform attacks. These toolkits have been recorded abusing vulnerabilities soon after their public disclosure, both by suspected novice attackers and more-dedicated attackers alike [RSD07]. This is particularly prevalent in the IoT, where patching is made even more costly and hence even less likely [HWMK17].

Furthering these problems, huge numbers of vulnerabilities are being discovered – more than $15,000$ vulnerabilities were reported to the National Vulnerability Database (NVD) in 2018 [NVD]. Consequently, any approach to cyber security must take into account the impossibility of maintaining a vulnerability-free system and be prepared to face rapid exploitation of newly-discovered vulnerabilities.

Aside from known vulnerabilities, defenders must also expect zero-day attacks, that is, attacks that have never been publicly observed before. These attacks cannot, by their nature, be mitigated through patching or traditional signature-based defences [BD12]. Some estimates measure the number of currently available zero-days in the thousands [MMBC09].

In light of this, practical cyber security cannot solely focus on attempting to fix every vulnerability; there are always systems that have not (or cannot) be patched, and there are always malicious actors discovering new vulnerabilities. Accordingly, network security analysis should attempt to understand how systems can be made secure in spite of the existence of weakness.

Attack graphs [SPEC01, WIL+08], alongside other techniques such as risk assessment [RDRN+18], offer the means by which an understanding of the possible impact of cyber attack can be gauged. Whereas many techniques offer an organisational perspective on security, attack graphs (and similar approaches such as attack trees [Sch99]) offer a technical perspective and provide a picture of the precise ways in which vulnerabilities can be exploited in conjunction with each other. These technical approaches offer network defenders the opportunity to estimate the precise increased risk caused by each exploit on their system.

1.2 Motivation

Attack graphs are, in their simplest form, a graph of the known vulnerabilities in a system, such that each vulnerability is connected to those it enables. The graph compactly represents the ways in which individual exploits can be combined into complex, multi-stage attacks.

Formalising the system in this way is intended to provide a basis for precise quantification of the security of the system [HOS09, PGLCX16]. Armed with an attack graph, defenders could optimally deploy their limited resources to minimise the impact of cyber attack, targeting consequential vulnerabilities while avoiding unnecessary expenditure on insignificant ones.

But such benefits are difficult to realise. A model is only as accurate as the data behind it. Attack graphs are highly dependent on precise, up-to-date, accurate and comprehensive information about the system and its exploits. Because they seek to illuminate all possible paths through the system, missing out a single vulnerability can have a significant impact on the graph, and consequently on any conclusions drawn from it.

Graphical methods are highly susceptible to combinatorial explosion, with the computational [SHJ+02] – and visual [NJ04] – complexity of models often growing unmanageably quickly. Attack graphs are no exception; creating attack graphs for even small networks requires careful restraints and restrictions [AWK02]. Creating attack graphs for modern systems with potentially hundreds of thousands of hosts poses even more significant challenges.

Creating a perfect representation of a system is of no use to a defender who cannot analyse it. To ensure attack graphs are worth the potentially-significant investment that their construction demands, analysis techniques must reliably translate the theoretical model into practical conclusions.

These challenges, among others, explain why there is little evidence to suggest that attack graphs are widely accepted by practitioners [SS15].

1.3 Scope

This thesis examines the theory behind attack graph models to enable theoretically-valid conclusions to be drawn from clear assumptions. It aims to adapt existing methods to provide novel techniques that are not susceptible – or are less susceptible – to the problems of other methods. The motivating example of the smart home is

intended as a particularly-suitable and worthwhile application, but the techniques discussed can be similarly applied to other systems, such as other Internet of Things scenarios or typical enterprise networks. While the examples and descriptions in this thesis primarily relate to network- or host-based exploits, this is not a necessity, and much of the work can readily be applied to more general attacks, such as social engineering.

Attack graph testing is rare in research because of the great difficulty it poses; providing such testing requires considerable amounts of data gathered across many systems over a long period. The relative infrequency of significant attacks against any individual system mean that observing real attacks is highly challenging, and confidently providing any conclusion about why a particular system was *not* compromised is near-impossible. Direct testing of these techniques in real-world scenarios is therefore considered out of scope for this thesis. This thesis focuses on providing a theoretical foundation on which we can reason about attack graphs grounded in objective statements, so that in the absence of empirical testing of any new techniques, practitioners can have the reassurance of theoretical validity, if not empirical soundness.

1.4 Research Outline

While end-user adoption is a complicated, multi-faceted issue, the underlying problems in the theory of attack graphs are at least partly responsible for the lack of real-world deployment of attack graph techniques. Specifically, I aim to consider each of the problems raised above:

Data. The data sources used in much of attack graph work often appear unsuitable and incomplete. Techniques that use this data are therefore unreliable.

Performance. Attack graph techniques have become significantly more efficient over time, but it is not clear or well-established how these performance improvements are achieved, or what the full consequences of these improvements are.

Analysis. Many attack graph analysis methods have been proposed, but few have clear justifications, and many lead to questionable security assessments in certain cases. Analysis often requires manual interpretation, making it slow and subjective.

In order to consider these problems, and provide novel and improved techniques, I began by surveying existing definitions of attack graphs, with particular concern for their formalisations, implicit assumptions and dependencies. I also examined alternatives to attack graphs, most prominently attack trees, and other methods of security quantification. During this process, I adapted my research plan to account for criticisms of quantification-based approaches to security, and ensure that my work facilitated comparisons with other quantifications. This literature review is presented in Chapter 2.

Following on from this, my aim was to establish a standardised definition which correlated well with the work presented by others. This definition of an attack graph, heavily inspired by previous work, is presented in Chapter 3, and serves to establish a consistent definition for use throughout my work. I intended this definition to be suitably general that it could fit a significant portion of the existing definitions of attack graphs of the same type (specifically transition models, see Section 2.2.6). In light of my earlier discussion on quantified security, I also established consistent examples, including the Smart Home dataset (Section 3.2.1). These are intended to enable consistent comparison throughout my work, and to provide sufficient detail for later work to be applied to the same scenarios, facilitating the baselining and cross-work comparison that is uncommon in existing attack graph work.

With the definitions and example scenarios established, I began to approach the problems listed above. Initially from a theoretical standpoint, and later by proposing practical techniques. By addressing the theoretical components of attack graphs first, I aimed to better understand the justifications and claims that can be made about the analysis and performance of attack graphs.

In Chapter 4, I proceeded by making clear assumptions and attempting to draw security conclusions from these assumptions. In this fashion, I aimed to present justified conclusions on attack graphs that improved the analysis possible and reduced the reliance on data. In particular, I hoped to provide an initial exploration of the use of attack graphs in a theoretical fashion; studying the properties of graphs in the abstract to discover theoretically-valid conclusions that could then be applied to attack graphs based on real data.

Continuing my theoretical approach, I studied the performance of attack graphs in existing work. Much existing work makes use of implicit and explicit assumptions to justify the significant improvements in performance that have been observed in attack graph techniques since their original formulation. In Chapter 5, I examined some of these assumptions and formalised two assumptions that were previously implicit. By

studying these assumptions independently of any proposed techniques, I intended to consider them in the abstract; assessing both their truthfulness and their benefits.

I subsequently aimed to construct an attack graph technique, built in light of the above theoretical work, that addressed my research aims of solving the three listed problems. To do so, I developed two contributions, presenting Topology-Inferred Graphs in Chapter 6 and Parametrised Attack Graphs in Chapter 7. This pair of contributions was intended to be independent but complementary; the first addressing the means by which attack graphs are constructed, with particular concerns for performance and data requirements, and the second addressing the means by which attack graphs can be analysed.

Finally, I conclude my work by combining the techniques of Chapters 6 and 7, building upon the theoretical work in the earlier chapters, into a combined technique, which is presented in Chapter 8 and implemented and demonstrated in Chapter 9.

1.5 Contribution

The contribution of my thesis is presented in the following chapters of work:

- Chapter 3: **Foundations**. In this chapter, I provide the groundwork for the rest of the thesis. I provide my own definition of attack graph, based on a stricter formalisation of template-matching methods. I provide an example use case, based on a smart home, and collect a set of real-world exploits to populate this example. This smart home example is used throughout the thesis as a motivating and illustrative example.

- Chapter 4: **Relative Attack-Graph Security**. I present a theoretically-grounded assessment of attack graph security in special cases, which can be used to test attack graph metrics for validity. Through this test, I find cases where commonly-cited metrics make illogical assessments. Finally, I propose alternative metrics that attempt to capture the same concept but in a valid fashion.

- Chapter 5: **Single-Precondition Assumption**. I examine the claimed performance benefits of a key attack graph construction assumption, monotonicity, and provide graph counterexamples that observe the assumption without performance improvements. I propose the formalisation of a *single-precondition assumption*, which is a concept implicitly used in existing work, but has not been

explicitly stated. I also provide a more general assumption, the *partitioned-precondition assumption*, which relaxes the single-precondition assumption but maintains much of the performance benefits. I provide justification for these assumptions by examining real-world exploits under the assumptions, and demonstrate their consequences for performance.

- Chapter 6: **Topology-Inferred Graphs**. I propose a novel attack graph generation method that circumvents the strenuous requirement of vulnerability data, by deriving attack graphs directly from the topology of the graph.

- Chapter 7: **Parametrised Attack Graphs**. I propose a novel attack graph analysis method that enables information about the attacker and the system to be combined with the graph to create parametrised attack graphs. These graphs enable analysts to examine the security impact of their possible choices, the effect of different attacker capabilities and drastically simplify the process of assigning probabilities to attack graphs. I provide deterministic and probabilistic security assessments that can be made with this method.

- Chapter 8: **Analysis**. I examine the results of combining all of the aforementioned contributions into a unified technique, and present a set of analysis methods that can be used to understand the resulting attack graphs. I also present a selection of possible use cases and explain how the analysis methods can be applied to them.

- Chapter 9: **Implementation and Use Case**. I provide an architectural overview of a Python implementation of the combined attack graph technique. I also demonstrate the technique and implementation on the smart home as an example use case.

Chapter 2

Literature Review

In this chapter, I provide a critical review of quantified network security, an overview and analysis of related works in attack graph analysis, and a discussion of how these have been applied in the smart home. This is presented through the following sections:

- Section 2.1: The problem space addressed by this thesis. In particular, I give an overview of techniques for analysing and quantifying network security, including measurements, metrics and models. Further, I present justification for the choice of *attack graphs* as the focus of my work.
- Section 2.2: A detailed discussion of key methods for generating attack graphs, that is, the creation and format of attack graphs. I also provide a discussion of the literature and themes within it.
- Section 2.3: Related work in the analysis of attack graphs, that is, how attack graphs can be used to make or influence security decisions.
- Section 2.4: A discussion of the Internet of Things and the *smart home* scenario, with particular concern over the differences between these and traditional scenarios. I also provide a discussion of existing attack graph methods in the smart home environment.

2.1 Network Security

System administrators charged with protecting networks are tasked with understanding and minimising the risk faced by the network.

Ensuring the security of a network may be performed *statically*, without regard to a particular attack, or it may be performed *dynamically*, in response to a specific

intrusion. Static security concerns the design and topology of the network, how it should be configured, which devices and services should be implemented in it, and other decisions that can be made while creating and maintaining a network. Dynamic security, on the other hand, concerns the discovery and identification of compromised components of the system, and the inference and prediction of properties and consequences of the attack.

Typical static security includes: consideration of network design principles, such as weakest-link security [Arc03], defence-in-depth [Wol16] and implementation of moving-target defences [ZZD+12, ZDO14]; vulnerability discovery [Nes15] and weakness enumeration [CWE]; deployment of software and hardware appliances and services; and trusted platform modules [LM].

Dynamic security instead relates to the system in motion: use of an intrusion detection system at host- and network-levels (e.g. misuse detection [Sno], or anomaly detection [Dar]); use of moving-target defences [1] [ZZD+12, ZDO14]; and monitoring of honeypots [Pro].

Security quantification is the process of assigning or estimating objective values of the network's security. These values can result from the direct measurement of part of the system, or they can be metrics derived from applying specific interpretative rules to raw measurements. In both cases, the aim is to perform an objective evaluation of all or part of the network to enable comparison. Security quantification is a crucial part of improving security [Jan09, RLFR17, San14, Dav09], and has a key role to play in both static and dynamic security.

Quantified comparison can be performed between potential strategies, which could be designs in the static case or incident responses in the dynamic case. Quantification which provides values with real-world meaning, such as cost or time, can also be used in conjunction with areas other than security. For instance, a metric relating to expected cost of compromise can be combined with a deployment cost to find the cheapest of a set of possible network options. Other quantifications provide values which cannot be directly mapped to physical quantities, and as such are only suitable for comparisons against other applications of the same quantification. Examples of such are more common from modelling-based approaches, with values such as *shortest path* [PS98].

The purpose of security assessment is two-fold: firstly, it aims to benchmark the security of the system so that it can be compared against internal or external standards.

[1] While the implementation and design of moving-target defences can be considered static security, their active use impacts dynamic security

For instance, a company might maintain internal regulations on newly commissioned systems to ensure they conform to a suitable level of security. Similarly, there may be external regulations, imposed by regulatory bodies which require adherence to established standards.

In conjunction with this, the second purpose of assessment is to inform decision making through the evaluation of hypotheticals: given a set of candidate configurations, which is more secure? This kind of assessment enables network defenders to optimise their use of the information available to them and make the most appropriate decision given the information they have.

Given these applications, the value of objective, comparable security evaluation is clear. It is necessary to ensure that any assessments are independent of the assessor to ensure accuracy, fairness and reliability. Accurate and precise security evaluation enables the consequences of security decisions to be understood, so that analysts can make the well-informed choices.

2.1.1 Quantification

Quantification ranges from simple numerical assignments through to interpretative models. Some authors make a clear distinction between *measurements* and *metrics*, although the definitions are not always consistent. In some definitions, measurements are the result of applying well-defined protocols to an object resulting in output values, and metrics are measurements with a rule for their interpretation [RS12a]. *Payne* [Pay06] defines metrics as *"subjective or objective interpretations of [measurement] data"*. Other authors, such as *Pendleton et al.* [PGLCX16] describe metrics as *"assigning a value to an object"* and measurements as *"estimating attributes of an object"*.

In light of these definitions, and in alignment with my aims, I will use the term measurement to refer to a single quantified property of the system, and metric to refer to precisely-derived interpretations of one or more measurements, as in *Rudolph et al.* [RS12a]. It is important that these interpretations are precisely-defined so that they can be considered independent of how they are applied. Accordingly, measurements should be unambiguous and specific, although require no stated connection to security; metrics should be objective combinations of measurements, combined with a potentially subjective or conjectured connection to security.

Ideally we could directly link a metric to security, but in practice the definition of security itself is subjective. As a result, we can only require that the metric's calculation is objective, not its stated implications. This enables the metric value to

be reliable and accurate, even if its security implication is not. When I use the term *objective comparison* in this thesis, I refer to a comparison which can be performed objectively, even if the reasons or implications are subjective.

Measurements

Measurements are direct empirical observations of the system in question. There are a considerable number of suggested network security measurements in the literature (for surveys, see [PGLCX16, RS12b]). Example measurements include *total number of missing security patches*, *number of systems where security requirements are not met*, or *average severity of vulnerabilities found*. These measurements provide the direct value of a security-related property of the system, but taken individually they do not show any overall picture of security. They also offer no interpretation of their values, and are instead raw facts about the system.

These measurements can be performed objectively, given a set of (potentially subjective) criteria. For instance, the *average severity of vulnerabilities* can use standardised severity scores (e.g. Common Vulnerability Scoring System [CVS]) so that the results are consistent. Measurements that consider conformance to security requirements can be performed objectively without making a claim about the quality or value of those security requirements. In practice, there may be subjective or interpretive elements to any measurement, but the aim is that these should be minimised.

Metrics

Metrics combine data from measurements to give an interpretation of what they may represent. This may be through simple combination or through complex models. The value of a metric is designed to be interpreted in a certain way, and therefore intended acts as an estimation of some security property. The process by which measurement data is converted to an output metric should be precise and well-defined, but the correlation between the metric and the intended security property is likely to be subjective.

Metrics can be applied at a variety of levels [RLFR17], from specific components of the system (such an individual piece of software [MMW14] or vulnerability [CVS] through to organisations [WSJ07a].

Network Models

The complexity of network systems often requires the use of a model, which can be populated with measurement data and from which metrics can be calculated.

These models are often graphical or state-based, with models such as attack trees [MO06], attack graphs [PS98], Petri nets [McD01], game-theoretic models [OA17], and stochastic models [AA13].

Some of these models (such as *Orojloo and Azgomi* [OA17] and *Madan et al.* [MG-PVT04]) model the system in an abstract sense, with generalised states that are not specific to the system. Such states include "warning", "recovery" and "emergency" and are not derived from measurements. These models can be contrasted with those (such as attack graphs and attack trees) which construct their graphical structure based on measurements from the network.

Models that use generalised states consider the system's overall health and recovery times, and do not track specific vulnerabilities or system states. Models which derive states from measurements examine the interrelationships between various vulnerabilities on the network. Consequently, the two models take different approaches to securing the network. Generalised-state models attempt to devise overall strategies for the system, by providing metrics that assess the overall impact of attackers on the system. Models with specific states capture the precise impact of attackers, but require more data to do so, and are more dependent on that data.

2.1.2 Attack Graphs and Attack Trees

Two prominent graphical models are attack trees and attack graphs. Attack trees [MO06] are graphs built through a process of refinement. The tree contains a single vertex representing the primary goal of the attacker. Each child of the goal vertex represents a sub-goal, the completion of some set of which leads to the completion of the larger goal. This is repeated down the tree until the leaf vertices. There is a large amount of work on attack trees and their variants [KPCS14], which builds upon them in various ways, such as supporting automated construction [Pau14], generalising the relationship between goals and sub-goals [Yag06], or focusing on particular incidents of interest, such as insider threat [RP05].

The second major variant of graphical models are attack graphs[2]. Formally, the differences between attack graphs and attack trees are slight. Attack graph vertices (typically but not always, as discussed below) represent properties of the system, and edges changes to those properties [PS98]. Attack graphs can contain cycles, for example, which must be handled separately in some cases (e.g. [WIL+08]), but it is not hard to see how an attack tree could be respecified as an attack graph,

[2]For the purpose of this discussion, by *graph* I refer to exclusively *attack graphs* and not *attack trees*. Generally the term *graph* would apply to both.

and vice versa (indeed some graph methods describe themselves as "trees with cross links" [OBM06]).

However, there are more significant differences in the intentions of the two techniques, and, perhaps consequently, in their applications. Whereas attack trees are commonly built by hand [Pau14, HKCH17], attack graphs are typically constructed automatically (e.g. Topological Vulnerability Analysis [JN10], NetSPA [ILP06], and others [CSHH16, ABD+18]). Attack graphs constructed from software vulnerability data do not include any other kind of attack [ILP06], while attack trees often include other types of attacker action, such as social engineering [IPHK16]. Attack graphs aim to understand the vulnerabilities of a networked system, whereas attack trees seek to explore any possible attacks on a target.

The differences, in the formal model and application, do not fundamentally separate attack graphs and trees. The differences in application are not requisite; it is perfectly reasonable to create attack graphs that include non-network vulnerabilities or that pertain to a specific attacker goal. Equally, many attack graph methods are based on *exploit dependency graphs* [NJOJ03], which result in attack graphs very similar in function to attack trees.

A key separator in this field is whether the graph or tree's state is encoded by a single vertex or by many. An attack in an attack graph model as originally stated (e.g. [PS98]) can be considered to move from state to state: at a given time, the whole system is represented by one particular vertex. The attack begins in one vertex and proceeds along a path. Conversely, in an attack tree the state of an attack is represented by a collection of vertices. The attack is considered to begin on a set of vertices and proceeds as an increasing set of vertices. This is necessary as some vertices are enabled by the logical conjunction of their children.

This separation is of particular interest because it does not cleanly divide attack graphs from attack trees. Indeed, many attack graph methods (notably *exploit-dependency graphs* [NJOJ03]) deviate from the original statement and instead model attacks in the attack tree fashion: an attack begins on multiple vertices and proceeds as a set of vertices. I use this to divide graphical methods into two categories: grouping *dependency* models with attack trees, and *transition* models separately[3]. These categories are named by the role their edges play: in a dependency model, each edge captures a dependency between two vertices, so that each vertex is linked with those

[3]Concepts similar to dependency-based graphs are referred to by other authors as "logical graphs" (e.g. [MGSPL17]) or "requires/provides" (e.g. [WLJ06, TL01]. The term "requires/provides" stems from the fact that these graphs are bipartite.

upon which it depends. In a transition model, each edge represents the transition from the state represented by one vertex to the state represented by another. A discussion of the differences between attack graph methods in each division is provided later in this chapter, in Section 2.2.6.

I believe that transition attack graphs better capture the intentions and idea behind attack graphs than dependency-based graphs. Mathematically, the ability to represent any attack as a single path through the network makes calculations more straightforward. I believe that the nature of dependency models makes them more suitable for day-to-day models drawn by hand by analysts to enhance their understanding, whereas transition models are better suited towards automatic modelling. For this reason this thesis primarily focuses on transition-based attack graphs.

2.1.3 Criticisms of Quantified Security

There is disagreement over the progress in, and current value of, quantified security. *Verendel* [Ver09] performs a study of 90 research papers that propose quantified security methods and finds that the majority offer no empirical evaluation. This leads to the conclusion that quantified security methods have not been adequately assessed for their representativeness of operational security.

Herley and van Oorschot [HvO17] support a this view from a philosophical basis, arguing that to justify security as a *science*, it is necessary that security claims should be falsifiable; that claims should have clear conditions that would demonstrate them false. *Herley* [Her17] further asserts that, regarding answers to questions of security, *"no progress is possible* [...] *if the justifications offered for them are immune to feedback and shrink from all of the risks associated with being tested against observation"*.

Authors such as *Sanders* [San14] provide a counter viewpoint: that metrics need not necessarily be perfect (or even completely correct) to be useful. The purpose of quantification is, perhaps counter-intuitively, not only to put values on things but to gain insight and aid in decision making [Jan09, RLFR17, Bay13].

In light of these viewpoints, the work in this thesis takes into account the difficulties in validation of security quantification. I attempt to perform and facilitate comparisons between my work and other quantifications: by directly applying the results of my theory to other works (see Chapter 4, particularly Section 4.3); and by providing clear comparable examples (see Section 3.3).

2.2 Attack Graph Generation

This thesis is primarily concerned with attack graphs, and so in this section I will give an overview of the most important generation techniques – that is, techniques by which attack graphs can be constructed from source data. Discussion of how these attack graphs are then analysed is set aside until Section 2.3.

2.2.1 Initial Methods

In this section, I present a few of the most important initial methods for attack graph generation. These begin with *Dacier et al.* [DD94] in 1994, who provide an initial model that is rooted in security models based on access control. *Phillips and Swiler* [PS98] update and generalise these techniques into broader methods for network security and present a version of attack graphs that underpins all subsequent work. Finally, *Sheyner et al.* [SHJ+02] provide a definition of attack graphs grounded in a concrete attack model.

Dacier et al. (Privilege Graphs)

Dacier et al. [DD94, DDK96] propose a graphical model of security based on transitions between Unix user or group privileges. This work presents an access control-centric view of security as a graphical model, and lays the groundwork for future research into more generalised models.

In this model, each vertex corresponds directly to one of the possible privilege levels. An edge between vertices represents a vulnerability (or benign method) for obtaining the privileges of one group using another. Edges can therefore represent complex, multi-step attacks, which are modelled as a single privilege step.

The focus solely around Unix permissions makes these graphs simpler to generate, but leaves out any exploits which do not affect permissions directly. Further, aggregating exploits into multi-step attacks weakens the benefits of attack graph work.

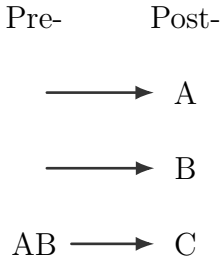
Phillips and Swiler (Attack Graphs)

Phillips and Swiler [PS98] is frequently cited as the first presentation of attack graphs. In contrast to the earlier work of *Dacier et al.*, *Phillips and Swiler* takes a very general view of the security of the system, making their model very flexible. The template-matching method presented here becomes a staple of future attack graph generation.

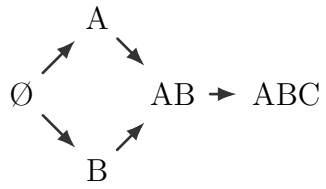
Their method begins with the creation of a set of templates, which describe the pre- and post-conditions of an exploit. The pre-conditions of the template are the

conditions which are necessary for the exploit to be performed; the post-conditions are the result of the exploit being performed.

These templates are essentially blueprints for edges – each edge on the graph is the result of a template being applied to a vertex, and each vertex is created as required, when the post-conditions of a template result in a new vertex.



(a) Exploit templates



(b) The resulting attack graph

Figure 2.1: A set of three templates, and the attack graph that they generate. The three templates are specified by their pre-conditions and post-conditions. The vertex labelled Ø is the vertex where all conditions are false.

In comparison with the earlier work, we can see that this gives us a huge variety of possible vertices; earlier work had a constrained vertex set (precisely the set of Unix groups). The model is, consequently, much better suited to model complex attacks and vulnerabilities, particularly those that involve network traversal.

This modelling power, however, comes with a significant drawback that, in many ways, will be the primary concern for future work: the set of vertices is exponential in the set of templates. This means for even modest networks the model quickly becomes infeasible.

Sheyner et al., Jha et al.

Sheyner et al. [SHJ+02, SW04] and *Jha et al.* [JSW02] adapt attack graph techniques to be suitable for model-checking, with the intention of utilising the considerable existing work in that area. They also include more general events than purely malicious attacks, such as random failures in the system. Despite these differences, the resulting model is broadly in the same vein as that of *Phillips and Swiler*; vertices are formed

via a matching process from templates, and each represents an atomic action or event on the network. Vertices are general states of the system.

This work suggests the use of automated vulnerability scanners as a data source for attack graphs, citing the manual construction process as "tedious, error-prone and unrealistic" [SW04].

They also provide one of the first demonstrations of just how inefficient early methods were: their model on 3 hosts had potentially 2^{91} states, of which 101 were reachable. This took on the order of seconds to compute. Once the network was grown to 5 hosts the number of reachable states ballooned to 6190 and the computation time was on the order of hours.

2.2.2 Key Methods and Tools

Building on these initial methods, later work attempted to address the concerns of initial critiques and augment attack graphs with further detail.

The primary drawback to early attack graph models (and to a lesser extent, recent models) is that of performance and scalability. Initial methods are prohibitively complex to generate, and performance improvements did not alleviate these problems enough to enable the modelling of networks much larger than a few hosts.

Ammann et al. [AWK02] proposed the simplifying assumption of *monotonicity* – attackers never lose privileges once they have been gained. This becomes a core assumption in most subsequent work, and is taken for granted in most work; the notable exception being the study of dynamic defences [ZZD+12].

With attack graph techniques established, research was implemented into a number of tools, such as Topological Vulnerability Analysis (*Jajodia et al.* [JNO05]) and NetSPA (*Ingols et al.* [ILP06]). Designed to provide a complete method of attack graph generation and analysis, later work often relies on them as a basis from which to work, offering some consistency and avoiding duplication of effort.

Ammann et al. (Monotonicity)

Ammann et al. [APRS05, AWK02] offer a number of innovations in their contributions. Most notably, they are the first to propose an explicit assumption of monotonicity, which is key to their claimed performance improvements. This assumption states that the attacker never loses privileges once they are gained.

This assumption is credited with reducing the complexity from exponential time to polynomial time and is a commonly-cited improvement in many subsequent works. A discussion of the importance of this assumption is given in Chapter 5.

(See also Section 2.2.3 for a discussion of this work's contribution to *host-based attack graphs*)

Jajodia et al. (Topological Vulnerability Analysis)

The Topological Vulnerability Analysis (TVA) tool (first presented [JNO05], recent developments [AJ18]) is an automated generation system for attack graphs. The tool combines data from vulnerability scanners, topology information and expert knowledge to construct an attack graph.

The primary focus of this work is on the process as a whole. While much work considered the graph construction process itself, *Jajodia et al.* [JNO05] provide a method for transforming network scan data into sets of exploits used in the attack graph construction methods.

The attack graphs constructed through TVA consist of multiple types of vertex. Some represent the conditions of the system while the rest represent exploits that are performed to move between conditions. Edges represent a dependency relationship: either exploits depending on conditions of the system in order to be performed, or conditions depending on exploits in order to exist.

Wang et al. [WLJ06] use the terminology that vulnerabilities are said to *require* their precondition vertices and *imply* their postcondition vertices; this language mirrors that of earlier work in attack scenario generation by *Templeton and Levitt* [TL01].

For modelling purposes, these two edges have a key distinction. Edges leading to an exploit from its conditions are considered to be conjunctive: the exploit can only be performed when *all* of its conditions are true. This means that attacks on the graph are (in general) trees, instead of paths, because performing a vertex may require multiple dependencies to be satisfied.

Ingols et al. (Multiple-Prerequisite Attack Graph/NetSPA)

Ingols et al. [ILP06] attempt to address the scaling problems of attack graphs with their Multiple-Prerequisite Attack Graph (MPAG). The key difference in this method is the separation of preconditions and postconditions of vulnerabilities. Previous methods typically use template-matching methods, which construct the graph iteratively by matching the preconditions of templates to the postconditions present on the graph.

In contrast, a MPAG defines preconditions more simply than previous work: if an attacker can reach a host, they can exploit the remote exploits of that host. Their

stated reason for this is that the information available in databases is not of a suitable format and standard to justify the specific preconditions used by other methods.

MPAGs consist of three vertex types:

- **State vertices**: these correspond to the typical vertices in other attack graph methods
- **Prerequisite vertices**: these represent either reachability groups (i.e. sets of hosts that can be reached) or credentials (i.e. the attacker having access to a credential). These roughly correspond to conditions in other models; they lead to vulnerabilities.
- **Vulnerability instances**: these directly correspond to specific vulnerabilities.

Edges in the graph take the following forms:

- Leading **from a state to a prerequisite**. These indicate that compromise of that state gives access to that prerequisite. For example, the state corresponding to the full compromise of a host would have an edge leading to the reachability groups which that host is a part of.
- Leading **from a prerequisite to a vulnerability instance**. These indicate that the prerequisite enables the vulnerability to be performed. Typically this means that the attacker has access to the host the vulnerability is present on.
- Leading **from a vulnerability to a state**. This indicates the postconditions of the vulnerability.

2.2.3 Host-Based Attack Graphs

Despite improvements to graph complexity through assumptions such as monotonicity, attack graphs are still complex – to generate, to analyse and to understand. To address this, a variant of attack graph methods was developed: *host-based attack graphs*.

These differ from traditional (or "*network-based*") attack graphs by modelling each host separately and using the graph to model the relationships between hosts. This differs from previous methods, where a state could contain information about any number of hosts. By making this restriction, *host-based methods* avoid much of the complexity of other methods.

This approach was first used somewhat implicitly by *Ammann et al.* [APRS05], who began their construction process through an access graph, and later expanded and described by *Hewett and Kijsanayothin* [HK08]. *Xie et al.* [XCT+09] offer an explicit division between the modelling of the network and of each host, which persists through to their resulting two-level graph.

Ammann et al. (Host-Based Attack Graphs)

Ammann et al. [AWK02,APRS05] provide a model constructed via a different process to earlier methods. They begin with topology information, encoded as an "access graph" – each host is a vertex in this graph, and edges correspond to the highest level of network access available between hosts. This access graph is then converted into an attack graph by using vulnerability information to find the maximal level of access possible on each edge.

The resulting graph is therefore a *host-based* attack graph – vertices in the graph correspond to single hosts rather than entire system states. This is a crucial difference and prevents state space explosion in the number of vertices. To construct such a graph, the complexity of vulnerabilities is necessarily restricted. A discussion of the significance of this restriction is provided in Chapter 5.

(See also Section 2.2.2 for a discussion of this work's contribution to *monotonicity*.)

Hewett and Kijsanayothin (Host-Centric Attack Graphs)

Hewett and Kijsanayothin [HK08] adapts the work of *Ammann et al.*, primarily with respect to the host-based nature of their attack graphs. They convert the construction process to a model checking process, similar to the work performed by *Sheyner et al.* on typical attack graphs.

The key difference proposed in their work compared to that of *Ammann et al.* is that they model each possible interaction, instead of aggregating to highest privilege level. This results in modelling every possible attack path instead of a subset.

Xie et al (Two-Layer Attack Graphs)

Xie et al. [XCT+09] propose a hierarchical model composed of two layers. The first layer is referred to as the *host access graph* and the second is a collection of *host-pair access graphs*.

Each edge in the host access graph is associated with one of the host-pair access graphs. The attacker then moves through the model by traversing edges in the host

access graph, and traverses each edge by moving across the corresponding host-pair graph.

This results in a model very similar to that of *Ammann et al.*. The most notable exception is that in this work, the resulting graph maintains the two-part structure, whereas in the earlier work this was combined to give a single resulting attack graph. By preserving the two-part structure in the final result, *Xie et al.* provide a much simpler overview of the network, with a graph consisting of a comparatively small set of vertices and edges.

Separating the graph into components also restricts combinatorial explosion by not modelling interactions between privileges on different hosts. This performance increase potentially comes with decrease in modelling power, as discussed in Chapter 5.

2.2.4 Probabilistic Attack Graphs

Muñoz-González et al. [MGSPL17] observed that the uncertainty present in any estimation of attacker capabilities makes these models all intrinsically probabilistic.

Throughout attack graph research, many graph methods have incorporated probabilistic modelling of exploits. These vary greatly in complexity, from the initial methods of *Swiler et al.* [SPEC01] and *Wang et al.* [WIL+08] to the Bayesian models of *Frigault and Wang* [FW08] and *Poolsappasit et al.* [PDR12].

Primarily, these methods involve assigning success probabilities to each exploit and then combining them to calculate the probability of a successful attack. While early methods treat each exploit as independent, later methods (particularly Bayesian methods) build conditional probabilities into the model, so that dependencies between exploits can be captured.

***Wang et al.* (Probabilities on Attack Graphs)**

Wang et al. [WIL+08] take a dependency model and attach probabilities to each exploit in the graph. They then calculate security metrics by composing these probabilities.

The inclusion of probabilities enables a risk-centric assessment of attack graph security, and avoids over-emphasis of difficult or impossible exploits. However, this early approach ignores the relationships between exploits.

In an attack graph, there may be a single exploit represented in different places throughout the graph. For instance, performing the same exploit against two hosts

would resulting in two separate states. In the approach of *Wang et al.*, these two instances of the same exploit would be independent. In fact, it is more reasonable to assume that an attacker would be very likely to be able to perform one if they can perform the other. This is particularly consequential when the two exploits are performed sequentially, as the probability is applied twice.

Frigault and Wang, Poolsappasit et al. (Bayesian Attack Graphs)

Frigault and Wang [FW08,FWSJ08] offer a more complex take on probabilistic attack graphs by building Bayesian Networks to model the probabilities on the attack graph. The resulting Bayesian Attack Graph contains an attack graph (again based on a dependency model) and a collection of conditional probability tables (CPTs).

These tables encode the dependencies between the probabilities of the exploits. By capturing this in precise detail, they are able to precisely model the relationships between exploits that were not captured in earlier methods.

Poolsappasit et al. [PDR12] provide an extended and detailed model of Bayesian Attack Graphs. They construct a multiobjective optimisation problem to solve the problem of how administrators can make restricted hardening choices optimally – the *"system administrators' dilemma"* [DPRW07]. They propose a genetic algorithm to solve this using Bayesian Attack Graphs.

A key drawback to such precise modelling is the strenuous data requirements they impose. As the number of exploits in a model increases, the number and size of CPTs required increases. Further, it is not clear that existing vulnerability databases are at all suitable for these probabilistic assignments. Other methods highlight the problems with gathering even simple data from these databases (e.g. [ILP06,LIS+06]). Further, these methods impose significant computational complexity on the resulting analysis.

Muñoz-González et al. (Approximate Inference on Bayesian Attack Graphs)

Muñoz-González et al. mitigate the complexity of Bayesian methods through the use of approximate inference, and contribute a method based on Loopy Belief Propagation that scales linearly in the number of graph vertices.

This method trades off accuracy for performance, but the resulting loss of accuracy is small, especially in the context of the data used to populate the model.

2.2.5 Attacker Modelling on Attack Graphs

While the majority of attack graph methods assume a generic attacker, some methods examine the threat caused by specific attackers.

Dantu et al. (Attacker Profiling)

Dantu et al. [DKC09] use surveys to establish a set of three attacker profiles, each with distinct properties. These are used to update conditional probabilities tables and, hence, to generate three separate analyses for a single graph. The results of these can then be compared to examine the consequences between different attackers.

Nguyen et al. (Uncertain Graphs)

Nguyen et al. [NPN17] present a method where attack graphs are considered to be uncertain, and probabilities are applied to each edge. The relationships between these probabilities are expressed as conditional on a collection of modelling artefacts, similar to parameters [4].

2.2.6 Evaluation

Transition and dependency models

Whilst all attack graph models seek to capture the combinations of exploits an attacker can use to move through the network, the exact representation changes between variations.

In transition models, each vertex encapsulates the full state of the system and attack at a given time, and so the attack moves from vertex to vertex. In these models, edges represent transitions between states, often corresponding to the performance of an exploit by an attacker.

In dependency models, each vertex only represents part of the system; this might be a single exploit, or a single condition. The state of the system at a particular time is represented by some subset of these vertices, each of which represents one component of the state of the system. In these models, edges represent dependencies between conditions and exploits.

Figure 2.2 shows a very simple example illustrated for both variants. For an example technique that generates graphs in the transition form see *Swiler et al.* [SPEC01], and for the dependency form see *Noel et al.* [NJOJ03].

[4]This work was published after the publication of my paper *Attacker-Parametrised Attack Graphs* [JvRNG16] which contains my work on parameters, as presented in Chapter 7

(a) Transition variant - each vertex represents a state of the whole system, each edge a transition between states.

(b) Dependency variant - small "dot" vertices represent exploits, larger vertices represent single properties of the system. Edges represent dependencies.

Figure 2.2: The same set of exploits shown through representative models of transition and dependency attack graph models. In both cases, the goal is reached through either the exploitation of one exploit or the combination of two others.

The small size of this example does not illustrate the growth of these two graphs; the number of required vertices in a dependency model is typically lower than that of the transition models, because each represents less information – the whole state of the system is represented by a set of vertices instead of the single vertex of a transition model. This leads un-optimised transition models to grow exponentially, although most techniques overcome this and there are a variety of ways to do so (see, for instance, host-based models [APRS05], and Chapter 5).

A full attack from one state to another can be represented as a path in a transition attack graph, with a sequence of exploits represented by a sequence of edges and vertices. In a dependency graph, a full attack from one state to a target property is represented as a tree. This is illustrated in Figure 2.3, where the same attack is represented in both variants. Attacks cannot be represented as paths in dependency models because of the presence of logical *and* operators across edges leading to an exploit: the attacker must achieve multiple conditions before being able to proceed with the attack, which are represented in different vertices.

Generalised process for transition models

Transition-based attack graph generation follows a general process: initially, data sources are gathered and formatted appropriately. Then, this data is used to create a set of templates, which dictate how the edges and vertices of the graph should be formed. Finally, the graph is then constructed through the application and instantiation of these templates.

Once this is complete, the graph may be pruned or reformatted, usually to reduce complexity. This outline is similar to that of *Kaynar et al.* [Kay16], which contains the

(a) Transition variant

(b) Dependency variant

Figure 2.3: As Figure 2.2, but with a possible complete attack highlighted. Note that for the transition variant, the attack is simply a path in the graph. For the dependency variant, the attack cannot be represented as a path.

phases *reachability analysis*, *attack graph modelling* and *attack graph core building*.

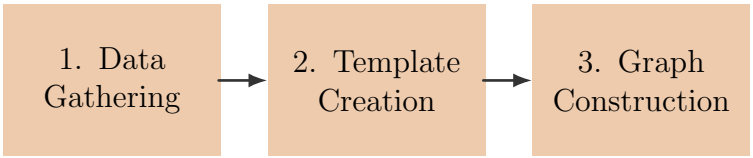


Figure 2.4: The stages of attack graph generation.

Data gathering is the first part of the process. The availability of information, and the quality of that information, controls how much can be modelled later in the process. This means by which this data gathering is performed is a practical concern, however, and so is out of scope for this work.

Template creation, where templates for vertices and edges are constructed, is the most relevant and important part of the process for my work. In this component of attack graph generation, a set of templates are built, each of which dictates the conditions under which an edge or vertex can exist on the graph. In this way, the graph is determined solely by the templates used. Edges in the graph are determined by the situations in which a templates preconditions and postconditions match vertices on the graph, and vertices themselves are added as a consequence of the templates.

Graph construction involves the creation of the final graph from the templates. Typically this is the repeated application of the templates to the partially-constructed graph. Upon each application, the template's preconditions are tested against the graph and, where satisfied, new edges or vertices are added according to the post-conditions. Templates are usually matched with a vertex, and then result in a new vertex with an edge from the matching vertex to the new vertex. A primary concern

at this stage is performance, and algorithms are designed to construct the graph in the least time. For some methods, this involves excluding template instances when they are considered to be irrelevant or otherwise unsuitable. This leads to smaller graphs, which is beneficial for complexity, both computationally and visually.

Describing the construction process in this way separates the three primary challenges in attack graph construction. First, gathering data that is comprehensive and accurate is difficult, and often relies on existing databases which are not necessarily suitable. Second, selecting how vulnerability information is converted into edges and vertices is a theoretical challenge, and structuring the graph differently has significant consequences for modelling power and complexity. Finally, constructing the graph requires efficient algorithms that build or approximate the full attack graph.

2.3 Attack Graph Analysis

Once an attack graph has been constructed, it must be analysed so that security conclusions can be made. In the literature these are often presented together, with a research paper providing the means by which a graph is generated and then techniques for how it can be analysed. By presenting these separately, we can decouple the two components of attack graph work, and consider analysis techniques in a generic sense, so they may be applied to a variety of types of attack graph.

2.3.1 Graph Metrics

One of the most common ways to derive security information from a graph is to apply a metric. In this case, and agreeing with the general definition, a metric is a map from an attack graph to a value, typically a number. This value is interpretable as a representation of the security of the system being modelled.

Metrics cover a wide variety of aspects, and numerous surveys exist (e.g. [RLFR17, HKCH17, Kay16]). In this section I provide a selection of metrics from related work.

Some of these metrics relate to both dependency and transition graphs, despite the differences in structure between these graph models. To enable this, I use the generalisation of paths to *attack sequences*[5]. An attack sequence is a sequence of exploits carried out, in order, by an attacker: in a transition graph this is represented by a path, in a dependency graph this is a tree rooted at the goal. The work cited may refer exclusively to paths, but the concepts can be generalised to either technique.

[5]Attack sequences are also called exploit sequences by some authors (e.g. [BG12]). I opt for the term *attack sequence* for consistency with *attack graph* and (as defined later) *attack suite*.

Initial Metrics

Initial metrics aimed to capture intuitive properties of security as quantified values from the graph. These metrics take an oversimplified approach to assessing security and, as such, many subsequent metrics build upon the same principles in a more nuanced fashion.

Shortest Sequence [PS98]. The security of the system is estimated via the length of the shortest attack sequence that successfully reaches the goal. This is based on intuitive notions of security: that the network is as strong as it's weakest link, and that requiring more exploits to be performed makes a sequence harder for the attacker. While these assumptions underpin later metrics, the shortest sequence does not take into account the consequence of having other sequences available, nor does it account for the relative difficulty of exploits on the graph.

Number of Sequences [ODK99]. The security of the system is estimated via the number of attack sequences that lead to the goal. The intuitive notion here is that giving the attacker more options provides them with more ways to succeed. Later work still assumes a correlation between security and the number of sequences (e.g. [WSJ07b]). Again, however, this metric does not take into account the relative difficulty of different exploits or sequences.

Sequence Metrics

These metrics are calculated by considering sequences on the graph and aggregating them. In the cited works these may be referred to as path metrics, but they are described here in terms of sequences so that they apply equally to dependency graphs and transition graphs.

Statistical Sequence Length Metrics [IB12]. In response to problems with earlier sequence-based metrics, a number of statistical metrics have been proposed. These attempt to better capture the different sequences in the graph, through measures such as standard deviation, mode and median. While these give some additional detail about the set of sequences, they do not account for relative exploit difficulty.

Minimum Exploit Cost [BG12]. Similar to a weighted shortest-path metric, the attacker's cost of exploitation of each exploit is calculated and summed to find the cost of an attack sequence. This metric assigns the each graph the value

of the lowest exploit cost of any attack sequence. This enables the metric to account for relative difficulties, but requires more information about the exploits.

Start- and Goal-State Metrics

The metrics above assume fixed initial and goal states. Some metrics assess a graph based on paths leading to and from possible initial and goal states.

Network Compromise Percentage [LIS+05,LIS+06]. The Network Compromise Percentage (NCP) of a network is the proportion of hosts that can be compromised by the attacker. NCP can either be a direct proportion of the hosts, or weighted according to host importance or criticality [LIS+05]. This can be viewed as the proportion of goal states that can be compromised.

Weakest Adversary [PJAS06]. This metric describes the security of the network in terms of the weakest adversary able to compromise the goal. *Pamula et al.* [PJAS06] define this as the least set of initial attributes necessary for an attacker to reach the goal. In this case, these initial attributes are conditions of the system. These initial attributes form initial states, so this metric seeks to measure the security of the graph based on the weakest initial state.

Probabilistic Metrics

Many metrics include probabilistic information in order to account for the relative difficulty of each exploit in the graph. By including this, the model can be risk-centric and assess the network via the probability that an attacker can reach the goal.

Compromise Probability [HZO+13]. The probability that an attacker can reach the goal from the initial state. This is typically calculated as a sequence metric, where each sequence is evaluated by the probability that an attacker can perform all of its exploits. Each exploit may be assumed to be independent [WIL+08], though more recent work avoids this assumption [HZO+13]. Many authors use CVSS [CVS] data to create probabilities [XLO+10]. Some work uses attacker models to derive multiple sets of probabilities [DKC09]. This metric combines well with impact information to calculate risk and provide an absolute measure of network security.

Bayesian Network Metrics [MGSPL17, FWSJ08, PDR12]. Bayesian network methods have been applied to attack graphs in order to further capture and model the dependencies between exploits. By fully modelling the possible combinations through conditional probability tables, these models allow finely-specified probabilistic relations between each exploit in the graph. The increased complexity incurred through these methods has been reduced from exponential to linear in recent work, through approximate inference techniques [MGSPL17].

Zero-day Metrics

The vast majority of attack graph methods consider only known vulnerabilities. Some work has attempted to extend attack graph techniques to the consideration of unknown vulnerabilities.

k-Zero Day Safety [WJS+14]. This work constructs attack graphs that include known vulnerabilities and possible zero-day exploits. Possible zero-days are derived from connectivity, existing services, and existing privileges. The metric value is based on the number of these zero-days that would be required in order to compromise the system. This has been used as a basis for network diversification techniques [BWJS16].

Network Diversity [ZWJ+16]. The above work was also used to propose a set of diversity metrics, which measure the security of the network through its diversity and hence resistance to zero-day exploits.

Bayesian Models [MGSPL17, XLO+10]. Some Bayesian methods incorporate a "leak factor" which captures the probability that a vulnerability scanner incorrectly reported an exploit as impossible, representing zero-days on the network.

The number of metrics available to apply to attack graphs is significant, and those presented above are only a small sample intended to represent the existing work. This is beneficial because typical analysis will involve the application of a multitude of metrics that can be aggregated and considered together. Consequently, a key problem in using metrics is not that suitable metrics do not exist, but that they are made hard to select and justify by the breadth of available choice. As a result, I do not focus on proposing novel metrics in this thesis, but instead aim to provide methods to assess validity of existing metrics (Chapter 4), and to provide novel techniques which maintain compatibility with, and extend, existing metrics (Chapters 7 and 8).

2.4 Smart Home

Throughout the thesis, I use the example of the smart home to explain, illustrate and motivate the methods presented. This example was chosen because of its emerging relevance, the comparatively-little attack graph work concerning it, and its particular suitability to the techniques proposed.

This is not intended to be prescriptive, and the techniques equally apply to other systems, including typical enterprise architectures.

2.4.1 Characterisation

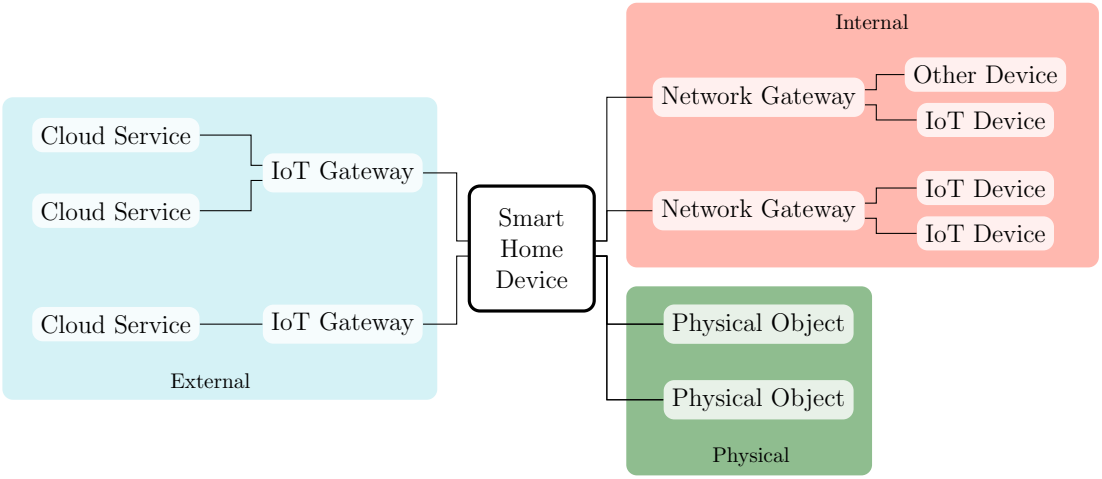


Figure 2.5: Visualisation of the connections made by a device in a smart home

The smart home is an example of the Internet of Things (IoT) paradigm applied to a domestic environment. The smart home includes networked and connected elements such as sensors, appliances, lighting, monitoring, alarms, and media systems, for purposes such as remote control, automation, adaptation to preferences, energy efficiency, assisted living and entertainment [TCC17, BODBW13, AIM10]. A key consideration in this area is not only the devices themselves but the means by which they interoperate [NCR17], with common standards ranging from the physical connection layer through to application standards. These connections take a variety of forms [NCR17, CWG+14] (shown in Figure 2.5), including interactions with physical objects such as sensors and switches, and communication with cloud services or local devices. Local device connections utilise a variety of protocols [CWG+14], such as

WiFi, Bluetooth, ZigBee [Zig18], and RFID. Cloud connections either directly connect to base stations via cellular connections or are part of capillary connections in which their traffic is passed through a local gateway [CWG+14].

The number of devices in the smart home of the future is not certain. One study [FJP16] (published 2016) asked respondents how many devices were present in their smart home and found that some users reported up to 100 devices, though the majority had less than 50. A survey of publicly available datasets [AEIE13] of smart home sensor deployments found studies used between 14 and 86 devices, although these only included sensors. In this thesis, for the purpose of considering the complexity of smart home methods, I will assume the number of devices is on the order of hundreds, as an upper bound on estimates in the related work. Because my work is intended for a smart home of the near future, it is likely the number of devices will have increased. These device counts often also include many identical devices (e.g. many identical lightbulbs or sensors); for the purpose of security analysis these can often be treated as a single device, reducing the complexity significantly.

2.4.2 Security Considerations in the Smart Home

As with all IoT applications, security considerations in the smart home are set apart from those of traditional networks by a number of important considerations:

Dynamic nature

IoT environments are considered to be constantly changing, with new devices entering and leaving the system frequently. This is a significant challenge for traditional risk analysis [NCR17], which relies on periodic updates. Methods which rely on risk analysis that can only be performed manually will become outdated as soon as the system changes. As a consequence, IoT risk analysis must be updated regularly and with minimal manual input. This is especially important in the smart home scenario (as opposed to, for example, industrial IoT), where the user cannot be assumed to have any expert knowledge.

Device ubiquity

Projections anticipate connectivity permeating almost every part of the smart home environment. This ubiquitousness of devices has a number of consequences for security. With a large number of devices, the ability to fully understand all of them becomes difficult. This also has consequences for connectivity: a key benefit of the

Internet of Things is the dynamic interoperability between its components [NCR17]. This flexibility of connections makes the smart home less suitable for static firewalls and reduces the trust that can be built between devices.

To reduce the costs of these devices and to minimise energy consumption, they have limited computational power [AIM10] and are therefore less-suitable for host-based security mechanisms, increasing the need for network-based defences and secure design.

Limited knowledge

Exacerbating problems caused by the number of devices, there are doubts over how much knowledge we can have over the devices present in the network [NCR17]. The high rate of change of devices and standards means that there is less time for knowledge to be obtained before it is outdated, and the heterogeneity of the components of the system makes knowledge-gathering more difficult.

This limited knowledge is a problem for risk assessment, and in particular graphical methods, which often depend on considerable amounts of specific data relating to topology, assets and vulnerabilities. Without this data, or in situation where it is less than accurate, many existing methods are not applicable.

For example, attack graphs based on CVE data can only be constructed when such data is available: not only must vulnerabilities be reported on the relevant databases, but vulnerability scanners must detect them on the system. *Chothia and de Ruiter* [CdR16] used the discovery of new vulnerabilities in IoT devices as a learning tool, with student groups successfully finding critical vulnerabilities in devices with no known weaknesses. This indicates both the relative ease of finding such weaknesses and the lack of reliable reporting.

Shared vulnerabilities

Shared standards and protocols offer a plethora of benefits to IoT applications. Standards exist throughout the network stack and can be applied across devices. This reduces the work required to develop each device and enables interoperability. Different IoT devices could implement or use the same protocols, programming frameworks, libraries, hardware, specialised operating systems, or standards [LS14].

But vulnerabilities in any one of these could act as shared points of failure between otherwise entirely distinct devices [BODBW13]. Exploits released against widely-

adopted standards often have significant consequences [6], and the impact of such an exploit would be exacerbated in the Internet of Things context where many devices share the same traits.

For example, a vulnerability assessment of IoT devices accessible over the internet by *Williams et al.* [WMS+17] found that the most common critical vulnerability for webcams was caused by a weakness in the MiniUPnP protocol they implemented. *Fernandes et al.* [FJP16] found a set of serious vulnerabilities in the way privileges were handled in a common programming framework, affecting all devices that implemented that framework.

System criticality

While the importance of computer networks in general is indisputable, the Internet of Things and smart home entail further critical dependencies. Use cases such as medical devices and assisted living require consistent security, and other applications such as maintaining heating systems during winter are important [BODBW13].

Less important components may become vital because of the dependence of others upon them. A remote control might not be considered crucial, unless the devices it controls cannot be used without it [BODBW13].

Assets as attack platforms

Nurse et al. [NCR17] describe the importance in the IoT context of considering assets not only as targets, but as platforms from which attackers can launch further attacks, both internal and external. This consideration is made more important by the dynamic nature of IoT systems, where assets can be compromised by attackers before being brought into the system and unwittingly facilitating further attacks.

2.4.3 Attack Graphs

There is little work that focuses directly on the application of attack graph theory to the smart home. *Agadakos et al.* [ALC+17] use model-checking methods to analyse the security of the IoT, particular with regard to how non-network attacks can be used. *Ge et al.* [GHGK17] provide a framework that uses the Hierarchical Attack Representation Model (HARM) [HK12] to analyse the internet of things.

The benefits of attack graphs match well with the problems faced by IoT deployments. In particular, attack graphs consider all combinations of exploits and devices,

[6] For example, the Heartbleed vulnerability in the widely-used OpenSSL library allowed unauthorized access to between 24-55% of popular HTTPS websites [DLK+14]

which makes them well-suited to situations where any device could be considered critical, and could be used to launch further attacks. The number of devices in smart home IoT is increasing, but is still well-below the point where attack graph methods would face computational problems – most concerns are based on the application of graphs to corporate networks, which might contain many thousands of devices. As a consequence, even comparatively inefficient methods may be suitable in the smart home context.

The dynamic nature of the smart home network means that any security analysis should be automated, and in particular, should be performed without expert input. Attack graphs can be constructed to satisfy this criteria.

The primary concern when applying attack graphs to the smart home context is the limited knowledge that vulnerability databases contain regarding these devices. Many vulnerable devices may be considered secure when purely relying on known, reported vulnerabilities, despite the relative ease with which vulnerabilities could be found. As a result, attack graph analysis dependent on these datasets will believe the system to be more secure than it is.

2.5 Critical Analysis

I identify the following commonalities and gaps in existing work, which I will attempt to address in this thesis:

2.5.1 Data sources

All attack graph generation papers surveyed in this chapter depend upon vulnerability data; that is, each technique requires the existence of an authoritative (and ideally near-exhaustive) database of vulnerabilities present in the system.

Existing online databases, such as the National Vulnerability Database (NVD) [NVD] is often referenced as a source of this information (e.g. [HK08]). Other work refers to the Common Vulnerability Enumeration (CVE) as a means for formatting such data (e.g. [SW04]); the similar Common Vulnerability Scoring System (CVSS) format, is often referenced as a source of probabilistic data (e.g. [MGSPL17,WIL+08]). Other works depend on, or assume the use of, vulnerability scanners, which typically connect to online databases to gather their results (e.g. [ILP06,HK08,AWK02]). Early work, in particular, suggest the use of databases without particular specification of which (e.g. [SHJ+02], which calls for the creation of such a database, and [PS98],

which was published two years before the creation of the NVD in 2000). One paper cited refers to the manual construction of vulnerability information [NPN17].

The notable exception is zero-day metrics, which attempt to understand zero-days through means other than direct vulnerability information (because, naturally, no such direct vulnerability information exists for zero-days). These works typically construct an attack graph using typical vulnerability data and then augment it with an additional model that attempts to capture unknown vulnerability: by studying connectivity and existing services, and existing privileges [WJS+14] and by modifying Bayesian methods to allow for incorrect reporting from vulnerability scanners [MGSPL17, XLO+10].

For work studying non-zero-day attacks, the significant dependence on vulnerability databases presents a weakness present in almost all surveyed work. Vulnerability databases, while containing considerable amounts of information, are not wholly suitable as evidenced by existing work [SS15]. Particularly in the Smart Home, relying on an online database results in incomplete information; this is demonstrated by the data set gathered in Section 3.2.1, for which many vulnerabilities could not be linked to corresponding CVE entries.

CVE and CVSS data in particular has been criticised before, in part because it does not align well with how it is often used. *Allodi and Massacci* [AM14] demonstrate that CVSS does not contain a proper characterisation of "likelihood of exploit", and that patching vulnerabilities directly based on CVSS scores was equivalent to randomly picking vulnerabilities to patch. More general criticism points out the subjectivity of CVSS scores, coupled with a lack of clear definitions or standardisation within scores [YMR16].

Overall, this gives a strong indication that analyses relying on such data should be careful to justify their reliance on it; techniques which avoid such data while still providing useful output would be greatly beneficial. Manual data generation, in particular, is unlikely to be feasible when considering networks that may consist of hundreds of thousands of devices.

In particular, techniques which rely on CVSS data to generate probabilities (e.g. [MGSPL17, WIL+08]) must be particularly conscious of these drawbacks, and there is a need for methods of alleviating the strict data requirements that these methods have. Due to the high number of probabilistic assignments that many techniques require (particularly when assigning conditional probabilities between sets of events), it may be necessary to rethink how these methods operate, rather than simply choosing a new data source.

2.5.2 Performance

From the extreme complexity of early methods [SHJ+02, PS98], attach graph techniques have made significant progress towards demonstrating reasonable construction and analysis times.

Some techniques have demonstrated near-linear performance on large networks (e.g. [ILP06]), others use the network topology to constrain the growth of the system through techniques such as host-based attack graphs [APRS05] and through multi-tiered graphical models such as Two-Layer Attack Graphs [XCT+09].

Many of the works cited explicitly credit the assumption of monotonicity (first presented [AWK02]) for controlling complexity (e.g. [ILP06, PDR12, AJ18]).

There is comparatively little discussion of the consequences of monotonicity, or whether it is a necessary (or indeed sufficient) assumption for controlling complexity. Some work consciously avoids the use of monotonicity (e.g. [ZZD+12]) due to modelling non-static defences.

Due to the importance of this assumption, and its ubiquity, it is important that it is well-understood; this will enable the claimed benefits to be fully justified and any drawbacks to be appreciated and accounted for.

2.5.3 Analysis

Early work explicitly stated the importance of modelling the attacker [PS98], and drew a distinction between the behaviour and the capability of an attacker.

Despite this, few techniques surveyed account for attacker models. *Dantu et al.* [DKC09] provide a technique which depends on a small set of predefined profiles, specified from an online survey, and other work has introduced modelling artefacts that attempt to capture generic properties of attackers [NPN17].

Other work relies on a singular, implicit model of an attacker. In deterministic methods, this attacker is often considered able to perform any exploit in the system and, further, to navigate through the system as if they had perfect knowledge (for instance, in metrics such as shortest path). This is reminiscent of the Dolev-Yao [Dol83] attacker from cryptographic protocol analysis, and seems unsuitable to the less-formal analysis of network security.

Assuming a near-omnipotent attacker results in metric summaries that are highly critical of the security of the network, predicting that any attacker can proceed down a number of short paths to their goal.

Probabilistic methods enhance this by adding a probabilistic layer between the attacker and the system, resulting in an attacker who is essentially *sometimes* omnipotent and *sometimes* impotent. Each exploit is randomly (through a probability distribution of varying complexity) either always possible or always impossible.

Such analysis does not factor in the depth and variety in real-world attackers; in terms of neither the behaviour of an attacker or in terms of the capability of the attacker. While attacker behaviour is complex and perhaps out-of-scope for most attack graph work, capturing the capabilities of attackers must be considered a critical component of any reasonable network security analysis.

2.6 Conclusion

Throughout the development of attack graphs, performance has been a considerable concern. While much progress has been made from the initial methods of *Swiler and Philips* [SPEC01] and *Sheyner et al.* [SHJ+02], transition models in particular still have limited methods to deal with state-space explosion in large graphs. Of these, host-centric models (e.g. [APRS05], [HK08]) offer promising improvements, but they restrict the model to solely focusing on hosts, and their drawbacks have not been fully considered. The literature leaves open the question of what precise assumption is necessary to control complexity, and the investigation of this forms the basis of the work in Chapter 5.

Aside from difficulties in computationally running attack graph models, they are also highly dependent on data sources. All of the attack graph models discussed above are strictly dependent on vulnerability information, often from sources such as the National Vulnerability Database (NVD) [NVD]. These data sources have not been demonstrated to be fully suitable and empirical evaluations of existing tools have shown them to be highly unreliable and inaccurate [SS15]. In Chapter 6, I propose a method of building attack graphs that does not directly rely on vulnerability information, circumventing this problem entirely.

Of particular concern are probabilistic assignments, which must be subjectively inferred from these data sources. This is worsened in more sophisticated methods, such as Bayesian Attack Graphs [Fri11], which increase the dependency on probability and increase the number of probabilistic assignments required. To address this, Chapter 7 offers an alternative method of assigning probability that drastically reduces the number of required assignments.

Once attack graphs are constructed, they must be analysed in order to produce actionable conclusions. Many metrics exist, but few, if any, have been suitably empirically tested. Consequently, it is difficult to justify security decisions based on these metrics, and therefore difficult to justify the use of attack graphs in practice. Chapter 4 provides a theoretical technique to invalidate some metrics and produce counterexamples, enabling researchers to critically consider metrics.

Chapter 3

Foundations

In this chapter, I will introduce and define the basis of the rest of this thesis. The purpose of this chapter is to establish definitions and ensure that later chapters are built upon clear definitions. In order to make my work more comparable to others, I also provide a list of the examples that I will use throughout the thesis, so they can be explained in more detail here before being used later.

3.1 Attack Graphs

3.1.1 Definition of Attack Graph

As evidenced by the literature review, the definition of attack graph varies between authors. In this subsection I will state, explain and justify my definition of attack graph that will be used throughout this thesis.

Templates

The construction of an attack graph begins with a set of attack templates. Each of these templates represents an atomic action or state change in the system. Often, templates directly correspond to an action by the attacker, most typically performing an exploit against a weakness in the system.

Templates are defined by their *preconditions* and their *postconditions*. These, respectively, dictate the necessary conditions on the system for the template to be performed, and the conditions that occur as a result of it being performed.

Each condition is a boolean (that is, either true or false) fact about the system and the attacker being modelled. These can relate purely to the system – for instance, whether or not a particular service is running – or they can relate to the attacker's relationship to the system – for instance, whether or not the attacker has access to

a given host. A combination of conditions is a state of the system, and a system state can be fully described by the conditions which are true in it. Hence for a model containing n conditions, we have 2^n states. I will use $\mathcal{S}$ to refer to the set of all states in the model, i.e. the power set of the set of conditions.

Template pre- and post-conditions can be specified in various ways. In general, preconditions may use logical operators to specify precisely which combinations of true and false conditions are sufficient for the template to be performed. In some instances, assumptions are made to restrict how these can be formed. In particular, it may be assumed that preconditions cannot require conditions to be false [AWK02]. In this work these assumptions are not made unless stated otherwise (notably, Chapter 5 considers these assumptions in depth).

Postconditions involve the addition or removal (or both) of conditions from the state. Some assumptions restrict postconditions to exclusively add conditions to the state [AWK02]. Again, these are not made in this thesis unless stated otherwise.

Formally, I define a template t as a map $t : m_t \longrightarrow \mathcal{S}$. m_t is the "matching set" of t: the set of states to which t can be applied and therefore the set of states which are said to match the preconditions of t. The state $t(s)$ is the state resulting from the application of the postconditions of t to the state s. I use $\mathcal{T}$ to refer to the set of all templates.

This definition is broad enough to include templates defined in a typical way. For a precondition defined using a logical expression on the set of system conditions, we can define m_t to be the set of states for which the expression is true. For a postcondition that adds some subset of conditions A and removes another such subset B, we can define $t(s) := (s \cup A) \setminus B$.

As an aside, we can also include some more peculiar templates, such as $t : s \mapsto s^{\mathrm{c}}$, where s^{c} is the complement of s in the set of all conditions. Such templates are unusual, and violate commonly-held assumptions such as monotonicity [AWK02]. However, they may be of interest when applied to atypical systems, such as systems that interact with the physical world.

Attack Graph

From a set of templates $\mathcal{T}$, we can construct the attack graph. The *full* attack graph is a graph, $G = (V, E)$, with vertex set V and edge set E. V contains a vertex for each possible state in the system. For convenience, and because there is a direct correspondence between states and vertices, I may refer to a state and its vertex interchangeably.

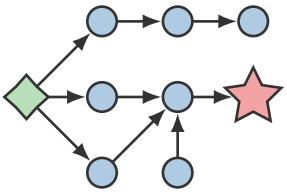
The edge set, E, is defined from the set of templates $\mathcal{T}$:

$$E = \bigcup_{t \in \mathcal{T}} \{(v, t(v), t) \mid v \in m_t, v \neq t(v)\}$$

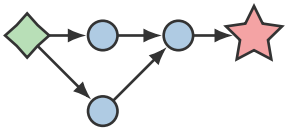
Each edge is a triple (v_1, v_2, t) consisting of two vertices (the start v_1 and end v_2) and a label, which is the template from which the edge was derived. Accordingly, G is a directed, labelled multi-graph with no loops. We require $v \neq t(v)$ so that the graph contains no loops, because such a template would represent the attacker performing an action which has no impact on the state of the system.

For practical purposes, it may be desirable to reduce the number of vertices and edges of the graph. This gives rise to *partial* attack graphs, which are subgraphs of the full graph – some vertices and edges have been pruned (i.e. removed). This is typically when attack graphs are not generated completely for performance reasons, so that vertices or edges of the graph are removed to simplify the graph. These removed elements may not be redundant, but may be removed despite this.

Figure 3.1 gives an example of a full attack graph and one of the possible partial attack graphs. While information is lost during this pruning process, it can be done in such a way that it does not affect later analysis. In this example, while four of the nine vertices have been removed, all paths from the "start" vertex to the goal vertex have been preserved.



(a) The full attack graph, containing all vertices and edges.



(b) A possible partial attack graph.

Figure 3.1: The same system modelled with a full and partial attack graph. The partial attack graph is a subgraph of the full graph, with extraneous vertices removed. The precise definition of extraneous varies. Here, vertices that have no path to the red vertex have been removed, together with vertices that have no paths to them from the green, "start" vertex.

This definition of attack graph is intentionally non-prescriptive; the context is separated from the model and the same mathematical basis can apply to many dif-

ferent scenarios. For example, typical attack-graph work relates to the exploitation of computer vulnerabilities across a network. In these cases, each template directly represents a software vulnerability, which may be taken from a vulnerability database. But this need not be the case (from the perspective of the definition, at least) – in some work, templates would correspond to actions performed by defenders instead of attackers, or random events that may occur by chance. They may also correspond to less traditional computer attacks, such as social engineering, physical tampering, or the use of side channels [ALC+17].

Start Vertices

Each attack on the system must start somewhere. This is formalised as a *start* (or *initial*) state. This state corresponds to the system before any actions have taken place. Without loss of generalisation, we can assume that conditions on the graph are defined such that they are false in the initial state. Consequently, the initial state is the state for which every condition is false.

It may seem appropriate to create multiple initial states corresponding to potentially different situations that the network may be in. Attackers such as malicious insiders may begin with initial privileges that advantage them. In this case, the attacker approaches the network, and hence the graph, from a different perspective, and has attack paths that are unavailable to other attackers.

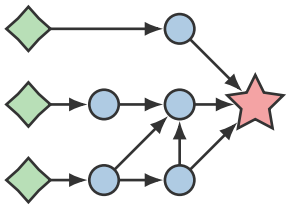
The system, too, may undergo non-malicious (and perhaps uncontrolled) changes that affect the weaknesses present in it. For example, services may be unavailable at certain times. These affect the attack graph and could be modelled as separate initial states.

In this thesis I opt for a single initial state. Instead of multiple initial states, I model the differences in attackers through alternative means (see Chapter 7). Differences in the initial state can satisfactorily be modelled via templates that match the initial state, i.e., they are modelled as transitions that occur without (and perhaps before) the attacker (shown in Figure 3.2).

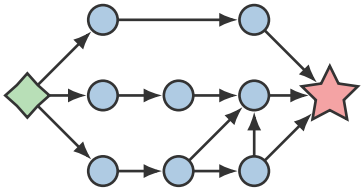
Goal Vertices

Many formalisms of attack graphs use the concept of a goal vertex. This is an intuitive way of orienting the attack graph: the attacker begins at the start and tries to reach the goal. However, it does not necessarily align well with attacker behaviour.

Defining a single goal assumes the attacker has only a single objective. Attackers may wish to impact the network in any of a number of ways, and so picking a state



(a) A graph with multiple initial vertices (as green diamonds), representing different possible initial states of the system.



(b) The same system being modelled, but where the initial states are reached by transitions from an artificial initial state.

Figure 3.2: Different methods of representing a system with three possible initial states. In this thesis, I opt for the representation shown in **(b)**

as their goal does not accurately represent them. Even in cases where the attacker has a single objective, multiple system states may satisfy their goal, and so there may be multiple points on the graph from which this is achieved. In fact an attacker may not have any goal in mind at all, and so expecting them to move towards a target is inaccurate.

On the other hand, it is convenient for many metrics to have a single goal vertex. This can then be used to calculate metrics such as "shortest path to goal" [PS98], or "number of paths to goal" [ODK99].

Fortunately, this can be decided on a case-by-case basis as the application demands. The existence of one or more goal states is not integral to the graph construction process, and when analysis methods require goal states a suitable vertex can be chosen as appropriate.

In the examples in this thesis I will typically include a single goal state, as this is a convenient way to intuitively reason about the graphs.

3.2 Motivating Example

While my thesis aims to approach attack graph modelling from a theoretical standpoint, it is important to consider the practicalities and applications of any theory during its development. Accordingly, I provide the *smart home* as a motivating example for my work.

Typical attack graph use cases model enterprise networks, and contain sets of hosts such as desktop computers and servers. The ubiquity of networks of this form justifies their focus in attack graph research, and the generic nature of such a use-case makes it transferable to many other scenarios.

Less work has focused on the application of attack graph modelling to more specific models of networked systems. In particular, I believe that the Internet of Things (IoT), particularly in the home environment, is an interesting, illustrative and highly suitable application of graphical security modelling.

3.2.1 Smart Home Example

To provide a consistent example for use throughout the thesis, I have collected a set of devices (with corresponding vulnerabilities) to use as an illustrative and explanatory dataset.

Methodology

The devices in this set were found by first creating a list of categories of smart home device. This list was gathered from related works [TCC17, AIM10, AEIE13]. The categories were as follows: Appliances, Cameras, Controls, Doorbells, Heating, Hubs, Lights, Locks, Media, Security, Sensors and Solar. The aim of selecting devices in this way was to produce a dataset that could reasonably correspond to the devices in a single well-connected home.

The devices were found by finding vulnerabilities through searches on vulnerability databases [NVD], and by searching for vulnerabilities in popular devices. When vulnerabilities were found, the devices they corresponded to were added to the dataset. While the majority of devices are suitable for the smart home, in order to populate the dataset with meaningful examples, some devices intended for office environments were included.

This methodology has a number of clear drawbacks that limit the conclusions that should be drawn from it. Most importantly, the devices in the dataset are chosen specifically because they are vulnerable, and hence are not necessarily representative of prevalence of vulnerabilities. I am only concerned with devices that contain known vulnerabilities – including devices without known vulnerabilities would not affect my work.

Further, the list of vulnerabilities is by no means exhaustive in the number of devices or vulnerabilities that exist, and only represents a small part of the large number of possible devices. It is also worth noting that many, if not all, of the vulnerabilities were described as having a patch available that resolves them.

These drawbacks should not limit the dataset from its intended use, as an illustration and example of typical exploits.

Results

The dataset contains 26 devices and corresponding vulnerabilities, with at least one device from each of the categories defined above. Some vulnerabilities affect multiple devices of different types. The full set of vulnerabilities is listed in Table 3.1. I have assigned the vulnerabilities an arbitrary ID, written `SHV-n` [1], for ease of reference in this thesis. These are listed by device, and in some cases multiple devices were susceptible to the same vulnerability.

ID	CVE, CWE or Source	Device Type
SHV-1	CVE-2018-11689	Media
SHV-2	CVE-2015-4400	Doorbell
SHV-3	CVE-2018-11629	Locks
SHV-4	CVE-2018-11629	Heating
SHV-5	CVE-2018-11629	Sensors
SHV-6	CVE-2018-11629	Controls
SHV-7	CVE-2017-14797	Lights
SHV-8	[Cha14]	Lights
SHV-9	[WTFB08]	Locks
SHV-10	[WTFB08]	Locks
SHV-11	CVE-2017-16867	Locks
SHV-12	[OK14]	Media
SHV-13	[Nar18]	Media
SHV-14	CWE-636	Security
SHV-15	CWE-307	Security
SHV-16	CVE-2015-2886	Cameras
SHV-17	CVE-2015-2889	Cameras
SHV-18	CVE-2015-2887	Cameras
SHV-19	CVE-2018-12258	Cameras
SHV-20	CVE-2018-12323	Cameras
SHV-21	[Ven15]	Appliances
SHV-22	CVE-2012-5862	Solar
SHV-23	[CKY18]	Cameras
SHV-24	[CKY18]	Lights
SHV-25	[CKY18]	Hubs
SHV-26	[CKY18]	Cameras

Table 3.1: The vulnerabilities of the smart home dataset

Approximately half the device vulnerabilities have a corresponding CVE ID (14 of 26). The remaining vulnerabilities were found in research papers, technical reports or

[1] for Smart Home Vulnerability

penetration testing reports and no corresponding CVE ID could be found. Although the methodology of this sample precludes a conclusive statement, the lack of CVE IDs for a significant number of these exploits does shed some doubt on the completeness of methods which rely solely on vulnerability databases.

3.3 Baseline systems

Throughout this thesis it will be necessary to use examples of attack graphs to explore, explain and evaluate the techniques and methods proposed. To make this consistent throughout my work, I first present a set of "baseline" systems.

Attack graph research is typically evaluated using systems created for that particular piece of work, making it challenging to compare different methods. To better enable comparisons between components of my work, I will aim to use the same system descriptions where possible.

3.3.1 Types of baseline system

The diversity of real-world systems means that it is suitable to select a number of different types of baseline systems, each chosen to represent a particular variety of network. To assist this, I propose three categories of baseline system.

Illustrative systems

These systems give rise to small attack graphs, that could be built by hand. These can be used for illustration purposes in research methods. These graphs typically only consist of a small set of vertices and edges. Because these are mostly for the purpose of providing examples, the primary concern with these graphs is simplicity and illustrative power. Consistency between illustrative graphs in different works (or in this case, throughout the thesis) makes comparisons easier and makes novel methods more easily understandable.

Realistic systems

These systems are represented by larger graphs, and more clearly map to possible real-world systems. These graphs typically have many more vertices and edges, and can grow complex, although they should not be prohibitively large. These systems are more likely to contain unexpected interactions because of the scale and diversity of the vertices and edges they contain. These graphs aim to demonstrate the results

of techniques on realistic graphs. As a result, their primary concern is how closely they represent real systems – without giving much concern to how well the methods will scale.

Simulated systems

Scalability research is considered key in attack graph work. Large networks create large graphs, and large graphs lead to computationally expensive models. To assess this, simulated systems should lead to large attack graphs in order to evaluate how efficiently techniques can run. Accordingly, there is less of a focus on realism. Ideally, they still have similar graphical properties to realistic graphs as, for instance, a huge but disconnected graph is unlikely to give a full assessment of a method's scalability.

3.3.2 Challenges in gathering baseline systems

Gathering or creating these systems poses a number of challenges. Most importantly, any such system must have topological and vulnerability information in order to be useful. As a result, it is difficult to find such examples in a complete enough form to be used. Other attack graph work often avoids applying methods to real-world networks, likely because of the sensitivity of the data required and the associated difficulties in finding it.

This challenge primarily applies to realistic systems. Illustrative systems are easy to find or create, and related literature is full of relevant examples. Simulated systems, by their nature, do not need to relate directly to real networks and therefore can be generated to mirror real systems without requiring confidential or sensitive data.

3.3.3 Baseline systems

This section details the systems that I intend to use.

Simple Illustration

This system is a small network, intended to illustrate attack graphs. I created this example to highlight the impact of the same vulnerabilities being present in many places on a network.

The system consists of a collection of seven hosts, each of which can either be compromised or not. This gives the system seven conditions. The vulnerabilities on the network are mostly distinct and independent, but one path through the network consists of the repeated application of the same exploit on different hosts. One of the

hosts is considered of interest to the attacker, and is the culmination of any successful attack on this system.

To give a narrative for this system, let the system relate to the network of a small online platform. Three of the hosts relate to the web services of the platform, each pertaining to a different component. The remaining hosts handle the back end services that support the platform. Most importantly, the "goal" of the attackers is a host containing a database of customer information.

A host-based attack graph (e.g. [APRS05], see Section 2.2.3) of the system is shown in Figure 3.3. In this model, each vertex (aside from the initial state) directly relates to the compromise of a host in the system.

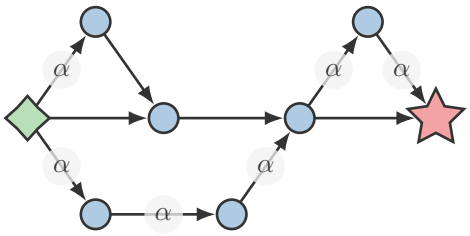


Figure 3.3: A host-based attack graph relating to this system. The edges labelled with α are the result of the same exploit being used against different hosts.

Online Retailer Illustration

The second illustration system is based on that of *Albanese et al.* [AJ18]. In the interest of standardisation and the encouragement of comparability, I have attempted to keep it as true to the original description as possible.

Figure 3.4 shows the host-based attack graph of the system, as derived from the probabilistic temporal attack graph in the original work.

The system is composed of seven hosts, arranged into three subnets. Hosts h_A and h_B form the first subnet, which has internet access. The hosts h_C and h_D form the second, which also has access to the internet. The final subnet is formed of h_E, h_F and h_G, which does not have external access and must be connected to from either of the other two subnets.

The attacker's goal is assumed to be host h_G, which hosts a database of customer, order and product information.

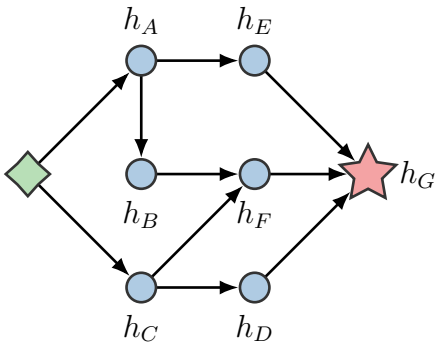


Figure 3.4: A host-based attack graph for the system found in Albanese *et al.* [AJ18].

Smart Home System

The third system is the motivating Smart Home example, as described in Section 3.2.1. This system represents a house containing a set of 26 smart devices of various types. Many of the vulnerabilities are possible for any attacker who has control over a device on the WiFi network to which the devices are connected. Some vulnerabilities require physical access from the attacker, which can be obtained maliciously by compromising the locks. Other vulnerabilities are possible for attackers that have no other privileges on the network: some by being close enough to the system to send crafted wireless signals, and others by connecting remotely over the internet or attacking cloud-based services.

Randomised Systems

Based on the smart home data, I created a means to generate arbitrarily large systems that follow the same patterns as those found in the smart home. These were used to test performance (Chapter 9), and so are not intended to be realistic – for example, the largest performance test used a smart home containing more than $30,000$ types of device.

3.4 Conclusion

In this chapter, I introduced my chosen formalisation of attack graph, based on template-matching methods. This technique will form the basis of all further discussions of attack graphs in this thesis. I opt for the use of a single goal state and initial state, primarily for simplicity, and because most methods can easily be adapted for multiple such states.

I also introduced a collection of baseline graphs relating to various scenarios, most importantly the smart home motivating example. The smart home example was constructed using samples of exploits from relevant databases. These will be used throughout the thesis to provide consistency in discussion and, particularly in the smart home, demonstrate the methods on a realistic example system.

Chapter 4

Relative Attack-Graph Security

In an ideal world, security analysts would have a way to objectively assign a value to the security of a system. Making security decisions would become straightforward: simply find the "security" score of each possible choice and compare. This score could even be used in conjunction with other measurements – such as cost or utility – and used in subsequent multi-objective optimisation methods.

Comparisons between the "security" scores of configurations of the same system would help decision-makers reach conclusions about which decisions to make. By knowing which options lead to the greatest "security", they can make an informed decision. Comparisons between different systems would assist with security evaluation, for purposes such as risk assessment.

Of course, no such "security" score exists: security is a complex, multi-faceted concept, and the meaning of security varies greatly between different applications and scenarios. Applying a single score to something so complex would lead to incorrect evaluations in different situations.

The benefits of objective security evaluations are clear, and while such a general and absolute measure of security is an unreasonable goal, we can begin to approach this problem through attack graph comparisons.

In this chapter, I provide the following:

- Section 4.1: An overview of my proposed method of comparing attack graphs.
- Section 4.2: A description of the rules by which attack graphs can be compared, first on attack sequences, then on attack suites, and finally on attack graphs themselves.
- Section 4.3: An application of attack graph comparisons to validate existing attack graph metrics.

- Section 4.4: A critical reflection on the work presented in this chapter.

4.1 Overview

In this chapter, I aim to provide restricted examples for which I believe objective security evaluations can be made. These examples have a number of caveats compared to the ideal case.

Firstly, and in keeping with the nature of this thesis, I will assume that systems can be reasonably modelled by attack graphs. Although this assumption should not be taken for granted, the accuracy and applicability of attack graphs is outside the scope of this chapter. Accordingly, during this chapter I refer to the security of the system as being accurately captured by its attack graph, even though there may be many practicalities that affect this.

Secondly, I will weaken the proposed goal: instead of providing an absolute notion of security, I will provide a relative notion. That is, I will not assign a "security" value to a system, but will claim that, for certain pairs of systems, one can be said to be objectively more secure than the other. This offers a closely-related concept, which provides many of the benefits described earlier.

Thirdly, this relative security comparison will not be applicable in all cases. Relaxing the requirement that every system can be compared means that the comparison only needs to apply in cases where it can be confidently asserted. In fact, comparisons will only be applicable in a small subset of system pairs.

Despite these considerations, I believe my proposed security comparison has practical benefits. While indisputable, objective security measurements are very difficult, subjective measurements are considerably easier, and there are many such examples throughout the literature (see Section 2.1.1). On attack graphs, these typically take the form of metrics, where an attack graph is assigned a score that attempts to capture some aspect of the security of the graph.

The difficulties of validating these subjective security metrics are clear: without any known baseline for security, it is impossible to know how accurate or reliable a particular metric is. Even if considerable data about real systems were available, the relative infrequency and considerable chance-based component of attacks would make statistical evaluation unreliable. By creating a set of examples in which graphs can be compared objectively, I offer a way for subjective metrics to be tested: any subjective metric should agree with the objective comparison in every case where the objective comparison is defined.

In fact, a number of metrics proposed and used in attack graph literature do not agree with the objective comparison I propose here. This enables me to find clear test cases which demonstrate intuitively incorrect security evaluations from the subjective metrics. I then propose an updated subjective metric which rectifies the problem while still attempting to capture the same aspect of security as the original metric.

To compare security objectively, subjective claims must be avoided. Claims which require data that is hard (or impossible) to gather must also be avoided. In particular, evaluating the difficulty of individual exploits is likely to involve one or other of these two issues, so in order to not make unsupported claims, I will not compare exploits. In practical situations, gathering the required data for such a comparison may lead to problems that cause the conclusion to never be clear, and certainly not objective as required.

For example, we might reasonably believe or assume that one system is objectively more secure than another when all of its exploits are more expensive for the attacker to perform than that of the other. This provides us with a way to compare some systems, as required; but in practice it is unreasonable to precisely determine (or necessarily even estimate) the cost of an exploit for an attacker. Consequently, while the comparison may be sound, it is not likely to ever be useful.

The only assumptions I make about individual exploits (beyond those used to construct the attack graph) are:

- The burden they each place on the attacker is *non-negative*, so that the attacker's gain from performing an exploit is solely captured in the attack graph structure (i.e. an exploit may provide more opportunities to the attacker).

- The burden they each place on any given attacker is *constant*. That is, each exploit may place a burden on the attacker, but any interactions between different exploits are captured in the attack graph. For instance, an attacker may have to complete one exploit in order to perform another, but this is encoded in the structure of the graph. This assumption has been made by other authors (e.g. [MGSPL17]).

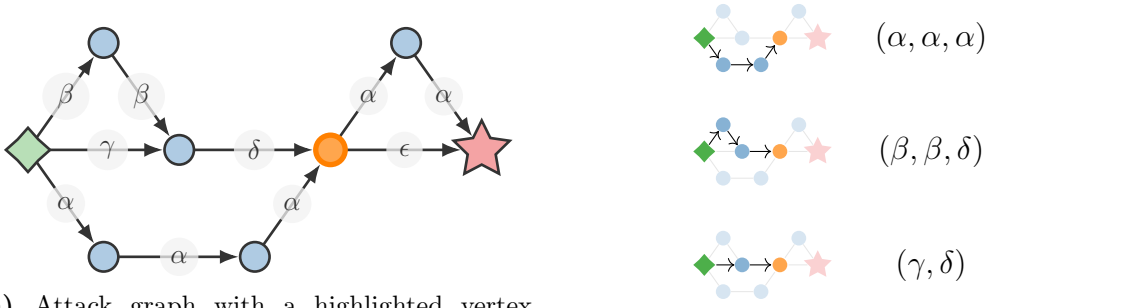
In my template-based formulation of attack graphs, each edge on the graph corresponds to a template, which corresponds to a real-world exploit. If two edges on the graph correspond to templates representing the same real-world exploit then we can say they are equal. If they correspond to different exploits, then we make no claims at all about how they compare.

4.2 Security Comparisons

The aim in this section is to demonstrate that there are pairs of attack graphs which can be compared objectively. That is to say, under the assumptions above, one of the attack graphs (or the system it represents) is more secure than the other.

I begin by attempting to find comparable pairs of attack *sequences*. An attack sequence is an ordered set of exploits. Each path in an attack graph corresponds, via the templates, to an attack sequence.

The comparison between sequences can then be extended to find comparable attack *suites*, which are sets of attack sequences. A set of paths in an attack graph (for instance, the set of paths from one vertex to another) corresponds to an attack suite. For a particular vertex, the attack suite corresponding to paths from the initial vertex to that vertex represents all the possible ways an attack could reach the vertex. These definitions are illustrated in Figure 4.1.



(a) Attack graph with a highlighted vertex. Edges are labelled with their template, from the template set $\mathcal{T} = \{\alpha, \beta, \delta, \gamma, \epsilon\}$, each of which corresponds to a unique exploit.

(b) Paths to the highlighted vertex from the initial state and their corresponding attack sequences. The set of these attack sequences is the attack suite of the vertex.

Figure 4.1: Illustration of the attack sequences, and the attack suite, for a given vertex (highlighted)

I will let S be the set of all finite attack sequences, so that each element of S is a finite, ordered set of exploits. Next, I define the desired security (pre)order, using $\overset{s}{=}$, $\overset{s}{\leq}$ and $\overset{s}{\geq}$, respectively denoting *equally as secure as*, *at most as secure as* and *at least as secure as*. This order will first be defined on attack sequences, then attack suites, and finally on attack graphs.

I do not claim that this preorder is total, as there will be pairs of elements for which we can make no statement. By definition it is clear that such a relation should be transitive (so that $a \overset{s}{\leq} b$ and $b \overset{s}{\leq} c$ implies $a \overset{s}{\leq} c$). It is a preorder (as opposed

to a partial order) because it is not anti-symmetric; $a \overset{s}{\leq} b$ and $b \overset{s}{\leq} a$ does not imply that $a = b$, only that $a \overset{s}{=} b$.

4.2.1 Attack Sequence Comparisons

The first step is to consider attack sequences. For two attack sequences, a and b, I will write ab as the concatenation of a and b, so that if $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_m)$,

$$ab = (a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m)$$

Or simply:

$$ab = a_1 a_2 \ldots a_n b_1 b_2 \ldots b_m$$

We can now examine the conditions under which we can make security comparisons between sequences.

A natural first step is to state that if two attack sequences are the same, they are equally secure: they contain precisely the same set of exploits and correspond to the same set of actions for the attacker.

Permutation

Two attack sequences may contain the same exploits, but in a different order. In this case, the two sequences are equally secure if the order of the exploits has no impact on the burden they place on the attacker.

This follows from the assumption that the burden placed on attackers by an exploit is independent of other exploits already completed. As a result, the burden placed on the attacker by the exploit is independent of its position in the sequence, and so the order of the exploits has no impact on security.

This does not mean that attackers are equally able to perform the attack sequence in any order: the fact that two sequences are equally secure does not mean they are both *possible*. They are only possible if they both appear in the attack graph.

Formally, we can state this rule as follows:

> **Definition:** Permutation Rule
>
> If $a \in S$, then for any permutation b of a, we have:

$$a \stackrel{s}{=} b$$

As a demonstration, consider a system consisting of three hosts. Compromising the third host (host C) is only possible when both of the first two (hosts A and B) have also been compromised. This system has three conditions, h_A, h_B, and h_C, one for the compromise of each host. There are also three templates, each assumed to correspond to a unique exploit:

$$\begin{aligned} t_1 &: \mathcal{S} \longrightarrow \mathcal{S} & t_1 &: s \mapsto s \cup \{h_A\} \\ t_2 &: \mathcal{S} \longrightarrow \mathcal{S} & t_2 &: s \mapsto s \cup \{h_B\} \\ t_3 &: \{h_A, h_B\} \longrightarrow \mathcal{S} & t_3 &: s \mapsto s \cup \{h_C\} \end{aligned}$$

There are therefore two paths in the resulting graph that lead to a state containing h_C, one corresponding to $t_1 t_2 t_3$ and the other $t_2 t_1 t_3$. These two sequences are permutations of each other, and therefore are equally as secure as each other. In the example, the attack consists of satisfying two independent prerequisites (h_A and h_B) before completing the final attack. The order in which the attacker performs the two prerequisites is unimportant because they do not effect each other. As a result, the two paths present the same burden to the attacker.

Subsequences

Extending a sequence by adding another exploit onto the end of it can only increase the burden performing the sequence places on the attacker, due to our assumption of non-negative burden. In order to complete the extended sequence, the attacker must carry out the shorter sequence, and then the additional exploit. Hence the longer path is at least as secure as the shorter. We cannot claim that it is *strictly* more secure, because we do not know that the burden placed on the attacker by the new exploit is non-zero, only that it is non-negative.

Formally:

Definition: Additional Step Rule
For $a \in S$, $b \in \mathcal{T}$:

$$a \overset{s}{\leq} ab$$

We can also see this applies to sequences with any number of additional steps, and irrespective of where they are in the path, provided they contain the other path as a subsequence:

Definition: Subsequence Rule
For a, $b \in S$, if a contains a permutation of b as a subsequence then:

$$b \overset{s}{\leq} a$$

Proposition 1. If an order on S obeys the Additional Step Rule and the Permutation Rule, then it obeys the Subsequence Rule.

Proof. Let a, $b \in S$, such that a contains a permutation of b as a subsequence. Let c be that permutation, so that $a = a_1 a_2 a_3 \ldots a_j$, $c = c_1 c_2 c_3 \ldots c_n$ and $(c_i)_{i=1}^{n}$ is a subsequence of $(a_i)_{i=1}^{j}$.

Then by the permutation rule, we can reorder the elements and take all the elements of c and move them to the start of a while preserving security. If $i_1, i_2, \ldots, i_k$ are the indices of (a_i) that are not part of the subsequence:

$$a \overset{s}{=} c_1 c_2 \ldots c_n a_{i_1} a_{i_2} \ldots a_{i_k}$$

Let

$$a^m := c_1 c_2 \ldots c_n a_{i_1} a_{i_2} \ldots a_{i_m}$$

So that $a^k \overset{s}{=} a$ and $a^0 = c$. Hence by the additional step rule

$$a^{m-1} = c_1 c_2 \ldots c_n a_{i_1} a_{i_2} \ldots a_{i_{m-1}} \overset{s}{\leq} c_1 c_2 \ldots c_n a_{i_1} a_{i_2} \ldots a_{i_m} = a^m$$

So

$$b \overset{s}{=} c = a^0 \overset{s}{\leq} a^1 \overset{s}{\leq} \ldots \overset{s}{\leq} a^k \overset{s}{=} a$$

and therefore

$$b \overset{s}{\leq} a$$

□

4.2.2 Attack Suites

These rules describe how we can compare two individual attack sequences, and hence two attack paths. But in an attack graph, we will have a set of possible paths that lead to the same vertex. Ideally, we would like to be able to say that a vertex is more secure than another vertex because it has more secure attack paths.

Each vertex in a graph has a set of paths that lead to it from the initial vertex. Each path in this set has a corresponding attack sequence, consisting of the relevant exploits. These sets of attack sequences are known as *attack suites* [MO06]. I will use S' to denote the set of attack suites (so, an element of S' is an attack suite, which is a set of attack sequences. An attack sequence is a sequence of exploits). Each vertex in the graph therefore has associated attack suite which contains the attack sequences that lead to it from the initial state.

Security Reducing Maps

In a similar fashion to attack sequences, if an attack suite is extended by adding another attack sequence, then it is at most as secure as the original suite. We cannot claim that it is strictly less secure after the addition, because it may be the case that the new sequence is impossible.

The extended suite is at most as secure because the additional sequence presents an attacker with another means to complete the attack suite. Specifically, if any attacker performs one of the sequences in the extended attack suite, then either they perform one of the original sequences, in which case the security is unchanged, or they perform the new sequence, in which case the security is potentially reduced.

This leads to a more general rule: consider two attack suites, A and B. If, for every attack sequence in A, there is an attack sequence in B which is at most as secure, then we can say that A is at least as secure as B.

Formally, this describes a map f, mapping A to B and satisfying $f(a) \overset{s}{\leq} a$. If such a map exists, every attack sequence in A has an equivalent in B, given by $f(a)$, which is at most as secure. I will refer to these maps as *security-reducing*.

> **Definition:** Security-reducing map
>
> For A, $B \in S'$, a map $f : A \longrightarrow B$ is *security-reducing* if $\forall a \in A$, $f(a) \overset{s}{\leq} a$

Definition: Suite Comparison Rule

For A, $B \in S'$, if there exists a security-reducing map from A to B, then

$$B \overset{s}{\leq} A$$

With this we can compare attack sequences. For example, we can see that:

$$A := \{a\} \overset{s}{\leq} \{ab, ac, ad, ae, af\} =: B$$

Proof. Let $g(xy) = x$ for x, y exploits. Then g is a security-reducing map, because $x \overset{s}{\leq} xy$, and $\forall b \in B$, $g(b) = \{a\} \in A$. □

This is because B represents a situation in which the attacker must perform the same exploit as in A and then another. Even though there are more attack sequences in B, it is more secure because the variety between these sequences is superficial – they all rely on compromise of a anyway.

Similarly, we can show that:

$$A := \{abc\} \overset{s}{\leq} \{bac, cab, bca\} =: B$$

Proof. Let $g(xyz) = abc$. Every path in B is a permutation of abc, and so is equally secure as abc. Hence $\forall x \in B$, $g(x) = x$. And clearly $g(B) = \{abc\} = A$. □

But there are many examples of sequences which we cannot compare. Some of these might be suites which are very different, and so this is not surprising.

But some are examples where it may seem natural that one is more secure than the other, even though we cannot make the comparison:

$$A = \{a\} \text{ and } B = \{b_1 b_2 b_3 \dots b_n\}$$

We might expect $A \overset{s}{\leq} B$. A seems to represent very poor security – it only takes one exploit to reach the goal – and B represents a very secure network. However, we cannot make this comparison because it would require us to compare exploits. It may be that a is nearly-impossible to perform and b_i are all trivial. In that case, B would be less secure than A. In fact, it could be the case that the two actually represent the same system, and that the exploits in (b_i) are smaller components of a larger exploit a. Without assuming further knowledge of the exploits, we cannot draw any conclusions between these two suites.

Similarly:

$$A = \{a_1, a_2, \ldots, a_n\} \text{ and } B = \{b\}$$

Again we might expect $A \overset{s}{\leq} B$. A has many one step paths to the goal whereas B has only one. Again, we cannot make this claim because we have know means to compare individual exploits. Even with many possible choices, it may be that b is trivial and each a_i impossible.

Security Ordering

We now have a collection of rules which, under our assumptions, serve as objective comparisons between specific pairs of attack suites. This set is not complete in two ways: firstly, many comparisons between attack sequences do not fall into any of the rules and so we cannot make any statements about these pairs; secondly, this list of rules is not comprehensive. Despite this, we can still define the concept of an *invalid* security order on the set S'.

> **Definition:** Invalid Security Order
>
> An order on the set S' is an invalid security order (with respect to $\overset{s}{\leq}$) if it breaks any of the rules above, that is the *Suite Comparison Rule* on attack suites, and the *Permutation Rule* and the *Edge Addition Rule* on attack sequences.

A security ordering being invalid with respect to $\overset{s}{\leq}$ does not mean it is not meaningful or useful, only that it does not conform to the objective security measure in all cases. This may mean it makes different assumptions to those made above.

Invalidation of security orders can be applied to metrics. A metric on the set of graphs is first used to create an order on the set: for a metric μ, we can define an order by saying that $A < B$ if $\mu(A) < \mu(B)$. We can then say a metric is invalid if its corresponding order is invalid.

4.2.3 Attack Graph Comparisons

Equipped with the means to compare attack suites, we can compare the security of individual vertices. Each vertex has a set of paths that lead to it from the initial vertex, and each of these corresponds to an attack sequence. The collection of sequences (i.e. attack suite) for a vertex encapsulates all possible attacks in the graph

that lead to that vertex. We can therefore relate the vertex to its attack suite, and hence compare vertices by comparing their attack suites.

By extension, we can therefore compare any two attack graphs for which there is a single goal – we just compare the attack suites of their goals.

For such a comparison to be possible, the attack graphs do not need to share the same state set or template set. However, they should contain some templates which correspond to the same exploits, as these will enable comparisons between the graphs.

As an example, we may wish to model the effect of a new vulnerability being discovered in the system. To do this, we could consider two attack graphs on the same set of states, but where one graph was constructed with an additional template – which corresponds to the new vulnerability. We can then compare these two graphs to see if the security of the system is impacted.

I extend the notation to attack graphs and say that $A \overset{s}{\leq} B$ (equivalently for $\overset{s}{=}$ and $\overset{s}{\geq}$) if $g_A \overset{s}{\leq} g_B$ (for g_A and g_B the attack suites of the goal vertices of A and B respectively). In this section, I assume the ordering $\overset{s}{\leq}$ obeys the rules above.

While I do not discuss it in this section, it should be possible to extend this to attack graphs with multiple goals, where two graphs are compared by each of their goals in turn (provided that the goals on each graph correspond to each other). Two graphs could be said to be comparable if they are comparable in all of their goals. In other words, if the goal vertices are at least as secure in every case, then the graph is at least as secure.

I proceed now by describing two graph modifications. These are processes by which a graph can be changed, so that the resulting graph can be compared to the original. By defining the graphs in this way, it is clear that the majority of the graph is consistent between the two versions.

Template Addition

(a) Without the extra template

(b) With the extra template

Figure 4.2: An attack graph that has been "modified" by adding a template. The template adds two edges, highlighted and dashed.

The first modification is template addition, where an attack graph is modified by adding an additional template to its construction. In other words, we consider two attack graphs, A and B. The template set of A is given by $\mathcal{T}$ and the template set of B is given by $\mathcal{T} \cup \{t\}$.

In practice, this could represent the addition of another action the attacker could make, perhaps because a new service was added to the system, because a control was removed, or because a new class of vulnerability was discovered. Figure 4.2 illustrates this.

The added template could have no impact because it was impossible for any attacker, or because it gave the attacker nothing of value when performed (i.e. the added edges did not form any paths to the goal). In this case, the security of A and B could be equal. On the other hand, the attacker could use this edge to their advantage and B could be less secure than A. However, the addition of a template to the system cannot *increase* security – the attacker has another weakness in the system at their disposal.

By construction, if $A = (V, E)$ then $B = (V, E \cup E')$ for some set of additional edges E', each of which is of the form $(v_1, t(v_1), t)$.

We can then prove that the addition of these edges does not increase security via the security ordering.

Theorem 1. Edge Addition
For attack graphs $A = (V, E)$ and $B = (V, E \cup E')$, we have:

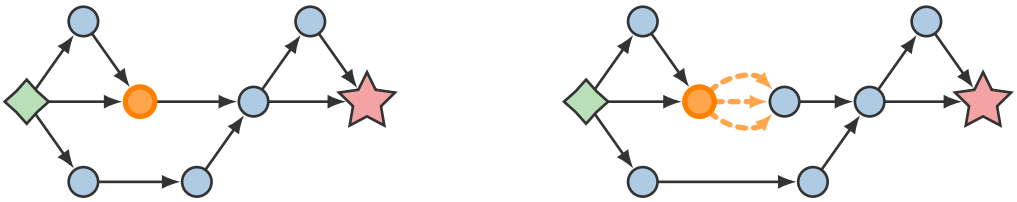
$$B \overset{s}{\leq} A$$

Proof. Let S_A and S_B be the attack suites of the goals of A and B. For each $a \in S_A$, there exists a corresponding path in A to the goal of A. Define p_A to be a bijection from S_A to the set of paths to goals of A, so that the templates of the path $p_A(a)$ are the attack sequence a. Equally, define p_B.

Because the edge set of B is a superset of the edge set of A we know that, for any $a \in S_A$, $p(a)$ is also a path in B. Hence we can consider the image of S_A under p as a set of paths in B. Then, because p_B is bijective we can find p_B^{-1} and hence $f := p_B^{-1} \circ p_A$ is a map from S_A to S_B.

By construction of f, we have that $f(a) = a$. Hence $f(a) \overset{s}{\leq} a$.

So f is a security-reducing map from S_A to S_B, and $B \overset{s}{\leq} A$. □



(a) Without the vertex split **(b)** With the vertex split

Figure 4.3: An attack graph being modified with a vertex split. The highlighted vertex v is split and the new vertex is connected with three new edges, shown highlighted and dashed.

Vertex splitting

The second modification is vertex splitting. Vertex splitting, illustrated in Figure 4.3, is a process by which a vertex is replaced with two connected vertices. For a chosen vertex v in the graph $A = (V, E)$, we form B by taking $B = (V \cup \{v'\}, E')$. In the new edge set, we ensure that every in-edge of v is still in E', but every out-edge from v now leads from v' instead. We then connect the two vertices v and v' with any number of edges that all lead from v to v'. In other words, E' satisfies the following conditions:

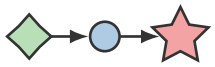
(1) For every edge $e \in E$ such that e is not incident to v, $e \in E'$

(2) For every edge $e \in E$ with $e = (u, v, t)$, there exists an edge (u, v, t) in E'

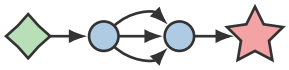
(3) For every edge $e \in E$ with $e = (v, u, t)$, there exists an edge (v', u, t') in E', and t and t' correspond to the same exploit.

(4) All other edges in E' are of the form (v, v', t) for some t, and there is at least one such edge.

To explain this process in terms of a system being modelled, consider a case (illustrated in Figure 4.4) where an attack graph contains three vertices and two edges, in a line starting from the start and leading to the goal. The first of these edges represents the compromise of a device, and the second represents the use of this compromised device to reach the goal. We then compare this with a system in which a privilege escalation is required on the device in order to perform the second exploit. This graph has the middle vertex split into two – one new vertex representing the initial device compromise and one representing the escalated privileges – and a set of edges added between these representing possible methods of escalation.



(a) Without the vertex split



(b) With the vertex split

Figure 4.4: A simple system having its security improved, represented by a vertex split.

From this example, we can see that vertex splitting should improve the security of the system: it represents an additional step being added for the attacker to reach the goal.

> **Theorem 2** (Vertex Splitting)**.** Let $A = (V, E)$ be an attack graph and form $B = (V \cup \{v'\}, E')$ via the vertex splitting process above, splitting the vertex v. Then:
>
> $$A \overset{s}{\leq} B$$

Proof. Let S_A and S_B be the attack suites to the goals of A and B. Define a function $g : S \longrightarrow S$, so that for a sequence $s = (s_i)_{i=1}^n$:

$$g(s) = (s_i : s \in E)$$

(where $\mathcal{E}_G$ is the edge-labelling map of B).

Then for $b \in S_B$, $g(b)$ is an attack sequence of exploits that only includes exploits that correspond to edges in A. So $g(b)$ can be interpreted as a (potentially impossible) sequence of exploits in A. Such a sequence is in S_A if it corresponds to a path in A – i.e. each consecutive pair of exploits in $g(b)$ are consecutive in A (so that the first terminates in the start vertex of the second). For each pair in $g(b)$, we have two cases:

1. The pair appears consecutively in b, and hence is consecutive in B and therefore A

2. The pair was originally separated in b by one or more other exploits corresponding to edges from $E' \setminus E$. Let such a pair correspond to the edges $e_1 = (v_1, v_2)$ and $e_2 = (v_3, v_4)$ in B. Then $v_2 = v$ and $v_3 = v'$. So this pair was consecutive before v was split into v and v', and hence is consecutive in A.

So therefore each pair is consecutive in A and so $g(b) \in S_A$. Therefore g is a map from S_B to S_A. Further, g is security-reducing, because $g(b)$ is a subsequence of b (Subsequence Rule).

So g is a security-reducing map from B to A and so $A \overset{s}{\leq} B$

□

4.3 Metric Validation

The rules stated in the previous section provide a method of comparing the security of two graphs in a precise way. For certain pairs of graphs, and under the assumptions made, we can say that one of the pair is objectively at least as secure as the other. To some degree, this achieves the goal of this chapter: the original intention was to provide a means for network defenders to objectively "score" the security of their system.

The objective security comparisons are a useful step towards this goal, but clearly fall short of achieving it entirely. Primarily, we can only perform relative comparisons, instead of the absolute evaluation that a score would provide. This means that the comparisons cannot easily be compared to other aspects of the design, such as usability or cost. We might know that a particular configuration is more secure than any alternatives, but without knowing how significant the difference is, it is impossible to justify any potential increase in cost.

Secondly, we are still restricted to making comparisons on a very specific set of pairs of graphs. This means that, in practice, the comparisons cannot (usually) be directly applied to two graphs that represent differently configured systems. We can demonstrate general principles of secure design, but we cannot examine specific cases.

The standard method of evaluating the security of an attack graph is through the application of a metric. In general, a metric is a map from the set of attack graphs (however they may be defined) to a real number. Metrics usually attempt to quantify some aspect of the security of the system, and it is typically assumed that the metric therefore reflects security to some degree.

Metrics, by associating each graph with a value, provide an absolute measure of security. While this is not always measured in real-world terms, it still offers a method for analysts to compare between security evaluations and other objectives. This is further enabled by metrics that measure in values that have real meaning. For example, some metrics directly associate the graph with a risk-based cost [AJ18] or with the expected time to compromise [Abr16].

Metrics provide valuable information about the security of the graph, but are often poorly justified and rarely validated [Ver09]. Many existing works simply state new metrics with a brief explanation of what it is intended to correspond to. This means that there is little guarantee of reliability for users applying the metric. Compounding this, there are a huge number of possible metrics [RLFR17] and choosing between them is difficult, particularly when they offer conflicting evaluations.

To help resolve these problems, I propose the use of the objective security comparison to validate metrics. Assume we have two graphs, A and B, such that $A \overset{s}{\leq} B$. If a metric μ claims to correspond to security then it should be the case that $\mu(A) \leq \mu(B)$. If we can find a pair of such graphs A and B for which this is not true then we can establish that, at least in some cases, μ does not represent security well. Conversely, if we cannot find such a pair of graphs then we can make a weak assertion in favour of the metric – we cannot make a definitive claim because we do not have a complete set of rules.

In this section, I give examples for which common metrics disagree with the objective ordering. I then use these as a basis to discuss why the disagreement arises and then demonstrate the disagreement on more realistic systems, therefore indicating how basing decisions on the metric could result in poorly-informed choices.

4.3.1 Number of Paths

Common sense tells us that the more ways an attacker has of accomplishing something, the more likely they are to succeed. If a system has many vulnerabilities that give the attacker access to critical targets, it is more likely that an attacker will be able to reach those targets.

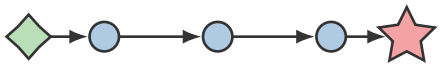
A system with many routes for the attacker also poses a problem for decision makers deploying hardening measures: with limited resources, only a subset of the possible routes can be mitigated against.

In an attack graph, this is represented by the number of paths to the goal. Each path equates to a combination of vulnerabilities that an attacker can use to reach their target. Accordingly, the *number of paths* metric has been proposed to capture this. This metric is defined on the set of attack graphs, and maps each graph to the total number of distinct paths from its initial state to the goal. Note that these are paths (as opposed to walks) so each path contains each vertex at most once.

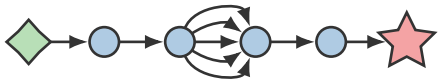
In keeping with our expectations for security, a lower metric value is considered to be more secure.

However, we can see this metric is easily skewed by modifications on the graph that have been demonstrated to have no impact on security. Consider a graph (shown in Figure 4.5) consisting of a chain of exploits, one after the other. This has only one path to the goal, and is consequently, and reasonably, considered secure by the metric. On the other hand, if one of the vertices is split into 2 vertices connected by 5 edges, we know that the new graph is at least as secure as the old one – it is the

result of a vertex split. However, the first graph has one path to the goal and the second has 5.



(a) A simple attack graph



(b) The simple attack graph with a split vertex

Figure 4.5: Two simple attack graphs, with **(b)** the result of a vertex split on **(a)**.

This demonstrates a toy example where there is a disagreement between the number of paths metric and the security ordering. As a result, we can claim that under the assumptions of our security ordering, the number of paths metric is an invalid security ordering. The obvious question for a practitioner seeking to apply the metric to ask is: does it matter? The simple example was constructed specifically to demonstrate the problem and does not accurately represent the complexity of a real network. Yet we can use this as a basis to discover more fundamental problems with the relationship between the number of paths and security.

In the example, the number of paths through the graph has increased but, in a more intuitive fashion, the number of ways for the attacker to reach the goal is still unchanged: the attacker still broadly follows the same, single route to the goal, albeit with some minor deviations. The *number of paths* metric weights each path equally, irrespective of how different they are.

We can see the impact of this problem on a graph with a bottleneck (i.e. a single edge which connects the initial vertex to the goal). An example is given in Figure 4.6. This graph has a very high number of paths because of all the possible ways to traverse the two, highly-connected sets of vertices. But every attack must utilise the same "bottleneck" exploit. As a result, the variety between all the different paths is somewhat artificial.

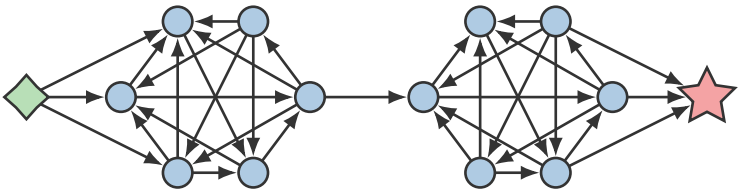


Figure 4.6: A graph with a clear bottleneck. This graph has a large number of paths between the initial state and the goal, but every path includes one specific exploit.

4.3.2 Min-cut

The number of paths attempts to capture the diversity of the paths that the attacker can use. Another natural way to do this is to find the minimal set of edges that must be removed to disconnect the initial state and the goal [SHJ+02]. We can create a *min-cut* metric that associates each graph with the size of this set. Each required cut represents a distinct way the attacker can reach the goal, and hence a path that must be cut. Applying this to the bottleneck graph in Figure 4.6 captures the idea behind the bottleneck clearly: the min-cut is one, found by removing the bottleneck edge.

However, we can also find cases where the min-cut results in comparisons that do not conform to the security ordering. In particular, we can consider two graphs that contain the same exploits but in different orders. By reordering the exploits we can change the structure of the graph. In the first suite, we have the sequences $\{abc, abd\}$ and in the second $\{bac, dba\}$. In the first graph, the edges corresponding to a and b are shared by the two sequences, and hence either of these can be removed to disconnect the graph. As a result, the min-cut of this graph is 1. On the other hand, the second graph has no such shared edges, and so has a min-cut of 2.

But we can construct security reducing maps in both directions between these two sets, and so they are considered equally secure. As a result, we have that the min-cut metric disagrees with the suite comparison rule, and so is an invalid security metric.

While this is, again, a toy example, it can be used to expose the problem with the min-cut metric. This is that the metric looks for cuts in terms of *edges* and not of *exploits*. As a result, it is susceptible to being modified by the specifics of how the graph is constructed. Instead we can consider the *min-cut of exploits* metric – i.e., how many exploits must have all their corresponding edges removed in order to disconnect the initial state and the goal. In the example above, this value is 1 in both cases, which agrees with the security ordering.

4.4 Critical Reflection

In this chapter, I approached the measurement of attack graph security from a theoretical basis. I specified a set of assumptions to work under and derived pairs of attack graphs that could be compared objectively under those assumptions.

From this, I was able to create a set of criteria against which I could evaluate other, subjective security metrics. I found that some commonly used metrics failed to validate against these criteria, and demonstrated cases in which the metric make a claim about security which violated with criteria and went against an intuitive notion of security.

4.4.1 Benefits

My approach enables intuitive notions of security to be formalised into specific rules for attack graphs. This enables us to make precise claims about the validity of security evaluations that otherwise are not possible. Attack graph methods suffer from a lack of validation, both theoretically and in practice. By offering this groundwork, the security comparisons provide a validation method that enables researchers to assess how well their evaluations conform to formalised notions of security.

My approach is suitable as a basis for future work. While I propose a fundamental set of rules under one set of assumptions, the process I use can be easily adapted and reused with different assumptions, or with further rules. This would result in separate sets of rules, which could then be compared with those found, or used to validate metrics independently. This would give practitioners a range of possible assumptions that they can make, which can be tailored to their particular use case.

Making assumptions explicitly enables them to be critiqued. This is an improvement over other metrics, which may be based largely on implicit intuitive notions of security and, hence, on assumptions that are hard to evaluate, critique, or test.

Defining only a partial order means I am not forced to make a claim about security without sufficient justification. In contrast, metrics that assign a value to every graph must, necessarily, make a claim about every graph. This makes their evaluations difficult to justify.

My approach proves that security conclusions can be made with little knowledge of exploits. Specifically, this approach avoids making any comparison between distinct exploits. Comparing the relative difficult of exploits is difficult to justify in practice, and so by avoiding it entirely prevents such comparisons from being necessary.

Building the restricted objective comparison benefits existing metrics. The application of my security ordering to metrics demonstrates cases in which they violate my assumptions and hence provide inaccurate security assessments. This does not mean the metrics are without merit, but highlights situations in which they should be used with caution. In situations where two similar metrics could be used, the invalidation of one is an indicator that the other metric may be a better choice for more reliable evaluations.

4.4.2 Drawbacks

These techniques are not comprehensive, and so it is difficult to make claims in an affirmative sense: we can claim that a metric is *invalid*, but not that it is certainly *valid*. A further security rule could reasonably be proposed that would invalidate a metric that is valid with respect to the rules I have proposed. This is not necessarily a problem with the method: security is too complex an issue to be comprehensively quantified outside of very specific circumstances.

In practice, many attack graphs will contain different templates. This work does not assume any means by which these templates can be compared. While this is broadly a strength, it means that we cannot make comparisons in many cases. This work could be complemented with a similar approach to template comparison.

While I believe my technique has value outside of my specific choices of assumption, the evaluation of metrics I perform here is dependent on the validity of these assumptions. As with all parts of security, there is scope for disagreement about whether these assumptions hold.

This method is not a direct method for practitioners to apply to understand their systems better; instead, it serves as a theoretical method for metrics to be evaluated against. To be useful to practitioners, it is still necessary to develop metrics or use those from other related work.

4.4.3 Conclusion

I believe my work in this chapter offers a different approach to attack graph security evaluation. This approach works well with existing approaches and I demonstrate how it can be used in conjunction with existing metrics, to strengthen confidence in them.

Future work in this area could assess and modify the assumptions made. Such work would offer alternative comparisons under different conditions, which may apply

better in different cases. The diversity of computer systems means that assumptions that may hold in one case are unsuitable for another. Applying my approach in specific use cases may enable much stronger assumptions to be made, which would result in stricter rules and potentially more comprehensive and complete comparisons.

This work is dependent on the attack graph being a reliable and practical model of security. Even without comparing exploits directly, we still depend on an accurate graph construction process, so that the structure of the graph represents reality as best as possible.

Restricting evaluations to situations where they can be concretely justified from assumptions has value, but makes it difficult to provide practical conclusions. In practice, intuitive security assessments provide practitioners with actionable information, despite a lack of theoretical certainty. In fact, for attack graphs to be useful the most pressing concern is perhaps that they should be practical. In the following chapters I focus on attack graph generation with a consideration for the practicalities of the process.

Chapter 5

Single-Precondition Assumption

The scalability of attack graphs has been the subject of a considerable amount of work. The assumption of monotonicity [AWK02] is perhaps the most frequently cited improvement to the attack graph processes, and is credited with a considerable reduction in graph size without meaningful loss in modelling power. Monotonicity is the assumption that a system under attack moves monotonically towards insecurity.

While this assumption is often made by authors, situations can be demonstrated in which it, alone, is insufficient to justify the performance improvements newer techniques have demonstrated. Given this, it remains to ask if there is an additional assumption being made implicitly that does explain the improvements.

In this chapter, I propose and discuss the single-precondition assumption, which states that each exploit is dependent on only a single precondition. By making this assumption explicit, I can examine the drawbacks and benefits of using it. This enables us to make the assumption conscious of its consequences and aware of the conditions it imposes.

This chapter consists of the following sections:

- Section 5.1: An overview of the scalability problem, existing solutions, and use of preconditions in current work.
- Section 5.2: A detailed explanation of the state-space problem in attack graphs, together will some examples of complexity in other attack graph work.
- Section 5.3: A formalisation of the explicit single-precondition assumption, together with a general form, the partition precondition assumption. These are contrasted with a formalisation of the assumption of monotonicity as defined on the template-based definition of attack graph.

- Section 5.4: A justification of the single-precondition assumption, through example vulnerabilities. These vulnerabilities are taken from the smart home vulnerability set, as well as drawn directly from generic vulnerability databases.
- Section 5.6: A critical reflection on the work in this chapter.

5.1 Overview

Attack graphs suffer from problems of scalability. The complexity of the model often grows exponentially with the size of the system, due to the exponential number of possible combinations of exploits. A widely-adopted improvement to the graph construction process is the assumption of monotonicity [AWK02] – the assumption that attackers do not lose privileges once they have been gained. This assumption is widely-regarded as significantly reducing complexity.

Techniques that use monotonicity have demonstrated much faster construction times – some even approaching linearity in favourable cases [ILP06]. Some techniques avoid creating many states by constructing dependency-based attack graphs, but as discussed these methods result in attack graphs more similar to attack trees.

Other techniques have similar performance improvements without building dependency graphs. In particular, host-centric attack graphs reduce the required number of vertices to one per host. Exploit preconditions were originally designed to include an arbitrary set of required properties, or preconditions. Host-centric (and similar) methods create attack graphs that implicitly no longer allow for arbitrary sets. Instead, it is assumed that the preconditions of an exploit relate to only one host, or in other cases, the stronger assumption that the preconditions of an exploit are a single required property.

One of the first examples of this assumption was made by *Ammann et al.* [APRS05], who constructed their attack graphs via a host-based method, creating an "access graph" containing hosts and their connectivity, then deriving an attack graph from this. They allow for multiple preconditions, but only across at most three hosts – the attacker, the victim and potentially a middle-man. By making this assumption, their work uses drastically fewer states and becomes considerably more computationally feasible.

Later work developed this idea; Hewett and Kijsanayothin [HK08] made the host-centric nature of their graphs explicit. In their model, each attack graph vertex corresponds only to the state of a single host. Consequently, their work implicitly disallows exploits with preconditions involving multiple hosts.

Ingols et al. [ILP06] proposed a multiple-prerequisite graph (see Section 2.2). These graphs are not entirely host-based or dependency-based. Instead, they contain vertices that directly represent prerequisites. This technique simplifies the modelling process and results in near-linear performance. However, in their simulations they allow only for two types of prerequisite: reachability and knowledge of credentials. Further, they assume there will be very few credentials (to achieve near-linear performance, their test contains no credentials). This effectively means their model assumes single preconditions. Each exploit has only one precondition: that the attacker has access to the target host.

Implicitly making this assumption hides it, and subsequently, to the best of my knowledge, there has not been any work explicitly considering the consequences or validity of the assumption. A priori, it is not clear that every exploit can reasonably be modelled under this assumption. In this section I aim to provide grounding and validation for this assumption so that I can use it throughout the rest of my work.

5.2 State-space Explosion Problem

Graphical models of security seek to provide a method of understanding the "whole picture" of the system's security. By constructing a model of the whole system, graphical models explore complex interactions that cannot be captured through the individual analysis of single parts. This necessarily involves the collection of detailed information about every part of the system; in a situation where attackers will use any route to reach their goal, even seemingly unimportant components of a network can be a gateway for attack.

Modelling complex systems causes scaling issues. Modern networks vary in size greatly, and systems being modelled could reasonably contain thousands of hosts. To successfully model such systems, scalability must be manageable. Techniques only suitable for small numbers of hosts cannot be used to capture the full scale of modern systems and so lose the ability to examine the paths attackers can take through the network.

The earliest attack graph methods (e.g. [PS98]) created vertices for each possible combination of properties of the network. Early techniques use template-matching methods with arbitrary preconditions, resulting in states that can be arbitrary combinations of property conditions. The number of states in such a model is as many as 2^n – one for each possible combination from a set of n properties – and so grows much too quickly to be usable.

Here, I aim to demonstrate that monotonicity, as first explicitly stated by Ammann et al. [AWK02], is not responsible for the reduction in complexity. Monotonicity is defined as *"the precondition of an exploit, once satisfied, never becomes unsatisfied"* [AWK02], and that *"the negation operator is not used to express the precondition of any exploit"* [AWK02]. The second of these assertions is less important; it serves simply to prevent preconditions being written "backwards", so that conditions becoming satisfied cannot cause exploits to no longer be possible.

Demonstrative Counterexample

To demonstrate this, I provide the following counterexample, of a system modelled under monotonicity that still requires a number of states exponential in the number of system properties.

Consider a network consisting of $n + 1$ hosts: one of the hosts is considered the *target*, and the remaining n are considered *gateways*. Each gateway is exploitable using a distinct vulnerability, which have no preconditions (i.e. it can be compromised by an attacker without having to do anything else first).

The target host is only exploitable via stolen credentials, available on the gateways. However, to increase the security of the system, the target will only accept remote access when presented with $n - 1$ credentials, from $n - 1$ distinct gateways.

Our system properties are therefore n properties each corresponding to the compromise of one of the distinct gateways, and one property corresponding to the compromise of the target.

The exploits (or their corresponding templates) are therefore as follows: there are n exploits, each corresponding to a different gateway; they have no preconditions and the postcondition that the corresponding gateway is compromised. There is one further exploit, which corresponds to the successful use of the stolen credentials: its precondition is the compromise of any set of $n - 1$ gateways, and its postcondition is successful compromise of the target.

We can see this model is monotonic: properties never become unsatisfied, and we do not use negation in preconditions.

To model this system as an attack graph, we must have a state for each possible subset of gateways, in order to accurately model the possible ways in which an attacker can obtain the credentials. But there are 2^n such combinations. Hence, the model will grow exponentially in complexity with respect to the number of hosts.

It can therefore be seen that monotonicity, as stated, is insufficient to stop attack graphs from growing with exponential complexity. Despite this, methods have demonstrated polynomial or better performance during graph construction. To achieve this, they have implicitly placed restrictions on their exploits, typically by centring their states around hosts. To explore this properly, I will now state it explicitly as the *single-precondition assumption*.

5.3 Attack Graph Assumptions

The assumption of a single precondition means that each exploit requires only one security condition in order to perform. This means that each vulnerability in the system requires only access to a single privilege level to perform; this might be root access to a particular host, or the ability to communicate to a device over a network.

This does not mean that the attack graph can model attack patterns that include only one vulnerability. It is still possible for a vulnerability to enable further vulnerabilities; the resulting attack graphs will still have multiple states and edges – and each edge will not necessarily start from the same vertex.

Because the possibility of performing any exploit can be exclusively determined by a single condition, it is no longer necessary to consider states that have more than one condition. This means that the potential state space of the model, for a system with n conditions, is reduced from 2^n to n.

5.3.1 Single-Precondition Assumption

I now formally define the assumption of a single precondition, using the definition of attack graph and template from Section 3.1.1:

> **Definition:** Single-Precondition Assumption
>
> Let $\mathcal{P}$ be the set of system properties, and $\mathcal{S}$ be the set of states ($\mathcal{S} := 2^{\mathcal{P}}$). Let $\mathcal{T}$ be the set of templates, as defined in Section 3.1.1, so that for a template t, m_t is the domain (or matching set) of t.
>
> Then the template set $\mathcal{T}$ is said to satisfy the assumption if $\forall\, t \in \mathcal{T}$, $\exists\, A_t \subseteq \mathcal{P}$ such that:
>
> $$m_t = \{s \in \mathcal{S} \mid A_t \cap s \neq \emptyset\}$$

A template t under this assumption matches whenever any of a set A_t of system conditions is true.

This enables us to consider each state as a singleton; there is no need to have a "memory" of properties other than that which was most recently added, and there is no need to consider multiple properties at once. This is because it is sufficient, for a given state s, to test if any of the states consisting only of a single property in s match; if $s \supseteq \{p\} \in m(t)$, then $\{p\} \in A_t$ and hence $s \in m(t)$.

The system can therefore be modelled as having only $|\mathcal{P}|$ states (one for each property, as opposed to $|\mathcal{S}| = 2^{|\mathcal{P}|}$). In the worst case, each template can match every state and lead to every other, this means the will be a total of $|\mathcal{P}|(|\mathcal{P}| - 1)$ instances of each template matching (that is, matching a precondition and leading to a unique state). This is equivalent to the corresponding graph for each template being complete (with directed edges in both directions).

The assumption has therefore brought down the vertex set of the resulting graph from potentially $2^{|\mathcal{P}|}$ to $|\mathcal{P}|$, and the edge set from $|\mathcal{T}|2^{2|\mathcal{P}|}$ to $|\mathcal{T}||\mathcal{P}|^2$

This assumption encapsulates the assumptions made by other methods that have significant performance improvements. For instance, in *multiple prerequisite attack graphs* [ILP06], each exploit is modelled with only the precondition that the target host is reachable (given that, as in the example of [ILP06], no credentials are used). As this is determinable from a single set of independent properties, the assumption is satisfied.

5.3.2 Partitioned-Precondition Assumption

Some methods use a middle-ground between host-centric graphs and network-centric graphs. For instance, the work of *Xie et al.* [XCT+09] involves a smaller attack graph for each pair of hosts. While these are not fully single-condition, they can be viewed as making a similar (weaker) assumption, which I formalise as follows:

> **Definition:** Partitioned-Precondition Assumption
>
> Let $\mathcal{P}$ be the set of system properties, and $\mathcal{S}$ be the set of states. Let $\mathcal{T}$ be the set of templates, as defined above, so that for a template t, m_t is the domain (or matching set) of t.
>
> Further, let $\{P_1, \cdots, P_k\}$ be a partition of $\mathcal{P}$ into k disjoint sets.
>
> Then the template set $\mathcal{T}$ satisfies the partitioned precondition assumption if, $\forall\, t \in \mathcal{T}$, $\forall\, s \in m_t\ \exists\, i, j \in [k]$ such that:

$$s \subseteq P_i \text{ and } t(s) \subseteq P_j$$

The condition $s \subseteq P_i$ ensures that each template every matching state of the template is entirely composed of properties from a single partition, and the condition $t(s) \subseteq P_j$ ensures that each template results in a state that is entirely composed of properties from a single partition.

In a similar fashion to the single-precondition assumption, this means the attack graph can effectively be partitioned into the same partitions as the property set. Because no template will ever lead to a state that contains properties from multiple partitions, no such vertex will exist on the graph. This means the number of vertices in the graph is controlled, because combinatorial effects only apply within each partition.

In the special case when $|P_1| = |P_2| = \cdots = |P_{|\mathcal{P}|}| = 1$, the template set also obeys the single-precondition assumption, because each set m_t contains only single-property sets (by the partitioned-precondition assumption) and therefore the A_t required by the definition of the single-precondition assumption can be found by taking the union across the elements of m_t.

The single-precondition assumption does not allow the graph to have any "memory" of more than one property simultaneously. Restricting the graph in this way reduces the number of states, but also potentially limits the exploits that can be modelled. Particularly, models which put particular focus on the host-pairs (e.g. [XCT+09] [HK12]) require multiple-property states, but restrict such states to only those properties which relate to the same host.

The partitioned precondition assumption enables templates to be based on partitions, so that exploits can be enabled by any combination of properties provided they are in the same partition. Transitioning to a state outside of a partition removes the properties associated with that partition, so that the model does not need to "remember" them.

This means we can precisely model combinations of related properties – for instance, on a particular subnet or host – without needing to resort to fully modelling every possible property combination.

The drawback of modelling them in this way is that the attack cannot gain some properties in a host, move to another host, and then return to the previous host to use a combination of the previously-gained properties and the newly-gained properties. For example, assume the system contains two hosts, host A and host B, and that host A has two properties, p_1 and p_2, and host B has one further property

p_3. Normally, we would expect that this could be modelled under the partitioned-precondition assumption, partitioning across hosts. However, this is not possible if there is an template that requires p_1 and p_2 to perform, if p_2 can only be acquired via p_3, which can only be acquired via p_1. In this case, an attacker would acquire p_1, but then move to host B in order to reach p_3. This would be used to move back to host A by reaching p_2, but the exploit requiring p_1 and p_2 would not be possible, because, under the partitioned-precondition assumption, the attacker would have lost p_1 when leaving host A.

Aside from this case, which is probably not common and can be modelled separately if it occurs (for instance, by joining the two hosts into a single partition), the partitioned-precondition assumption enables us to restrict the state space without significantly losing modelling power.

A graph under the partitioned-precondition assumption only needs to contain states that consist of combinations of properties in the same partition. For a partition of $\mathcal{P}$ into partitions of at most size n, the total number of states is now bounded above by $\frac{|\mathcal{P}|}{n}(2^n - 1)$. [1]

In the extreme cases, we have $n = |\mathcal{P}|$, where the entire set is in one partition and we have no benefit: the state space is $2^{|\mathcal{P}|}$ (as it is without the assumption). Equally, if $n = 1$ we have the single-precondition assumption, as explained above, and the state space is bounded by $|\mathcal{P}|$.

For small enough n, this is a significant decrease in state space. More importantly, given a fixed partition size, the size of the graph scales linearly with the number of properties. This means arbitrarily large networks can be modelled (at least, the graph can be constructed) if each component of the network is suitably partitioned – for example into subnets or individual hosts.

5.3.3 Monotonicity

For completeness, and using the same definitions above, I propose the following formalisation of monotonicity:

> **Definition:** Monotonicity Assumption
>
> Let $\mathcal{P}$ be the set of system properties, and $\mathcal{S}$ be the set of states. Let $\mathcal{T}$ be the set of templates, as defined above. Let $m(t)$ be the set of states that

[1] We subtract one from the total number of states of a partition here due to the fact that each partition would have a state representing no compromise, which is unnecessary.

match t, and $t(s)$ be the state resulting from the postconditions of t being applied to s.

Then $\mathcal{T}$ satisfies the monotonicity assumption if $\forall t \in \mathcal{T}$, $\forall s \in \mathcal{S}$:

$$s \subset t(s) \tag{5.1}$$

and also that $\forall s \in \mathcal{S}$ and $p \in \mathcal{P}$:

$$s \in m(t) \implies s \cup \{p\} \in m(t) \tag{5.2}$$

The first of these assertions means that the set of properties in a state is never decreased by a template: the result of applying the postconditions of t to any state which it matches is a state that, at least, contains all the properties of the matched state.

The second assertion means that transitions never require a property to *not* be present. *Ammann et al.* describe this as follows: *"the negation operator is not used to express the precondition of any exploit"* [AWK02], meaning that each exploit can never be made impossible by the presence of a property.

Comparison with Single Precondition

While the single-precondition assumption is not mathematically a stronger version of monotonicity, making the single-precondition assumption effectively implies monotonicity. Under the single-precondition assumption, the attack can lose properties, but the attack can be considered as having kept them; once an attack has a property, there is no reason to need to use it more than once – doing so would necessarily imply the attack contained a cycle, representing the attacker needlessly going in circles. As a result, the single-precondition assumption can be viewed as a stronger version of monotonicity.

Applying the monotonicity assumption (without the single-precondition assumption) results in a reduction in state transitions, as some transitions are no longer possible, and it may result in improvements to modelling: the order in which attackers perform attacks is no longer important. Importantly, however, monotonicity alone does not prevent the system from having exponential complexity, as shown by the earlier example.

5.4 Justification of Single Preconditions

To justify the single-precondition assumption, I will demonstrate a number of exploits as single-prerequisites. In doing so, I demonstrate that a representative model of security can be made under these assumptions.

5.4.1 Smart Home Example

The smart home vulnerability data set includes a collection of vulnerabilities that relate to smart devices that might be in a typical smart home. In this section, I examine this data set and model each as a single-precondition vulnerability – in each case finding a privilege state that, alone, reasonably captures the preconditions of the exploit.

The most common requirement of exploits in the dataset (11 of 26) involves interacting with the target device over the network. As a result, they place no restrictions on the pre-existing compromises of the attacker; as long as the attacker can communicate with the target, they have all they require for the attack. As a result, these can naturally be modelled as single preconditions. Control over any host that can communicate in the required fashion with the target is sufficient and necessary for performing the exploit. As this can be expressed as the disjunction of any of the individual compromises, it is suitably modelled as having a single precondition.

Some of the exploits (2 of 26) require no preconditions at all for attackers to exploit them; these typically involve poor security on cloud systems attached to the devices that either leak information, such as a camera stream (`SHV-16`); or that allow script execution, such as `SHV-1`. This can trivially be modelled as a "single" precondition (in that it has one precondition that matches any state).

In a few cases (3 of 26), the exploits required attackers to have been granted restricted access to the device in order to escalate that into stronger privileges. In these cases, any attacker with the single precondition of restricted access (either granted legitimately or from earlier exploits) would be able to perform the exploit.

The remaining exploits required either physical access to the device (4 of 26) or to be nearby, so that they can broadcast or intercept short-range wireless communication (4 of 26).

The exploits with the most sophisticated precondition were `SHV-7` and `SHV-21` which required the ability to inspect and modify traffic on the network. In both cases, this allowed them to acquire secrets from poorly-secured communications. By

defining the ability to modify network traffic as a condition, both of these exploits can be modelled as single preconditions.

These categories cover the whole set of vulnerabilities, demonstrating how each of the vulnerabilities can be expressed as having a single precondition. The breakdown of each is summarised in Table 5.1.

ID	Device Type	Type	ID	Device Type	Type
SHV-1	Media	None	SHV-14	Security	Nearby
SHV-2	Doorbell	Physical	SHV-15	Security	Nearby
SHV-3	Locks	Network	SHV-16	Cameras	None
SHV-4	Heating	Network	SHV-17	Cameras	Granted
SHV-5	Sensors	Network	SHV-18	Cameras	Physical
SHV-6	Controls	Network	SHV-19	Cameras	Physical
SHV-7	Lights	Inspect Traffic	SHV-20	Cameras	Network
SHV-8	Lights	Nearby	SHV-21	Appliances	Inspect Traffic
SHV-9	Locks	Granted	SHV-22	Solar	Network
SHV-10	Locks	Physical	SHV-23	Cameras	Network
SHV-11	Locks	Granted(1)	SHV-24	Lights	Network
SHV-12	Media	Nearby	SHV-25	Hubs	Network
SHV-13	Media	Network	SHV-26	Cameras	Network

(1): This exploit involved access granted to delivery drivers being used to enter a home. It could equally be performed by an attacker who waited until a delivery driver was making a delivery.

Table 5.1: The types of exploits in the Smart Home set, as categorised above. Each category corresponds to an exploit type that can be modelled as having a single precondition

5.4.2 NVD Examples

To explore the validity of the assumption on vulnerabilities outside of the Smart Home, I sampled a selection of exploits from the National Vulnerability Database [NVD]. This was chosen because of its common use in other attack graph work.

The National Vulnerability Database contains a large dataset of vulnerabilities in a standardised format. To sample the data, I selected data from 2018 and sampled randomly within the vulnerabilities. The vulnerabilities mostly related to the security of networks and hosts, with a few exceptions which were excluded. Those excluded related to specific online platforms, or to cryptocurrency.

I selected 10 exploits to examine in detail, which were as follows:

- CVE-2018-12438: An attack on an Elliptic Curve Cryptography library. An attacker who had access to the host (or virtual machine running on the host) could recover a private key through a memory cache side-channel. This exploit only required the attacker to have access to the host.

- CVE-2018-12776: A popular PDF reading programme had a code execution vulnerability. An attacker could construct a file that, when opened in the programme, allowed arbitrary code execution. This exploit requires no privileges on the attacker to perform. It does, however, require action from legitimate users to enable.

- CVE-2018-12228: Abruptly ending a connection to a VoIP server caused the server to crash. This could be performed by any attacker able to make connections to the VoIP server – which might require authentication.

- CVE-2018-1303: A popular web server could be crashed by a remote user who sent a specially-crafted HTTP header. This required no authentication, so could be performed by any attacker who could communicate with the server.

- CVE-2018-14472: An online content management system was vulnerable to injection attacks from anyone who could access the webpage. This may require credentials to access the system.

- CVE-2018-11400: A home security system was vulnerable to physical tampering. An attacker with physical access could remove the batteries from the device to cause it to silently fail.

- CVE-2018-2930: A high-performance cluster was vulnerable to exploit by unauthenticated attackers with remote access to the system. The attack enabled them to take over the cluster. The attacker only required remote access to the system to perform the attack.

- CVE-2018-6592: A USB storage device protection was vulnerable to exploitation from local users that enabled them to recover data from the device without authentication. Any attacker with access to the host could perform the exploit.

- CVE-2018-5210: A mobile device was vulnerable to a buffer overflow that enabled attackers to discover the device's unlock information. The attackers could perform the exploit remotely with no authentication.

- CVE-2018-8715: An HTTP library was vulnerable to crafted HTTP requests that allowed attackers to bypass authentication. This was possible for any attacker who could communicate with the server.

It is apparent that the majority of these exploits can be expressed as single-precondition vulnerabilities. In many cases, the vulnerability only requires remote access to the target device.

One notable exception is CVE-2018-12776; this required action from the legitimate user, such as opening an attachment from an attacker's email. This highlights a general issue with attack graph modelling: typically the focus is purely on network-based exploits and not exploits that require social engineering. In the context of typical attack graphs, this exploit is expressible as having a single precondition.

The other potential difficulty with single-precondition vulnerabilities is the combination of credential use and access. Some vulnerabilities, such as CVE-2018-12228 and CVE-2018-14472 involve exploits in authenticated systems. In these cases, it is not clear that these can reasonably be modelled in a single-precondition fashion: attackers require *both* access and authentication. This is highlighted in *Ingols et al.* [ILP06] who present a method that explicitly defines credentials as a separate kind of precondition. In practice, this could be resolved by modelling either one or the other, whichever is considered more difficult for the attacker. In situations where the server in question is widely-accessible (such as the web system in CVE-2018-14472) the preconditions need only include the credentials: the server can be assumed to be accessible to anyone.

While the majority of vulnerabilities could be modelled accurately as single preconditions, a discussion of how exceptions could be handled is provided in Section 5.5.1.

5.5 Discussion

The heterogeneous and complex nature of cyber attacks mean that almost any assumption restricts the power of the model to capture the full spectrum of exploits. This was a problem identified by the original proponents of the monotonicity assumption [AWK02]. Despite this, monotonicity is widely accepted and adopted in attack graph models. I believe this is due to the nature of cyber threat modelling in two respects:

1. Firstly, cyber threat modelling is a difficult problem with no ground truth. Making assumptions that allow us to model the majority of things better at the expense of a minority results in a better overall understanding of the situation. Hence, such a trade-off is worthwhile, even if it results in some aspects being missed.

2. Secondly, it is usually possible to compensate for these mistakes; we can replace problematic cases with their worst-case-scenario. As a result we end up with security models that overestimate the risk, but, provided the model is close enough, this is better than no model at all.

In proposing the assumption of monotonicity, the authors pointed out that some exploits could only be completed once – in performing them, they sabotaged their preconditions. While these were not technically monotonic, it was seen suitable to model them as effectively monotonic: the attacker had already performed the exploit, so it would never be necessary for them to perform it again anyway.

5.5.1 Exceptions

There are a variety of exploits that are not straightforwardly modelled as single-precondition exploits.

Any precondition can be expressed in many ways, depending on the level of abstraction being used. The same exploit could require "access to a host", or it could require "access to the correct port", or it could require "a trusted connection to the host". The first of these could be a combination of the other two: so in one abstraction the exploit has a single precondition, and in the other it has two.

Exploits like this can be modelled as single preconditions by selecting a suitable level of abstraction.

In some cases, it is less clear how multiple requirements can be considered as one. For instance, the conditions might relate to different hosts, or one might represent stolen credentials while the other represents network access. In this case, and other cases where preconditions are strictly distinct, it is less clear how we can accurately model the system without significant performance penalties.

One solution is to assume the worst-case scenario: instead of requiring both preconditions to hold, consider the exploit to be possible when either of them holds. This overestimates the attacker's ability to perform the attack, but avoids relaxing the assumption and hence increasing the complexity of the model. This especially applicable in cases where an attack requires one difficult precondition and one more

straightforward preconditions, as the attacker who has achieved the difficult precondition will be more likely to have already satisfied the straightforward one. For example, if an internet-facing webserver is vulnerable to an attack that requires connectivity to it and a forged credential, it is reasonable to model this as only requiring the forged credential.

Alternatively, we can model these as a partitioned precondition where appropriate. This is particularly applicable to some complex host-based models, where attackers can combine as many exploits as required within a single host in order to escalate privilege or transition to another host.

5.5.2 Single Preconditions and Dependency Graphs

As discussed in the introduction of this chapter, monotonicity also led to dependency-based attack graphs, where edges in the graph can be combined with logical operators, so that a vertex can be enabled by a more complex condition on its preconditions.

Under the single-precondition assumption, we assume that every exploit requires only a single condition to be true in order to be successfully exploited. If we apply this to a dependency-based graph, the result is a graph which only contains logical OR operators; in other words, the resulting dependency graph is essentially equivalent to a transition attack graph.

The partitioned precondition has a similar effect, but where AND operators are allowed provided that their operands are part of the same partition of the states.

5.6 Critical Reflection

In this chapter, I examined the state space of attack graph techniques. In particular, I challenged the benefits of monotonicity, and provided an example which is monotonic but with exponential state growth. To explain the performance benefits of many methods that cite monotonicity, I formalised the implicit assumption of single preconditions, and provided a generalised version that encapsulates the assumptions of other methods.

I justified this assumption by studying a sample of real exploits, from the smart home set and from vulnerability databases. These were found broadly to be in line with the assumption. The main exception to this was exploits which required stolen credentials.

5.6.1 Benefits

There is a considerable performance benefit to constructing graphs under the single-precondition assumption. The resulting graphs are orders of magnitude smaller, which means graph analysis can be performed more quickly. Aside from this, the simplicity of the graph makes them naturally more tractable by analysts, meaning they can be more reasonably worked with.

An important benefit is that the resulting graph parallels the underlying structure of the system being modelled - because each state in the network is a state on the graph, our graph is directly a model of network states. This means that the final graph mirrors the structure of the input data. Aside from making it much more understandable by analysts.

While this assumption is made by most other attack graph work, it is not made and analysed explicitly. Doing so here enables us to see precisely the assumption that is being made and its consequences. Further, the stricter partitioned-precondition assumption can be made. This assumption works well with the hierarchical nature of computer networks, which naturally fall into partitions of hosts and subnets. Under the partitioned-precondition assumption, such networks can be modelled finely within partitions without scalability problems.

5.6.2 Drawbacks

The primary drawback of making this assumption is the loss of ability to model privilege states with high granularity. If we model individual software access and compromise to a fine degree, then we will almost certainly require multiple preconditions, particularly where exploits require operating system and application level privileges. In practice, the data used to build attack graphs necessitates this level of modelling regardless.

5.6.3 Conclusion

Making explicit assumptions in security research is important, so that reliability and reproducibility of results can be ensured. Models which implicitly impose restrictions on transitions, to various degrees, make it difficult to critique them and compare them with others.

The work in this chapter aims to make key assumptions clear and explicit. As assumptions, it is likely that there will be practical situations in which they do not

apply. Explicitly stating them aids practitioners seeking to implement attack graph techniques, by allowing them to test and consider the assumptions they make.

Chapter 6

Topology-Inferred Graphs

Attack graphs depend on up-to-date and accurate vulnerability information. The challenges associated with gathering this data are often overlooked by proponents of attack graph techniques, but act as a significant barrier to anyone wishing to implement attack graphs in practice.

Datasets such as the National Vulnerability Database [NVD] are frequently suggested as sources of vulnerability information, but this data is often encoded in formats that make it unsuitable for automatic graph generation, forcing analysts to manually convert data or requiring the use of natural language processing techniques to parse prose [SAT17].

In this chapter, I present a method to creating attack graphs *without* the use of vulnerability information. This leads to an attack graph variant with benefits over typical methods. Firstly, it is considerably easier to construct – both computationally and in terms of data requirements. Secondly, it handles zero-day vulnerabilities as well as pre-discovered vulnerabilities; this is something that few other attack graph methods can do (see Section 2.3.1). Thirdly, the final attack graph mirrors the topology of the network, resulting in a more understandable graphical output for analysis compared to standard attack graphs.

This *topology-inferred* attack graph is very similar to a typically vulnerability-based attack graph, but has a greater number of edges and analysis methods must be adapted to ensure they still provide a meaningful output. This also has a risk of increasing the complexity of graph analysis.

This chapter consists of the following sections:

- Section 6.1: An overview of topology-inferred graphs.
- Section 6.2: A process for the construction of attack graphs from topology information.

- Section 6.3: A discussion with examples of possible levels of compromise for hosts and networks.
- Section 6.4: A worked example demonstrating the construction process.
- Section 6.5: A critical reflection on the the work of this chapter.

6.1 Overview

Attack graphs are normally built via vulnerability information. Each edge in the graph – in almost any formulation – directly corresponds to a known vulnerability. This information is often automatically gathered by scanners, which look for software to match against vulnerability databases. As a result, the attack graph represents the layout of where vulnerabilities are *known to exist* in the system.

There are many problems with this method. Running vulnerability scans can be time consuming, and is unreliable. Data sources commonly cited in other work are often incomplete: in terms of exploits they contain, and in terms of the data that they contain about each vulnerability. An edge on a typical attack graph relies on a vulnerability being known, being reported correctly and in the correct database, and the software with the vulnerability being correctly identified.

Instead of relying on vulnerability scanners, I propose creating attack graphs by assuming that any connection to a device is a potential vulnerability. This entirely circumvents the need for vulnerability information. The resulting attack graph becomes a layout of where vulnerabilities *could exist* in the system.

By considering where vulnerabilities could exist, irrespective of where they do exist, we can include unknown, zero-day vulnerabilities in our analysis. In situations where we have very limited information – such as systems composed of new or little-used devices and software that are unlikely to have reported vulnerabilities – vulnerability-based attack graphs will typically significantly under-report the potential weaknesses in the system. In contrast, this method will work equally well irrespective of the information we have about vulnerabilities.

6.2 Generation of Topology-Inferred Graphs

To build a topology-inferred attack graph, we require the following input data:

- The set of hosts in the system.

- The connectivity between these hosts, indicating which hosts can communicate directly with each other in any capacity.

Both of these are requirements common to all attack graph construction methods. In this case, I do not model precisely the parameters of the connections that can be made between the hosts, instead simply modelling them as connected if they can send and receive packets between each other on any port.

The precise construction of the graph is not necessarily prescriptive, and can be adjusted for specific situations. The key property of the graph is that there should be an edge between any two states which could potentially have an exploit that leads between them.

Through the use of the partitioned-precondition assumption (or, as a stronger form, the single-precondition assumption), we can construct such a graph with a vertex set that scales linearly in the number of hosts and connections. We define a set of properties that relate to hosts, and subsequently construct the set of system properties as the union of these properties: For a set of hosts H, and generic host properties $\{p_1, \ldots, p_n\}$, we create the set of host properties:

$$\mathcal{P}' = \bigcup_{h \in H} \{p_1^h, \ldots, p_n^h\}$$

Where p_i^h is the property relating to host h having property p_i. For instance, the generic property p_1 might correspond to the attacker gaining root access, and therefore p_1^h is the property that the attacker has root access to host h.

We also augment this set with a set of properties, $\{c_1, \ldots, c_m\}$, that relate to the attacker's ability to access the connections between the devices, so that our full set of system properties is given:

$$\mathcal{P} = \mathcal{P}' \cup \{c_1, \ldots, c_m\}$$

In the most basic case, we have one connection property for each communicating pair of hosts, but other means to build this connection set are provided later.

We then define the template set so that it obeys the Partitioned Precondition Assumption. First, we define our property set partition, so that we partition $\mathcal{P}$ into disjoint sets, with one set for the properties of each host, and a singleton set for each connection property.

Within each host partition set, templates should be defined to represent any possible transition between the possible states. These will mostly relate to the attacker

gaining and losing privileges within a single host, often referred to as privilege escalation (or indeed, de-escalation). The template set should also include exploits that map states of a host to the connections properties that relate to that host, and vice versa. These templates represent network traversal in the attack.

Because each of these templates is defined in accordance with the Partitioned Precondition Assumption, the total vertex set of the resulting attack graph that relates to hosts is linear in the number of hosts.

6.2.1 Creating Connection Properties

The number of possible connections between hosts may not be linear. In the basic case presented above, the number of connections may grow with $|H|^2$. However, in most cases this will not be necessary.

The connectivity of hosts can be viewed as an undirected graph with vertex set H and edge set $E := \{(h_1, h_2) \mid h_1, h_2 \in H,\ h_1 \text{ and } h_2 \text{ can communicate } \}$.

We can reduce this set to a set of networks, N, with the following properties:

- Each network is a set of hosts; $\forall n \in N,\ n \subset H$.
- Every edge is represented; $\forall e = (h_1, h_2) \in E,\ \exists n \in N$, such that $h_1,\ h_2 \in n$.
- No other connections are represented; $\forall n \in N,\ \forall h_1, h_2 \in n,\ \exists e \in E$ joining h_1 and h_2.

For instance, we can model a 3-host system as follows: first, assume that the system is unrestricted and any device can communicate with any other. Then each host is connected to both of the other hosts. As a result, the graph can be modelled with one network, which contains all three hosts.

Now instead assume the system is firewalled so that two of the hosts are unable to communicate with each other. Then the system can no longer be modelled as one connection. Such a network would necessarily include all three hosts, because it must include every edge in the connectivity graph, but would therefore place the non-communicating hosts in the same network, implying an edge where there is no connection. Instead, we can model the system with two networks, one between each firewalled host and the unfirewalled host. This is shown in Figure 6.1

A set of networks with these properties is not unique: for a particular connectivity graph, there may be many ways to structure the networks. To some degree, choosing a network set is a trade-off between complexity in the graph construction process (i.e.,

(a) Unrestricted Connectivity

(b) Restricted Connectivity

Figure 6.1: The connections on a 3-host system, shown with and without connectivity restrictions. Circles represent hosts, and rounded rectangles show which hosts are contained in the corresponding connection. In the restricted version, the left and rightmost hosts are prevented from communicating.

running expensive algorithms that find the minimal set of networks) and complexity in later graph analysis (i.e., running analysis on graphs with too many networks).

Finding a possible set of networks is equivalent[1] to finding a partition of the edge set of the connection graph, so that each network is composed of all the vertices that appear in the edges of one of the partitions.

To quickly find a network set, we can construct our edge partition to contain only single edges, so that we find a partition into as many sets as the connectivity graph has edges. This initial approach creates up to $\frac{h(h-1)}{2}$ connections. As a result, we can see that any connectivity graph can be efficiently converted into a set of at most $\frac{h(h-1)}{2}$ networks.

In general, we wish to efficiently minimise the number of partitions in this set, so that the resulting graph contains as few network vertices as possible. In practice, this step will greatly benefit from domain-specific knowledge. For instance, a system composed of a number of (real) networks that allow unrestricted communication can easily be modelled by taking the partitions to be the edges contained within each network.

Some other example topologies (each on h hosts) and their corresponding growth:

- Each host can only communicate with a set of k trusted hosts: the number of networks has an $O(h)$ solution in construction and output size, as each host joins a connection with itself and the trusted hosts.

- Each host can fully communicate, except for k disjoint pairs which have been blocked: the number of networks is $4k(k-1)$, which is the number of maximal cliques on the induced subgraph of vertices which have been blocked.

[1] A network set does not need to be a partition of the edge set, but if an edge appears in multiple networks then a simpler network set can be formed by removing the duplicated edge from one of the networks.

Given a set of networks, we can create the connection property set in the same way the host property set was defined. We define a set of generic network properties, and then union these across the set of networks.

The resulting graph contains property sets which can be partitioned, as per the Partitioned Precondition Assumption, into sets of two types: some contain properties relating to a host, and the others relating to a network.

6.2.2 Edges

Directed edges are added to the topology-inferred graph to represent possible attack steps that an attacker can perform to move from one state to another, as in other attack graph formulations. Typical methods take the topology and match it against vulnerability information in order to create edges that represent known vulnerabilities that have been detected on the network. Here, instead, we do not filter by known vulnerabilities. As a result, the topology-inferred graph is similar to a supergraph of other host-centric attack graphs – but consequently is not predicated on knowledge of vulnerabilities in the system.

I propose three categories for these edges. The precise rules for how edges are formed can be varied based on application, but the following rules are generic:

1. Edges **between states of the same host** – these represent the attacker performing privilege (de-)escalation. For instance, an attacker with a low level of privilege on a host could perform an exploit that allows them to escalate to root privileges. Equally, an attacker with root control can trivially gain a lower level of privilege.

2. Edges **from a connection to one of its hosts** – these represent remote exploitation of a host, and is a key part of lateral movement. Once an attacker can access a network, they can launch attacks against all the devices in the network they can communicate with. For instance, an attacker that can communicate with a database server can then exploit vulnerabilities in the server.

3. Edges **from a host to one of its connection** – these represent a compromised host being used to further the attack, by giving the attacker access to the networks which the host is attached to. This represents an attacker who, having compromised one host, uses it to launch attacks across the network.

Hosts are considered to be vulnerable to attacks over any network they are connected to, and are usable as staging points for further attacks onto any of these networks.

Edge difficulty

For the purpose of later analysis, it is beneficial to label edges as either being *trivial* or requiring an *exploit*.

A *trivial* edge is one that represents something that is possible without any additional weakness in the system. For instance, an attacker with root access to a device can "trivially" disable the device – root compromise represents total control, so an attacker can easily stop the device from functioning. Another example is gaining network access from root compromise. With full access to the device, there is nothing preventing the attacker from sending information to the device's legitimate network connections.

All other edges are considered potentially "difficult" and require an additional exploit.

Figure 6.2 shows an example graph consisting of one host and one network modelled with host states *root*, *functionality* and *availability* and one network state *access*.

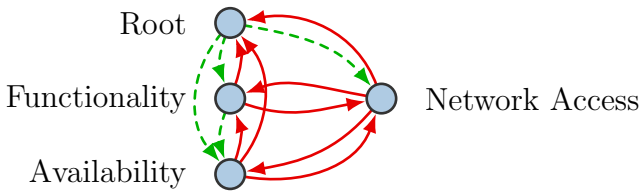


Figure 6.2: Figure showing the edge rules for a device and one of its networks. Trivial edges are shown green and dashed. The full attack graph would be made up of many such subgraphs.

We then add all these edges to the graph. We also add an edge from the initial state to each other state.

Edge Count

In general, for a system of h hosts that communicate over n connections each, with c_h and c_d host and device states (respectively), we have a total edge count of:

$$c_h \cdot (c_h - 1) \cdot h + c_n \cdot (c_n - 1) \cdot n + 2c_h c_n \cdot hn$$

In practice, systems are likely composed of many devices that connect to a single network, and a small set of devices that act as bridges between two different networks, connecting the networks into a hierarchical tree. Under these assumptions, with h single-network devices on n networks, and $n-1$ bridge devices, we have the following edge set size:

$$c_h \cdot (c_h - 1) \cdot [h + n - 1] + c_n \cdot (c_n - 1) \cdot n + 2c_h c_n \cdot n$$

In this case, the edge set growth is linear in n and h.

The resulting graph can then be treated as a worst-case-scenario attack graph, and should be a supergraph of any attack graph generated using vulnerability information.

6.3 Device and Network Compromise Levels

Grouping compromise levels in attack graphs is not a new concept, and has been suggested by a number of authors. *Ammann et al.* [APRS05] use the (ordered) levels `none`, `connectivity`, `pass-through`, `user` and `admin`. Hewett and Kijsanayothin [HK08] provide an example that uses the levels `none`, `user` and `root`. Ingols et al. [ILP06] model vulnerabilities as providing an *effect*, which is one of four access levels: `root`, `user`, `denial of service` and `other`. Aksu et al. [ABD+18] use the privilege levels `admin` and `user`, but further model privilege levels separately for applications, operating systems and for virtual machines.

While there is some variety in levels chosen by other authors, all models include the concept of *user-level* access and of *root-level access*. This is unsurprising as these are fundamental parts of operating system design. Only one of these methods models *denial of service*. This is likely due to a difference in approach between techniques; compromise levels can be viewed purely in terms of their potential as preconditions, or in terms of general network conditions. When considered purely as a precondition (that is, as its ability to act as a requirement for a future exploit) it may be considered unimportant to model *denial of service*. Viewed as a general network condition, and as the end-result of an attack, *denial of service* becomes more important.

The exact levels of compromise are not certain and depend on the desired modelling resolution and on the system being modelled.

6.3.1 Smart Home Compromise Levels

Using these as a basis, I have examined the Smart Home vulnerability set and categorised the pre- and post-conditions of these exploits into levels. This created the following list of device compromise levels:

- **Root** – the device is compromised fully and the attacker has full control over it. A typical example would be the attacker having access to a root shell, so that they can execute arbitrary commands and access arbitrary data. As in other models, this represents the ultimate level of privilege for an attacker over a device. Many examples from the dataset lead to this level of privilege, for instance `SHV-3`, where devices had unchangeable and standardised root passwords.

- **Script** – the attacker can execute restricted code on the device. A typical example is being able to execute arbitrary JavaScript, so that the attacker can modify what the device is displaying. This does not enable the attacker to access data on the device. Such a compromise would likely be followed by attempts at privilege escalation, either through a vulnerability in the scripting restrictions or through tricking a user into granting further access. An example from the dataset is `SHV-1`, which enabled attackers to send crafted URLs to Smart DVRs and run arbitrary JavaScript.

- **Granted** – the attacker has been granted access to a device, either through compromised access to the device's controls or through genuinely-granted access. Such genuinely-granted access could be temporary access to a lock, or the ability to view a camera stream from a device. A typical example in the Smart Home set is vulnerability `SHV-9`, where an attacker could use temporary "guest" access to a smart lock to compromise the device and gain permanent access.

- **Functional** – the attacker has functional access over a device, such as having restricted shell access or by compromising a configuration or administration system for the device. In contrast to *root* access, where the attacker can also manipulate the device itself, the attacker with *functional* access only has control over the device as it was intended to function – for example, opening locks or turning off the device. A typical example would be gaining access to the stream from a camera (`SHV-16`).

- **Denial Of Service** – the attacker can cause the device to no longer function as intended, either temporarily or permanently. A typical example would be using root access to switch the device off. An example from the Smart Home vulnerability set is `SHV-15` where an attacker near to a home security device can spoof connections from the security components and report them as being in a secure state even when they are not.

These levels share parallels with the compromise levels chosen in other work; I model *user* access with more fine-grained levels to reflect the importance of user-level compromise in this scenario. Specifically, *script*, *granted* and *functional* could all be considered subtypes of *user* access.

For the vulnerabilities in the Smart Home set, almost every network-based vulnerability simply required access to the network: the vulnerability was exploited by sending crafted packets, so any device that could send packets to the device was able to perform the exploit. Only two exploits required a higher level of privilege than that, `SHV-7` and `SHV-21`, which required the ability to read traffic from the network. The network compromise levels can therefore be summarised:

- **Access** – the attacker can send packets and receive responses over the network. This may be from compromised root access to a device, or may be from a stolen access key being used to connect a new device to the network. This is the case for almost every vulnerability that requires network access; a typical example is `SHV-3`, in which devices were left with a default, hard-coded root password. Attackers who could connect to the device with telnet were therefore able to gain complete control of the device.

- **Man-in-the-middle** – the attacker can view and manipulate arbitrary packets on the network. This includes packets sent between uncompromised devices on the network. This control exists at a network level and does not necessarily involve compromise of security on other layers. For instance, web content transmitted over HTTPS will not necessarily be rendered insecure by an attacker with this level of compromise. This is quite an unusual level of access to a network; one example that required this level of access was `SHV-21`, where a smart fridge attempted communication with an external server but without the use of certificate pinning, so that an attacker could masquerade as the external server and steal credentials.

The devices were also often able to be compromised by attackers who could physically access the device, or attackers who were close enough to spoof connections. A typical physical compromise is SHV-2, where an attacker was able to physically unplug a doorbell, reset it and pair it with their own device. This caused the cryptographic key for the doorbell's original WiFi network to be leaked. A typical "nearby" compromise is seen in SHV-8, where a smart lightbulb could be forced to rejoin the network by spoofed packets and would leak WiFi credentials in the process.

Only one vulnerability was not grouped into these categories. SHV-21 grants the attacker access to the Google account of the compromised user. As a result, further attacks using these credentials do not strictly relate to the network being modelled and are outside the scope of the model.

The full list of vulnerabilities as categorised is provided in Table 6.1.

ID	Device Type	Pre-	Post-	ID	Device Type	Pre-	Post-
SHV-1	Media	None	Scrp	SHV-14	Security	Near	DoS
SHV-2	Doorbell	Phys	WiFi	SHV-15	Security	Near	DoS
SHV-3	Locks	Netw	Root	SHV-16	Cameras	None	Func
SHV-4	Heating	Netw	Root	SHV-17	Cameras	Grnt	Func
SHV-5	Sensors	Netw	Root	SHV-18	Cameras	Phys	Root
SHV-6	Controls	Netw	Root	SHV-19	Cameras	Phys	Root
SHV-7	Lights	MiTM	Root	SHV-20	Cameras	Netw	Func
SHV-8	Lights	Near	WiFi	SHV-21	Appliances	MiTM	N/A
SHV-9	Locks	Grnt	Func	SHV-22	Solar	Netw	Root
SHV-10	Locks	Phys	Func	SHV-23	Cameras	Netw	Root
SHV-11	Locks	Grnt	Func	SHV-24	Lights	Netw	Root
SHV-12	Media	Near	Root	SHV-25	Hubs	Netw	DoS
SHV-13	Media	Netw	Root	SHV-26	Cameras	Netw	Root

Table 6.1: The full list of vulnerabilities from the Smart Home set with their pre- and post- condition category. Device levels: Script (Scrp), Root (Root), Functionality (Func), Granted (Grnt), Denial of Service (DoS). Network compromises relate to one WiFi network; Access (WiFi) and Man-in-the-Middle (MitM). Near and Phys relate to *nearby* and *physical*, as described above

6.3.2 Initial State

The initial state, that is, where no system properties are true, represents the "beginning" of an attack – the attacker has not yet acquired any privilege or performed any exploits.

Vertices

Not all attackers are equally capable. While subtleties of attacker capability are probably best left to attack graph analysis, rather than construction (see Chapter 7), we can capture general categories of attacker in the structure of the graph.

In particular, different attackers have different choices for entry onto the network. Whilst all attackers can perform attacks over the internet, comparatively few can perform attacks that require them to be within wireless communication range. Still fewer can perform attacks that require physical proximity to devices in the system.

We can model these entry points as distinct initial states with distinct vertices. Each entry point corresponds to a different initial property of the system, and consequently a different starting vertex. Modelling them in this way enables us to consider the effect of each separately.

Unfortunately, because we create an edge from each initial state to each non-initial state, creating multiple initial states can greatly increase the number of edges. These edges are not particularly impactful when considering paths through the graph, however, because they lead from the same vertex which has no paths to it, as a result there is no complexity increase for algorithms such as finding the shortest path – in these cases the number of initial states is essentially irrelevant.

We could, alternatively, model the initial states as a single vertex and consider the initial state of the attacker as part of graph analysis. This is beneficial because modelling concepts like physical proximity separately to system states means we can factor it in to attacks without violating the single precondition assumption. Under the assumption, we cannot model the attacker using physical proximity in conjunction with other privilege states, because this would require two preconditions. But the fact that an attacker is physically proximate to a system does not vary over the course of an attack (at least, it could reasonably be modelled as constant even if the attacker only happens to be present to perform a certain exploit). As a result, there is no need to model it as a system state.

Overall, while it is possible to model the system with multiple initial states, it is likely easier in practice to restrict the graph construction process to considering only

a single initial state, and graph analysis techniques can factor in the differences in attackers – perhaps by generating graphs separately for each possible initial state.

Network

As an alternative to modelling the initial state as a vertex, we can model it in the topology-inferred graph as a network in itself, representing those devices which are externally accessible. This enables it to connect to an arbitrary selection of devices which make external connections.

This is a convenient option for modelling, because the initial state is not treated differently to other networks, other than being the starting point for paths of the attacker.

6.4 Example

To demonstrate the construction of topology-induced attack graphs, I provide an illustrative example, based on the topology information from a small example used by *Albanese and Jajodia* [AJ18]. The network in question relates to an online shopping system. I omit a full description of the hosts (which can be found in [AJ18]) and instead present the graph of the system's connectivity in Figure 6.3.

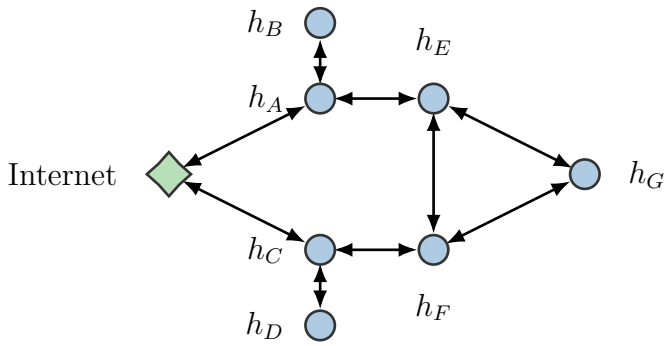


Figure 6.3: The topology graph for an example found in [AJ18]. Hosts are circles, the internet is marked with a diamond. Bidirectional edges indicate connectivity between hosts (and the internet).

This connectivity is based on the example as previously presented: I have assumed that the firewalls block all connections between the database servers (h_B, h_D and h_G) and hosts outside their subnets.

Typical analyses, including the original analysis performed on this example, proceed by combining this topology information with vulnerabilities. Instead I present the topology-induced attack graph, which does not require vulnerability information.

First, the connectivity is modelled as connections of hosts that can all communicate. These are shown in Figure 6.4. The original network is connected as 3 subnets, but due to the firewalling present on the network, these result in 6 maximal cliques, and hence 6 different connections. Of these, 3 are the subnets themselves, and the remaining 3 are connections between the subnets.

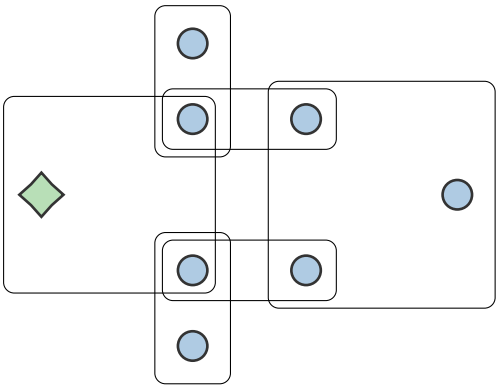


Figure 6.4: The topology of the example in Figure 6.3 converted to a set of 6 connections – each rounded box represents a single connection containing all the hosts inside the box. The diamond, which previously represented the internet, is now the initial state.

The final step is to build the vertices and edges of the final attack graph, through the rules described above. For simplicity in this example, I will model only one level of host compromise, representing total root access; and one level of network compromise, representing network access. The result of this process is shown in Figure 6.5. This may be too coarse to be used in practice (although it is similar to existing host-centric work [APRS05]). The resulting graph is an aggregation of the graph that would be formed if more properties were used: each host vertex in this graph would be split into a small group of vertices, one for each host state, and each edge duplicated to reach these new vertices.

This final graph closely mirrors the original topology, as intended.

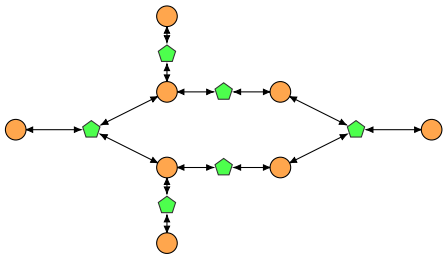


Figure 6.5: The final attack graph as generated from the topology of the example. Orange circles represent hosts (or the initial vertex) and green pentagons represent networks.

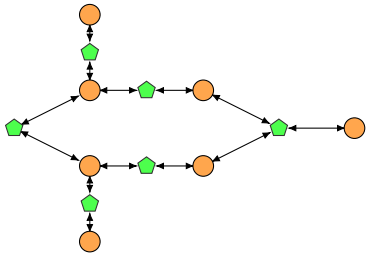


Figure 6.6: The same graph as Figure 6.5, but where the initial state is modelled as a network, instead of a vertex

6.5 Critical Reflection

In this chapter, I examined the possibility of building attack graphs without vulnerability information. My proposed method involves creating a graph from the topology of the system, with the properties of a transition attack graph. Using the Partitioned Precondition Assumption, a set of arbitrary host states can be converted into a graph whose vertex set scales linearly in the number of hosts.

The set of host states can be created based on the modelling level required. In many analyses, it may be suitable to simply choose a single level of compromise, and assume that an attacker with any access to a device has full access. In others, multiple host levels can be derived from expert experience or sample vulnerabilities.

6.5.1 Benefits

Graphs constructed in this way circumvent entirely the need for vulnerability information. This dependency, present in all other attack graph methods, is difficult to satisfy to a suitable degree, and constructing graphs without it simplifies the data requirements considerably, especially in cases where little is known about the hosts on the system.

As a by-product of not using vulnerability information, constructing the graph from the topology is very quick. This means graphs can be generated and updated rapidly, ensuring analysis is always performed on an up-to-date model of the system.

Further, the resulting graph has no bias towards known vulnerability. This makes them highly suitable for cases where few vulnerabilities are known. Combining this generation technique with information about exploit likelihood enables a new variant of attack graph analysis, which focuses on the system's design in an abstract sense, independent of the vulnerabilities that have been discovered.

Large attack graphs often become unwieldy and hard to comprehend for analysts. This is, in large part, because an attack graph often contains no implicit structure. Similar to host-based graphs, topology-inferred attack graphs mirror the structure of the system underlying them, so that they can be naturally formatted and comprehended by laying them out according to the topology of the system.

6.5.2 Drawbacks

The resulting graph is comparatively limited in terms of its vertices, but contains many more edges than a typical graph (in some ways it can be viewed as a supergraph of any other transition attack graph, with an edge wherever an edge could be). While this does not make the construction process take longer, it will impact further analysis of the graph. This is mitigated somewhat by the different meaning of the edges – because the edges are not certain to represent exploits, they can be pruned away more readily without significant loss of likely attack paths.

Because no vulnerability information is used, the graph does not place any significance on edges that represent exploits that have already been discovered. This means that the resulting analysis may miss attack paths that already exist on the graph. To resolve this, it is possible to combine a topology-inferred attack graph and an exploit-based attack graph to add a second set of "known" edges to the graph, creating a graph that considers both zero-days and known vulnerabilities. However, this requires vulnerability information, and graphs built from major databases have been shown to be poor predictors of exploitability [SS15].

Analysis of the resulting graph is not straightforward. Finding paths in the graph is not sufficient to make claims about security, because each edge represents only the possibility of exploitation – a path in the graph is more representative of a path of communication than exploitation. As a result, analysis must take into account the unlikely nature of the edges in order to make meaningful assessments of the system.

6.5.3 Conclusion

Topology-inferred attack graphs offer a way for analysts to quickly build attack graphs that represent their system in a clear way, that does not depend on questionable data sources. These attack graphs encode possible vulnerability, instead of known vulnerability, and as a result can consider zero-days as well as known weaknesses. Analysis of these graphs is independent of the set of known vulnerabilities, which makes it less dependent on which set of weaknesses happen to be known or patched, and more dependent on the architecture and structure of the system.

They are therefore more suitable for studying the security of the design of the system, instead of the security of the currently-known vulnerabilities. They can either be used alone, to build an abstract model of the security of the system's design, or in conjunction with typical attack graphs, so that both aspects are considered.

Constructing the attack graph in this way results in larger, less specific graphs, with potentially many more edges. Further analysis of the graph must take this into account; existing methods will severely underestimate the security of the system, by implicitly assuming every edge on the graph is a real, instead of potential, exploit. In the next chapter, I present parametrised attack graphs, which will form the basis of an analysis method suitable to topology-inferred graphs.

Chapter 7

Parametrised Attack Graphs

Beyond creating the structure of an attack graph, some attack graph generation techniques augment the graphs with additional information that can later be used in analysis. In particular, edges in the graph might be labelled with information that analysis methods can use. This additional information helps inform analysis, so that edges can be treated according to some of the characteristics of their corresponding exploits, instead of homogeneously.

But the restricted data in cyber contexts means that this extra information is often poorly verified and of varying quality. Many techniques (e.g. [XLO+10]) use CVSS scores to augment the graph, such as by adding probabilities to each edge. However, studies have shown this data is unsuitable for this purpose [ZCO11]. *Sommestad et al.* [SS15] also find significant empirical problems with existing methods, many of which are based on these data sources.

In this section, I augment the edges of graphs using a set of parameters. Parameters represent an abstraction of the kind of information typically used to augment graphs. For example, instead of using CVSS data to estimate probabilities for each exploit directly, I propose modelling these via a set of parameters. Each parameter represents some aspect of the exploit, and so the probability of an exploit is derived from the exploit's parameters.

By working with parameters instead of directly with exploit information, the data and work required to use the technique is drastically reduced. This means that it is feasible for analysts to manually verify and work with parameters, and hence greatly improve the quality of the data.

In this chapter, I provide the following:

- Section 7.1. An overview of my novel parametrised attack graph technique.

- Section 7.2. A description of scenarios, parameters and profiles; the building blocks of the method.
- Section 7.3. The formal definition of my proposed method of combining parameters with attack graphs.
- Section 7.4. Deterministic techniques for analysing parametrised attack graphs. These avoid the use of probabilities so that probabilistic data, which may be impractical, is not necessary.
- Section 7.5. Probabilistic techniques that use parametrised attack graphs to create risk-based assessments of security.
- Section 7.6. A critical reflection on the work in this chapter.

7.1 Overview

Attack graphs represent a networked system in terms of its vulnerabilities. Standard attack graphs contain information about the topology of the network, and of the hosts and their software. This enables them to show how these hosts can be abused by attackers to reach their goal.

But an attacker's path through the network is affected by more than just vulnerabilities present on the network. Many factors play a role in determining exactly which set of exploits are possible for a particular attacker and the paths they can use to move laterally in the network:

- **Capabilities of the attacker**: Different attackers have different skill sets; the global connectivity of the internet means networks are under attack from attackers with a significant variety of backgrounds, from novices with freely-available exploit tools to nation states with virtually-unlimited resources.
- **Choices of the network's defenders**: Network defenders deploy defences on the network to mitigate and prevent exploits. But practicalities mean that they cannot deploy every defence they have available and as such must make decisions on what to use. These choices change the landscape of the exploits significantly by preventing some – but potentially enabling others. Even decisions that are not security-related, such as choosing one piece of software over another, can enable attackers.

- **Conditions of the system**: Aside from concious choices of the actors involved, systems naturally face different conditions that change what might be possible on the network. For instance, an exploit might only be possible under certain conditions, like when a server has been taken offline, or when a legitimate user is performing an action.

- **Interdependencies of exploits**: Exploits on the network are not independent. Networks as a whole may be diverse, but exploits often fall into categories and a single attacker might use the same vulnerabilities multiple times as they move across the system. Attackers who can perform an exploit once can presumably perform it again when the opportunity arises.

In order to properly encapsulate the security of the system, attack graphs should incorporate all of these factors into their construction and analysis. Failing to do so often forces a worst-case scenario, which makes analysis less useful by providing looser bounds on the impact of attacks.

To capture this, I propose a model of attack graphs which includes the notion of *parameters* – each encoding information of a fact about the system being modelled. Such information could relate to the attacker, choices of the defenders, or conditions of the system. These parameters can then be combined to create *scenarios*: a scenario is a particular combination of these parameters.

By modelling the parameters, my proposed model captures the interdependencies between exploits and security decisions, and enables analysis to build graph models that demonstrate how the system responds to changes in configuration, and to different attackers. This allows decision makers to select the best set of actions to take to secure their networks, to better understand the risks posed by benign conditions on their network, and to calculate the danger posed by attackers of varied skill sets.

7.2 Scenarios

To begin the process of modelling this information, we define a set of *parameters*. Each of these corresponds to a single fact about the system being modelled, and should be selected so that it is either true or false. I divide these parameters into two categories: *controlled* and *uncontrolled*. These two categories represent very different parts of the scenario, but are modelled similarly.

7.2.1 Controlled Parameters

Controlled parameters relate to aspects of the system which are under the control of the defenders. A fact about the network is modelled as a controlled parameter if its value is chosen by the defender. A typical example would be the deployment of a particular piece of hardware, such as a firewall, or the enacting of a particular strict security policy.

These parameters can affect the security of the network in either direction; deploying a firewall would presumably increase the security, but choosing to enable a network service might decrease it. In some cases, it could not be clear how the security would be impacted. Choosing to use one piece of server software over another might introduce a wholly different set of vulnerabilities.

Examples of controlled parameters might include choices of software or of hardware. They could also include configuration choices, either individual software or in the topology of the network. It need not include every possible choice, and instead should only include the most pertinent choices – those that are currently being decided on, for instance. In fact, for performance is it necessary to select as small a selection as possible.

7.2.2 Uncontrolled Parameters

Uncontrolled parameters are all other aspects of the scenario: everything that is not in the control of the network defenders. Again, these parameters can affect the security in either direction. While many of them might relate to conditions which weaken the network, or where the attacker has greater ability to perform exploits, some of uncontrolled parameters might correspond to situations which improve security. For instance, a particular service on the system being offline might increase the security because it is no longer exploitable.

Examples of uncontrolled parameters fall into different categories. Firstly, there are parameters that relate to the attacker themselves. There could be multiple parameters defined by the skill or capability of an attacker; whether they have access to considerable computing power, or whether they have the knowledge to perform a particular kind of complex exploit.

Secondly, there are parameters which relate to the attacker's relationship with the network. The same attacker might have entirely different routes through networks depending on what they know about it and what access they have to it. For instance,

an attacker might be able to gain physical access to devices on the network, or they might be an insider with legitimate credentials.

Finally, there are parameters which do not relate to the attacker but instead to the network itself. During the normal operation of the network, events occur which may make exploits more or less difficult to perform for the attacker. Some exploits might rely on specific actions by genuine users, or on events such as devices restarting; or the network might be susceptible to more exploits during a particularly high-demand period, where some security mechanisms have to be foregone.

7.2.3 Profiles

To encode this, I give the following definitions:

> **Definition:** Scenario
>
> Let Ψ be the *scenario parameter set*, elements of which are called *scenario parameters*. A subset of $A \subset \Psi$ is called a *scenario*.
>
> If a scenario parameter α is such that $\alpha \in A$, then the scenario A is said to have the parameter α. Parameters in the scenario parameter set can either be *controlled* or *uncontrolled*.
>
> The set of all controlled parameters is given by Ψ_c, the set of all uncontrolled parameters by Ψ_u. As a convention, controlled parameters will be written with a tilde (e.g. $\widetilde{\alpha}$).

A scenario is therefore a particular instance of the possible values of the parameters. A single scenario relates to one possible set of facts about the network and the attacker.

The scenario can be split into two components, the *attacker profile* and the *situation profile*. Because these two components are quite different, it can be useful to discuss them individually. For instance, when configuring a network, the defenders are only concerned with the configuration of the situation profile – they have no control over the attacker profile. Conversely, once these choices have been made, the analysis revolves around the impact of the attacker profile.

An *attacker profile* is a set of *uncontrolled* parameters; a *situation profile* is a set of *controlled properties*. Hence for any scenario, we can uniquely decompose it into an attacker profile and a situation profile (Figure 7.1).

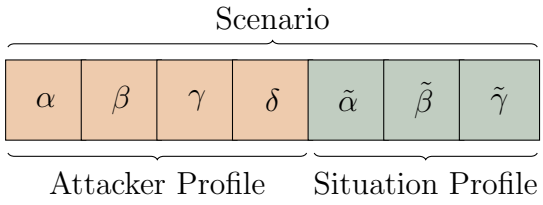


Figure 7.1: Each scenario is uniquely determined by its attacker and situation profiles.

7.3 Parametrised Attack Graphs

Each parameter in the scenario affects which exploits on the system can be performed, either by changing which exploits are present on the network or the attacker's ability to perform them. An exploit might be present in every scenario for which the attacker has the appropriate capability; it might be present in every scenario for which it was not patched out by a defender action.

As a result, each scenario corresponds to a different attack graph. A typical attack graph models a worst-case scenario, where every exploit is possible (at least, under the set of defender choices that was being modelled). Each scenario's corresponding attack graph is a subgraph of this – edges have been removed if they correspond to exploits that are not possible under the scenario.

7.3.1 Formal Definition

The existence of each exploit (and therefore each edge) can be expressed as a condition on parameters. We can define a function $f_e : 2^\Psi \longrightarrow \{0, 1\}$ from the set of scenarios to 0 or 1. For a given edge e, f_e maps a scenario to 1 if the edge is possible under that scenario and 0 otherwise. Such an f_e might be conditional on a single parameter, so that $f_e(A) = 1$ if and only if that parameter is in A, i.e. $f_e(A) = \mathbb{1}_A(\alpha)$ for some parameter α.

Using this, I extend the definition of attack graph:

> **Definition:** Base Attack Graph
>
> A base attack graph G is an attack graph where each edge e has an associated map $f_e : 2^\Psi \longrightarrow \{0, 1\}$.

Each scenario can then be applied to the base attack graph to create a *parametrised attack graph*:

Definition: Parametrised Attack Graph

A parametrised attack graph $G_A = (V, E_A)$, for a base attack graph G and scenario A, is the subgraph of G containing only those edges for which $f_e(A) = 1$, i.e.

$$E_A = \{e \mid e \in E \text{ and } f_e(A) = 1\}$$

The vertex set of G_A is equal to that of G.

An attack graph, a set of parameters and a set of edge condition maps therefore leads to a set of parametrised attack graphs. With n parameters, we have 2^n (potentially) distinct parametrised graphs, each of which is an attack graph in its own right, corresponding to a distinct scenario.

7.3.2 Example

In some cases, edges will be enabled by a single parameter. The following example assumes that each edge is conditional only on a single parameter. In these cases (e.g. Figure 7.2) each edge will simply be labelled with that parameter. In more complex examples the edge would be associated with a logical condition on the properties (e.g. $\alpha \vee \beta$) and the corresponding map is defined to be 1 when this condition holds and otherwise 0.

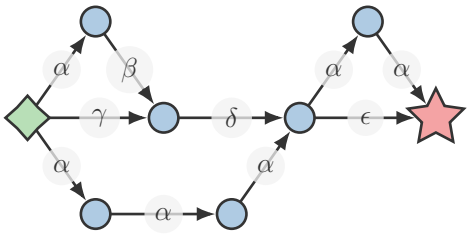
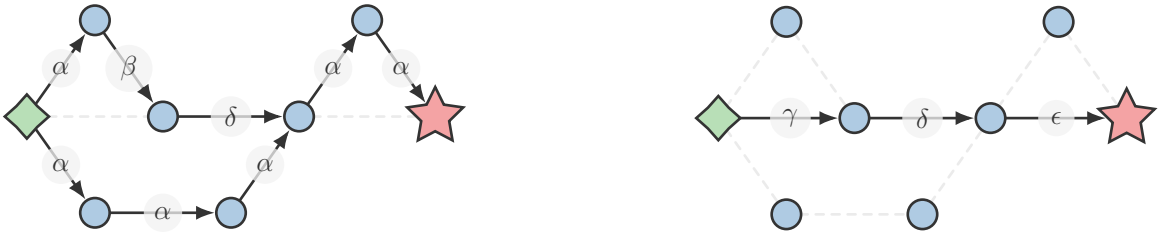


Figure 7.2: A base attack graph with each edge labelled with a single parameter. In this case, an edge labelled with parameter α has corresponding map $f_\cdot(A) = \mathbb{1}_A(\alpha)$

Figure 7.2 shows an example base attack graph. Each edge has been labelled with the single property that enables it. From this graph, we can calculate a parametrised graph for any scenario. For example, Figure 7.3 shows this graph for two scenarios. These scenarios demonstrate the differences possible between two parametrised attack graphs – both share the same base attack graph, but result in very different pictures

of the system's security.



(a) $\{\alpha, \beta, \delta\}$ **(b)** $\{\gamma, \delta, \epsilon\}$

Figure 7.3: The parametrised attack graphs of two scenarios for the base attack graph of Figure 7.2. Base attack graph edges not present in the parametrised graph are shown dashed for clarity.

Overall, we can derive a total of 32 parametrised graphs, each of which corresponds to a different scenario for this base attack graph. These are shown in Figure 7.4. Note that this is not the set of all possible subgraphs; only those that correspond to a profile are included (or possible).

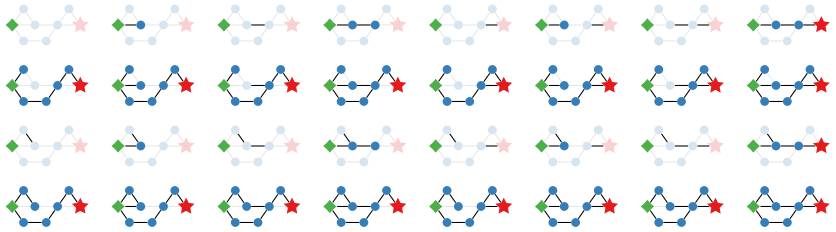


Figure 7.4: A collection of parametrised attack graphs corresponding to the same base attack graph (Figure 7.2). Each small graph corresponds to a separate scenario.

7.4 Deterministic Analysis

Parametrised graphs enable a number of analysis methods. By creating a separate attack graph for each scenario, we can perform typical graph analyses on each graph separately. This enables analysis similar to existing methods: we can pick individual scenarios of interest and analyse them.

Because scenarios are derived from parameters, the set of parametrised attack

graphs has an underlying structure not present in other methods that consider sets of attack graphs. This structure enables us to relate conclusions on the set of graphs to the structure – and hence the parameters that cause them. For instance, we could observe that all the graphs corresponding to a particular parameter have no paths to a vertex of interest. We can then infer that the parameter is responsible for this.

This enables analysts to examine the attack graphs and then draw conclusions about the importance of controlled and uncontrolled properties, guiding them in which decisions to take, and which attackers to concern themselves with.

7.4.1 Metrics

Much of attack graph analysis revolves around metrics: a map from the set of attack graphs to a (typically) real-valued output. These measure some aspect of the security of the system. We might consider longer paths more secure, and hence the shortest path to a vertex in the graph might indicate the security of that vertex: a vertex with a short path is in some way easier for an attacker to reach.

We can expand the application of metrics from single graphs to parametrised graphs. Unfortunately, it is not realistic to directly aggregate the metric values of separate graphs. We cannot, for example, say that the security of the system is the average of the parametrised graphs. This is because such an analysis does not take into account the likelihood of each profile, and hence would be highly dependent on the way the parameters were chosen.

This section provides some methods that avoid aggregation and instead give results based around taking metrics of graphs separately. In order to aggregate metrics, we need to define a probability measure on the set of parameters, a method for which is given in Section 7.5.1.

Special Cases

Some of the most similar work to the methods presented in this chapter involve constructing profiles of particular attackers of interest and modelling these individually. This can be replicated using parametrised attack graphs, by simply selecting the attacker profile that corresponds to the attacker that we would like to model.

This is not limited to modelling attackers: we can also consider any particular situation by applying metrics to the corresponding parametrised graph. This gives us a separate metric value for each situation and hence each can be considered individually for the risk it poses to the system.

In the case where we have a selection of choices of situation profile, and believe our network to be most at-risk from some set of typical attackers, we can examine each way these can combine into a scenario. Table 7.1 shows an example, with a metric calculated for each combination of situation profiles (i.e. the controlled properties) c_1,c_2,c_3 and attacker profiles (i.e. the uncontrolled properties) u_1, u_2, u_3.

	c_1	c_2	c_3
u_1	2	3	4
u_2	4	3	6
u_3	6	3	6

Table 7.1: Each column corresponds to a situation profile, and each row to an attacker profile. The value in the corresponding cell is the metric value of the parametrised graph derived from the scenario formed by combining the situation and attacker profiles.

In this example, we can see that choosing c_2 results in a reliable metric value of 3 across all the attacker profiles, although the best value is achieved by c_1 – albeit only under one attacker profile.

Parameter Selection

Considering only special cases means that defenders have to construct profiles based on their existing knowledge – given that we have a graph for each possible combination of parameter, we can avoid this by applying metrics to every graph and then examining the results. In particular, for a given attacker profile (for instance, the worst-case scenario) we can apply a metric to each possible scenario with that attacker profile. This effectively ranks every scenario profile – and hence every option the defenders have available to them. By doing this, defenders can find the optimal profile and adopt the corresponding set of choices accordingly.

This technique can also be applied in the other direction: for a given choice of controlled properties, we can rank the security impact of all possible attacker profiles and hence discover which profiles stand to have the most significant impact on the network (as measured by the metric).

Bounds over Uncontrolled Parameters

More generally, we can set bounds on metric values. For instance, we might consider it necessary that the attack graph has at least a certain metric value – any graphs with a lower value are considered insecure (or, at least, not sufficiently secure). We can then apply the metric value to each parametrised graph, and therefore determine which attack graphs are suitably secure.

From there, we can find the set of situation profiles which have no insecure parametrised graph – this is the set of situation profiles such that, for any set of uncontrolled properties, the resulting scenario's graph is secure. As a result, these profiles are secure (to the level required) against attack from any possible attacker profile. Such situation profiles therefore correspond to a set of choices which are suitably secure irrespective of the attacker.

We can also find attacker profiles that are insecure irrespective of the choices of the defenders. In these cases, it may indicate that additional security precautions are required.

7.4.2 Reachability

One of the most meaningful properties of an attack graph is that of reachability: if a path exists from one vertex to another, it means there exists a chain of exploits that can be performed by an attacker to move between the two corresponding states. These are particularly important in the case where the first vertex is the initial vertex. In this case, the attacker has a path from having no privileges to the target vertex.

If a vertex has a path from the initial vertex to it, then it is reachable by the attacker.

In typical attack graphs, it is very common for there to be multiple paths from the initial vertex to any other vertex on the graph. This is because they model the worst-case scenario: the attacker typically can perform any exploit on the network. However, when we consider parametrised graphs, which are subgraphs of the base attack graph, there are fewer edges and therefore likely fewer paths for the attacker to take.

We can therefore consider what conditions make vertices reachable, and how parameters and profiles affect this.

Necessary Sets

For some vertices, it may be that any profile that can reach a vertex includes some set of uncontrolled parameters. In this case, these parameters are therefore *necessary* to reach the vertex. This can be used by defenders to gain insight into the nature of attackers that can reach that vertex. If this vertex represents an important target for the attackers, then finding a necessary set of the vertex demonstrates which attackers, and under which conditions, the target can be reached.

If, for instance, a parameter relating to network downtime is necessary for an important target, then defenders could choose to adapt their security during that downtime; making the paths to the vertex impossible during the downtime means that attackers will be unable to reach the vertex at all.

These necessary sets can also be considered across possible situation profiles. If, for instance, an unlikely uncontrolled parameter is necessary to reach the target vertex in all scenarios with a particular controlled parameter, it may indicate that choosing the controller parameter is beneficial for the security of the target.

Edge Burden

In large networks, there will always be considerable repetition of vulnerabilities throughout the network. This means that an attacker performing a series of exploits might be required to repeat the same exploit multiple times against different hosts. More generally, they might be required to utilise the same skills or resources many times during an attack.

This is captured in the model by edges that depend on the same conditions: any attacker that can perform one of the exploits can perform the other, and hence both edges are present.

We can take this further and consider the burden placed on an attacker by a particular edge. We do this by calculating the set of attacker profiles that can reach the start of the edge, R, and the profiles that can perform the edge, P. If $R \subset P$, we can say that the edge places no burden on the attacker: if they can reach the edge, they can perform it. As a result, the exploit effectively provides no resistance to an attacker (we could, if desired, remove the edge and merge the two vertices together to represent this. In practice, this may not be accurate – an edge that requires no extra parameters may still deter attackers for other reasons).

Conversely, we may find that there are many profiles that can reach the edge but not perform it – so that they are in R but not P. In this case, the edge provides a

barrier to some profiles that can reach it. This may be determinable as a particular parameter or set of parameters: it may be that (for example), for $s \in R$, $s \in P \Leftrightarrow \alpha \in s$. In this case, the edge specifically places the burden on attackers of requiring the parameter α. This is not the same as the edge being conditional on α – it may be that the edge poses other requirements as well, but that the other requirements are necessary to reach the edge, and hence true for for every element of R.

We may also find that there are edges for which $R \cap P = \varnothing$. This means that no attacker who can reach the edge can perform it. Assuming the edge is not impossible for every scenario, this implies that something that enables the attacker to reach the edge necessarily makes the edge impossible. By way of example, assume there is a parameter that relates to deploying software A instead software B. The edge may be reachable only by vulnerabilities in software A, but exists in software B. Because only one of the two pieces of software is ever installed, it is never possible to reach *and* perform the exploit.

Superfluous Parameters

We can also take this concept and apply it to parameters themselves: does a particular parameter ever change the vertices or edges that an attacker can perform?

It may be the case that some parameters only provide new edges to the attacker but never new vertices. In these cases, the attacker does gain from the parameter being true, but only by giving them more ways to reach the same vertices. As a result, this parameter could be considered superfluous – it provides very little to the attacker.

Even stronger, a parameter could never provide new edges to the attacker: this means the attacker never gains anything from the parameter being true. In this case, the defenders can regard it as unimportant – potentially removing it from the model entirely.

Controlled parameters can also be superfluous with respect to edges (or vertices): these are security decisions which have very limited impact on security – adding or removing them does not change the edges (or vertices) of the resulting attack graph. These decisions can therefore be taken without security in mind. For example, it might be that the choice of deploying a software package or not is superfluous and hence it is unimportant for security.

We can also condition superfluousness on other parameters. For instance, it might be that we have two controlled parameters, one that relates to deploying general filtering rules, and one that relates to deploying strict rules. After analysis, we discover

that the strict filtering is superfluous on the condition that the general filtering is true. As a result, we can see that deploying the strict filtering on top of the general filtering is unnecessary: there is no further restriction to the attacker by having the extra filtering.

7.5 Probabilistic Analysis

Deterministic analysis provides us with a number of conclusions about the security of the system. However, without a method to measure likelihood we are unable to assess risk; we can only consider the impact of a scenario.

To measure likelihood, it is first necessary to assign probabilities to the model, so that we can assess the probability of a scenario and, hence, the risk posed. This is not always straightforward – probabilities are often hard to justify, especially in a security context where we have little-to-no relevant data. For new systems, historical compromise data does not exist; even old systems will probably not have suffered a statistically meaningful number of attacks.

In spite of this, I offer a method for assigning probabilities to parametrised attack graphs which is designed to mitigate these problems as much as possible. In contrast to other probabilistic attack graph methods (particularly Bayesian Attack Graphs), this method requires fewer probability assignments to be made. This decreases the work required significantly enough that it can be performed manually by experts, unlike methods which require automated estimation.

7.5.1 Assigning Probabilities

The uncertain component of the model lies in the uncontrolled parameters. Controlled parameters are chosen by the defenders, so do not need to be modelled probabilistically. Each uncontrolled parameter, on the other hand, relates to an aspect of the attacker or system which is not fully controlled by the defender.

To estimate the probability of a particular scenario, we must therefore have an estimate of the probability of the uncontrolled properties that make it up. In the simplest case, we can simply assign these directly and assume that each of them is independent.

In many cases this is not an unreasonable assumption; parameters that relate to random failures on the system might well be independent, and are very likely to be independent of parameters that relate to the attacker.

Other parameters, however, will be dependent on each other. An attacker who has one skill may be more likely to possess another: they may be similar, or they may both be the result of training. In these cases, we can model the dependencies between them as well.

Formal Definition

The probabilities in the model are represented via a probability measure $\mathbb{P} : \mathcal{P}(2^{\Psi}) \longrightarrow [0, 1]$ on the probability space $(2^{\Psi}, \mathcal{P}(2^{\Psi}), \mathbb{P})$. Each outcome in this probability space is an element of 2^{Ψ} and is a scenario. Because this is a finite set, we use its power set $\mathcal{P}(2^{\Psi})$ as the set of events. $\mathbb{P}$ is therefore a map from a scenario to the probability of that scenario occurring.

From the perspective of network defenders, controlled parameters are deterministic. As a result, we have that, for any controlled parameter α_c, $\mathbb{P}[\alpha_c] \in \{0, 1\}$. The exact value is determined by whether or not the defenders choose to enact the choice that α_c represents or not.

Under the assumption that (uncontrolled) parameter probabilities are independent, we can specify the probability fully by defining it on the set of singleton scenarios $\{\{\alpha\} \mid x \in \Psi\}$ (i.e. scenarios with only one parameter). We can then extend this to the set $\mathcal{P}(\Psi)$ through independence:

$$\mathbb{P}(A) = \prod_{\alpha \in A} \mathbb{P}(\{\alpha\}) \prod_{\alpha \in \Psi \setminus A} 1 - \mathbb{P}(\{\alpha\})$$

This allows us to measure the probability of any scenario and its corresponding parametrised graph – requiring only a probability assignment for each uncontrolled parameter.

When we do not assume independence, we require additional probability assignments to specify the dependencies between parameters. This could be done through conditional probability tables.

7.5.2 Analysis with Probabilities

We can aggregate across scenarios by weighting them by their probabilities. This allows us to perform analysis on sets of attack graphs.

Expectation

As discussed before, metrics are an important component of attack graph analysis. A metric is a map that takes an attack graph to a value, and is used to assess some aspect of the security of the graph. By taking the expectation of a set of attack graphs, we can generalise this assessment to sets of attack graphs instead of individual graphs.

We want to measure the expected value across a set of scenario graphs of a metric $\mu : G \longrightarrow E$ (for G the set of attack graphs and E an arbitrary measurable space – most commonly $\mathbb{R}$). To do so, we define a random variable $M : 2^{\Psi} \longrightarrow E$, $M : A \mapsto \mu(G_A)$, with G_A the parametrised attack graph corresponding to the scenario A. The expectation of this random variable across a set of scenarios S is the expected value of the metric across that set, and is given by:

$$\mathbb{E}[M] = \sum_{a \in S} \mu(G_a)\mathbb{P}[a]$$

This represents a likelihood-weighted average of the value of the metric. In cases where the metric measures impact, this expectation is a calculation of the risk posed to the system. For example, if the metric μ represents the financial cost of an attacker on a specific attack graph, then M represents the expected cost of a random attacker.

Extending metrics in this way enables us to convert any typical attack graph metric defined for use on a single graph and extend it to the analysis of sets of parametrised graphs.

We can also define metrics which would be somewhat trivial on a single attack graph, but gain meaning when extended. For instance, we can define μ to be the reachability of a particular target vertex v: $\mu(G)$ is defined as 1 if v is reachable from the initial vertex of G and 0 otherwise. On an individual graph, this does not tell us a great deal; however, the corresponding extended metric M gives us the probability that a randomly-selected attacker can reach the target vertex. This avoids the overly-binary nature of typical reachability analysis; in almost any typical attack graph, every vertex will be reachable. This does not necessarily mean that every attacker can reach every vertex.

Parameter Selection

Equipped with the means to aggregate across scenarios, we can re-approach the idea of parameter selection discussed in Section 7.4.1. We can now consider the security of a set of attack graphs as a whole, rather than as individual graphs.

For a set of controlled parameters, we can find the expected value of a metric across the set of all scenarios whose situation profile is those parameters. This represents the expected security (as measured by the metric) of the network defenders taking the course of action corresponding to those parameters. By calculating this value for every situation profile we can find the best course of action, and do so in terms of attacker likelihood.

We can do this by associating to each situation profile the expectation of our chosen metric conditioned by that situation profile. This provides a risk-weighted score of each situation profile. For instance, if our metric is expected cost of compromise, then this technique provides the expected cost of breaches under each situation profile individually. Adding this to the direct cost of implementing each profile, we can find the total cost, both indirect and direct, of implementing each situation profile, enabling network defenders to select the most cost-effective solution easily.

Multiple metrics can be applied in the same way, so that multi-parameter optimisation can be performed, or so that profiles that do not meet policy or required standards can be excluded from the analysis.

Conditional Expectation

Analysts can focus on a specific case – for instance, insider threat – by taking the conditional expectation of a metric, conditioned against certain parameters. By calculating conditional expectation assuming the parameter to be true and, separately, to be false, two separate analyses can be performed and compared to determine the precise impacts of that particular parameter.

This can either be a controlled parameter, so network defenders can determine the particular benefits and weaknesses of a course of action, or it can be an uncontrolled parameter, so that analysis can be focused on a particular attacker of instance.

If, for instance, a company believed an event would significantly increase the likelihood of an insider attack, analysis could be conditioned against the relevant parameter, to perform it under the assumption that the attacker was an insider.

Cumulative Distribution Plots

Understanding the risk posed by attackers to networks is difficult, and techniques for presenting this information to analysts are important. One such method made possible via probabilistic assignments is the use of cumulative distribution plots.

For a given metric of interest μ that takes real values, we can plot the values of the metric against the probability the metric is less than that value. This gives a

visual display of the value of the metric across the spectrum of possible attackers, as weighted by their probability.

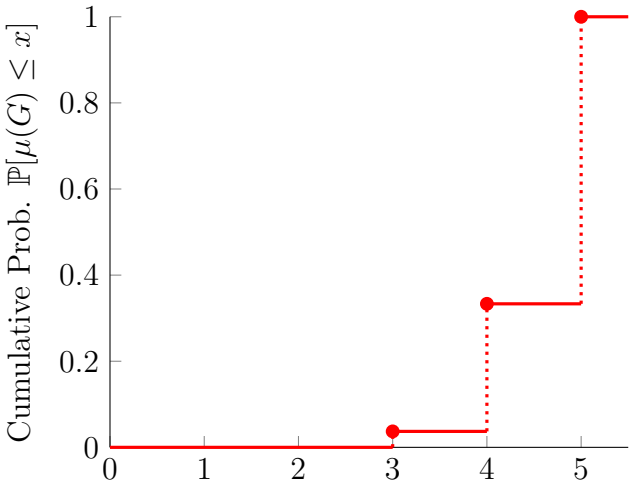


Figure 7.5: The cumulative distribution of *length of the shortest path to the goal*.

Figure 7.5 shows an example cumulative distribution plot for the metric *length of shortest path to goal* (μ)[1]. From this diagram, we can see that all attack paths to the goal require at least 3 edges, and that some attackers require 4 or 5. Very few attackers are able to perform the optimal 3-step attack. However, we can also see that every attacker can reach the goal, as the total probability is 1.

Aside from directly reading values from the plot, we can use the shape to determine readily what indications the metric gives about the security of the network. By conditioning the probabilities against a given situation profile, we can create a cumulative distribution plot specifically for that situation. Figure 7.6 shows two cumulative distribution plots, which could be created by conditioning against two candidate situations that the network defenders are deciding between.

Detailed axes are intentionally omitted here, so that the distinction between the graphs can be seen purely through their shapes; the left graph represents a poor security choice (with respect to the chosen metric), whereas the graph on the right represents a well-secured network. We can see that the left graph rapidly rises for low values of the metric, meaning that it is likely an attacker can perform the optimal

[1]The figures in this section relate to an analysis of the **Simple Illustration** baseline example, although the full derivation of the values is omitted here

Figure 7.6: Two example cumulative distribution plots of the *shortest path to goal* metric, each corresponding to different configurations of a system.

attack. Conversely, the flatness of the right graph means that the majority of attackers cannot perform the optimal attack, and indeed most attackers can only perform far from optimal attacks.

Further, we can see that in the right graph the total probability is not 1 – this means that there are some attackers that cannot perform the attack at all. Conversely, the left graph reaches 1 quickly, indicating that all attackers can reach the goal.

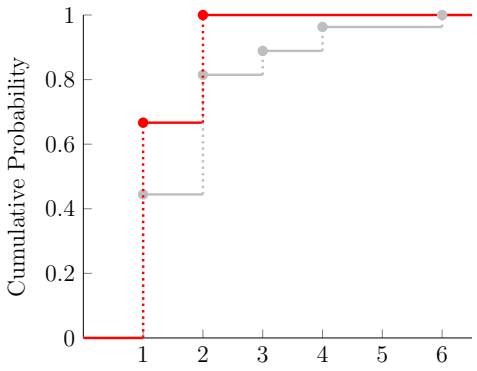


Figure 7.7: The cumulative distribution of the **number of paths to the goal** metric. Without controls is plotted in gray, and with in red.

In Figure 7.7 we see the effect of enacting a control on a plot, this time representing the *number of paths to the goal*. Because lower values of this metric are better (as opposed to the metric above), the ideal graph shape is the opposite to those above. In this case, the controls push the plot upwards, indicating that more attackers are restricted onto a single path.

These graphs enable a rapid inspection of the value of a metric across the whole

spectrum of attackers, instead of taking a single value to represent all attackers. Performing attack graph analysis without parametrising is equivalent to assuming all attackers use the base attack graph, containing every possible edge. On the cumulative distribution plots, this is equivalent to the plot being the worst-case scenario, with every attacker able to perform the optimal attack.

7.6 Critical Reflection

In this chapter, I presented parametrised attack graphs. These attack graphs are augmented with a map from each edge that conditions it on a set of parameters. Each parameter relates to some fact about a possible attack scenario – facts about the attacker, or how the system is configured, or about generic states the system was in. These parameters can be combined in any combination to make scenarios.

For any scenario, the base attack graph can be converted into a parametrised attack graph, representing the attack graph in that scenario – so that the graph only contains the edges that are possible under those properties. For instance, if a parameter relates to turning off a particular vulnerable service, then the exploits of that service will only appear in the parametrised graphs of scenarios that do not contain that parameter.

These parametrised graphs can then be analysed in a variety of ways, which use the structure of the scenario set to make conclusions about the parameters from the resulting parametrised graphs.

7.6.1 Benefits

Modelling each scenario with its own attack graph, gives a conclusion based only on attacks from that profile. These conclusions are then aggregated to give an overall picture of the network. In this way, I avoid conflating parameters and over- or underestimating the consequence of a single parameter.

By defining scenarios as collections of parameters and generate a complete set of scenarios, I ensure analysis is not focused exclusively on expected combinations of attackers or situations. Existing methods typically model a small selection of known or expected attackers and system configurations, which neglects the possible effects of slight deviation from these sets.

Considering attackers in terms of their capabilities leads to useful statements about these capabilities. Necessary or sufficient capability sets provide information to analysts which can aid them both when designing and evaluating networks, and

when reacting to live incidents. This can be performed without ascribing probabilistic information to the attackers, which may be difficult.

While probabilistic assignment is difficult, this method avoids complex probability assignments by shifting probabilities from edges to uncontrolled parameters. In particular, we do not require many large conditional probability tables, as in Bayesian Attack Graphs. The number of potential uncontrolled parameters is also much smaller than the number of edges, meaning fewer probability assignments must be made. A typical attack graph could have hundreds or thousands of edges [NJ04], but could be modelled with a small set of capabilities. By assigning probabilities to a smaller set we make it feasible for analysts to spend more time on each value and consider each dependency. Uncontrolled parameter probabilities may also be reusable between different systems. This would result in significant reductions to the work required.

Having a probabilistic distribution of scenarios enables metrics to be calculated as their expectation, giving a risk-centric view of the system's security, instead of assuming the attacker is able to perform every exploit. This means the consequence of controlled parameters can be more readily examined, and subsequent decisions can be well-informed.

7.6.2 Drawbacks

Assigning probability distributions to attacker parameters is not straightforward. While I believe it is a less-challenging task than that required by similar methods (e.g. individual edge probabilities in most methods, or Bayesian conditional probability tables for edges in Bayesian graphs), it may still be prohibitively difficult, or result in inaccurate or poorly-justified analysis. In these cases, the method can provide deterministic analysis that gives meaningful assessment outside of the difficult of probabilities.

Choosing controlled parameters is comparatively straightforward, as these are predominantly determined by the choices analysts can make. Uncontrolled parameters are less clear, as there is little basis from which to estimate possible attacker capability. Faced with these concerns it is possible to apply the method exclusively regarding controlled parameters. However, the analysis can be improved by adding any uncontrolled parameters – there is no requirement for the uncontrolled parameter set to be complete.

There may be some computational overhead in using this method, but it will be highly dependent on the configuration of the system. In the worst case, to calculate a metric independently on all parametrised graphs the resulting analysis must be

performed 2^n times over, which may be a significant increase. In practice, it is likely that the majority of these resulting parametrised graphs are considerably smaller and so have much faster analysis processes. Ultimately, if analysis was feasible before, performing it multiple times will not prevent it being feasible. Further, because each analysis is (in the worst case) independent, it is highly suitable for parallelisation.

7.6.3 Conclusion

Parametrised attack graphs provide a way to augment attack graphs with additional configuration information. Parameters can be considered as fixed system properties – they are chosen at the "beginning" of a particular attack and remain constant. These parameters then change the attacks possible in the graph for that attack, and can fundamentally alter the structure of the graph.

By building the set of scenarios to be any combination of parameters, the method examines the full spectrum of possible attackers and defender choices, resulting in an attack graph which, in the spirit of attack graph analysis, considers the interactions between every possible component, to explore the possibilities enabled by unlikely or unconsidered combinations.

This process has consequences for the modelling process, both in terms of data requirements and in terms of computational complexity. But the model can be applied with as few or as many parameters as desired. In the simplest case, a single controlled parameter can be used to create two attack graphs that represent the consequences of a single decision. In the most complex case, a set of controlled parameters can be used in conjunction with probabilistically-determined uncontrolled parameters, giving analysts a risk-centric view of every possible combination of decisions they could make.

Chapter 8

Analysis and Validation

The two previous chapters present novel methods for attack graph generation. Individually, they provide new approaches to attack graphs and seek to rectify problems with existing techniques.

Topology-inferred graphs avoid the reliance of attack graphs on vulnerability data, which is difficult to acquire and trust. But without vulnerability information, topology-inferred graphs have a large collection of edges for which little is known. As a result, their analysis is challenging.

Parametrised attack graphs aim to reduce the dependency of methods on detailed information and augment the graph in a way that can facilitate meaningful conclusions. This makes them well-suited for application to topology-inferred graphs. By parametrising topology-inferred graphs, we augment the graph with the information analysts require to perform their work and understand the graph.

In this chapter, I aim to provide a complete overview of the attack graph process. This includes generating the graphs using the methods detailed in this thesis, a collection of techniques for analysing the graph in a static environment, and a collection of techniques for utilising the attack graph of a system under attack.

In this chapter, I provide the following:

- Section 8.1: An overview of the combined technique
- Section 8.2: A discussion of how parametrisation can be performed on topology-inferred graph.
- Section 8.3: A description of a possible smart home use case, divided into a collection of firewalled logical networks.

- Section 8.4: A set of methods through which analysts can understand the security of the static attack graph generated through the combined technique on the smart home.
- Section 8.5: A set of methods through which analysts can understand the security of a dynamic system through its attack graph, that is, the ways in which the combined attack graph technique can assist with the reaction to an in-progress attack.
- Section 8.6: A set of use cases, where a smart home owner could use the above analysis techniques to make practical decisions.
- Section 8.7: A critical reflection on the combined technique and analysis methods.

8.1 Overview

I now present the combination of the previous chapters of work, into a parametrised, topology-inferred attack graph.

To begin this process, we require two data sets. Firstly, we require a collection of hosts to model, together with supplementary information about each host. The supplementary information required depends on the level of detail desired. In the extreme case, the model is possible with no extra information – but by adding in more information we can refine and focus the model.

Secondly, we require connectivity information between the hosts. This should include specifications of which hosts can communicate with each other, but does not need to include specifics of how they can do so (such as which ports are available). In general, this would form a connectivity matrix, where each directed pair of hosts is marked as connected or not.

We are then able to construct the attack graph. We proceed by creating the topology-inferred graph, as detailed in Chapter 6. The result of this process is an attack graph consisting of a small set of vertices for each host and for each connection. In this chapter, the graph is constructed using single host compromise levels, so that it obeys the single-precondition assumption, but this could be equally applied to graphs using multiple compromise levels under the partitioned-precondition assumption.

We now augment this attack graph with parameters, as detailed in Chapter 7. However, this is less straightforward than on typical attack graphs. In typical cases, we create a condition on parameters for each edge in the graph, which determines

if that edge is possible or not. In topology-inferred cases, we have no information regarding each edge, because each edge only relates to connectivity.

Instead, we rely on host information to apply parametrisation to the edges in the graph. Each edge leads to either a host or a network state. Edges that lead to a host state therefore represent the presence of a vulnerability in that host. Because we have no further information, we can only base our parametrisation on the host, and so we use the supplementary host information to characterise the edge. A full discussion of this process is available in Section 8.2.

This combined technique differs in a number of practical ways from attack graphs. While the end result is an attack graph very similar to others, the lack of vulnerability information creates a fundamental difference. Attack graphs model vulnerabilities that are known. They are, in their simplest form, a compact representation of known weakness.

Without the vulnerability information, the combined technique produces graphs which are not based on *known* weakness, but instead based on *expected* weakness. Typical attack graphs make no attempt to model the consequence of zero-day attacks (i.e unknown vulnerabilities). This is a product of their construction. On the other hand, by not relying on known vulnerabilities, the combined technique captures zero-day weaknesses implicitly.

This makes it especially suitable for situations in which the system's components have weaker security, possibly because they have not undergone security auditing, or they were developed without security in mind. In these situations, the devices may have many vulnerabilities that are not yet discovered, and hence many potential zero-day flaws. Models which only account for known vulnerabilities are unsuitable for use in these cases, because the vulnerabilities have not been discovered.

Rapid availability and deployment of security patches also invalidates typical attack graph approaches, because patched systems are unlikely (or at least less likely) to have known vulnerabilities.

8.2 Parameters on Topology-Inferred Graphs

The principal challenge in combining the techniques is developing a parameter set for a topology-inferred graph. Parametrisation involves labelling each edge in the graph with a condition on the set of parameters, such that the edge is considered possible when the condition is satisfied. This is intended to be based on the exploit

the edge corresponds to. If an exploit is prevented by the deployment of a firewall, its corresponding edge would be conditioned on the firewall not being present.

But topology-inferred graphs have no information about the exploits that each edge relates to. As a result, we have to generate parameters from the hosts.

Vulnerabilities involve a weakness in the host that can be exploited by the attacker. This can be in an individual piece of software (or perhaps the device's firmware), or it can be in the protocols and technologies used by the host. In the smart home dataset, some vulnerabilities represented weakness in the firmware of the device (e.g. `SHV-8`, where data was transported without encryption; `SHV-3`, where the device could be reset to an unsafe state). Others represented weaknesses in shared protocols or libraries, so that multiple devices contained the same vulnerability (e.g. `SHV-3`, `SHV-4`, `SHV-5`, `SHV-6`, which all related to the discovery of the same hardcoded password; `SHV-13`, which related to a potential security weakness that could affect any Android device). Aside from vulnerabilities in individual devices and in protocols, some vulnerabilities stem from development mistakes and exist in all devices from a specific vendor (e.g. `SHV-16`, which related to all the baby monitor devices from a particular vendor).

Because of this, I propose creating a parameter set that contains parameters such as device type, vendor, and implemented protocols. Each of these parameters can be thought of as representing a vulnerability being discovered in that device, in the devices from that vendor or in the protocol. Each edge would then be conditioned based on the host that it leads to, so that an edge leading to a host would be considered possible if one of the corresponding parameters is present.

Connecting the edges through parameters in this way helps expose the connections between potential vulnerabilities. For example, if a vendor is compromised and their private keys released, it may enable attackers to compromise any devices they have made. If our system contains multiple devices from that vendor, it might be possible for the attacker to move between them (potentially using other vulnerabilities in addition) and reach their goal. By modelling the potential connection between edges, we capture interactions such as these.

These parameter sets are comparatively easy to gather data for: device types and vendor are straightforward, and implemented network protocols can be ascertained either through publicly available information or estimated via simple scans (e.g. [nma]). Certainly, this data is considerably easier to ascertain than vulnerability information, which requires this information in order to determine.

8.3 Simulated Smart Home

To illustrate and explore possible analysis methods of a parametrised, topology-inferred graph I now provide an example analysis for a simulated smart home environment.

For the purpose of this analysis, I will use the single-precondition assumption and model each host with a single compromise level. This analysis could equally be performed under the partitioned-precondition assumption with multiple compromise levels per host. The device compromise level will represent root access, so that if a device is successfully exploited, it is considered fully under the attacker's control. For networks, I will consider the compromise level "access", where the attacker with this compromise level can communicate with any other device on that network, but not eavesdrop communication between other, uncompromised devices. Finally, I will primarily be concerned with the initial state relating to remote access, so that we are considering attacks that reach the smart home from the internet.

Modelling devices with only a single compromise level represents the worst-case scenario, and assumes that every compromised device is totally compromised. This in itself may be a useful simplification, so that we provide a lower-bound on our system's security: we may underestimate how secure the system is, but we should not overestimate it.

8.3.1 System Definition

The smart home in this example is composed of a set of fictional devices, D, that are intended to represent typical (although not specific) real devices. Each device is assigned a collection of parameters, as detailed in the previous section, including a manufacturer, an operating system and one or more protocols. I use Ψ_d to refer to this mapping, such that for a device $d \in D$, Ψ_d is the set of parameters that d is assigned.

Each device d corresponds to a single vertex, v_d, in the attack graph (under the assumption that we model one compromise level). We can therefore specify in-edges of this vertex; there exists one such in edge for each network this device is part of. We can also define the condition for these edges, so that they exist if any of the properties in Ψ_d are in the scenario:

$$f_e(S) = \begin{cases} 1 & \text{if } S \cap \Psi_d \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Devices on the network are considered to have varied impact when compromised. Devices (such as lights and switches) have less significant consequences, whereas others (say, the front door lock) have considerably larger impact.

8.3.2 Logical Networks

The general case can be expressed in terms of logical networks: each device is part of one or more logical networks, so that it can communicate freely only with devices that it shares a logical network with. These logical networks might correspond to physical networks (particularly in cases with varied network protocols), or they might be the consequence of firewalling. They may correspond to pairs of devices, or the entire network may be unrestricted and consist of a single logical network.

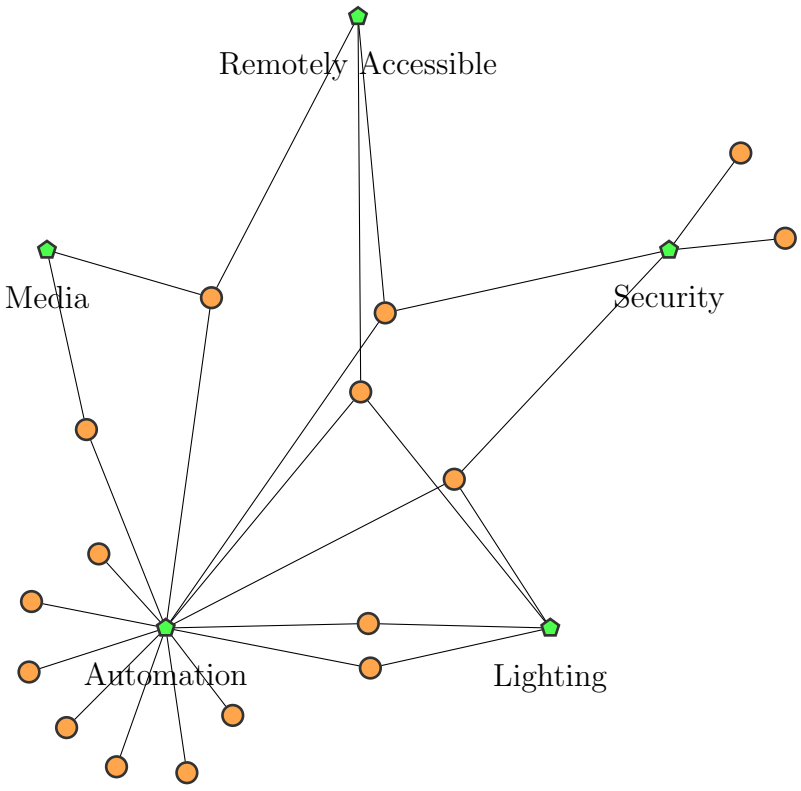


Figure 8.1: An example logical network topology. Logical networks are shown as green pentagons, with hosts as orange circles.

The connections in a topology-inferred attack graph can be formed directly from these logical networks. An example of such a network (shown in Figure 8.1) consists

of five logical networks:

- **Media**, with devices such as a television, speakers, or storage device, which are connected so they can share audio and video content.
- **Automation**, with devices such as sensors, actuators, and controlling devices such as digital assistants, which are connected so they can communicate sensor data and send commands. In this smart home example, almost every device is part of this logical network, except for some security-related devices which are excluded.
- **Lighting**, with devices such as lights, switches and digital assistants, which are connected so they can control each other.
- **Security**, with devices such as the front door lock, security light, and door bell, which are connected so they can interoperate as required.
- **Remotely Accessible**, which contains all devices that are able to communicate directly with devices on the internet.

The majority of devices in this system cannot directly communicate with the internet, but instead rely on local communication to an associated hub or other device. For instance, a front door lock might be remotely controllable over the internet, but this takes place via the front door lock's security gateway.

The Remotely Accessible logical network becomes the initial state in our attack graph, as it contains all devices which can be compromised directly from the internet. Each device is part of one or more logical networks.

The attack graph can be built and analysed for this generic example. We can find relevant sets that demonstrate security properties of the network. For instance, we can again consider the sets of parameters that allow the compromise of a device $d \in D$. A set of parameters S is sufficient to compromise a device d if there exists a path from the initial state to d, that satisfies $f_e(S) = 1$ for every edge e in the path, i.e., if the attack graph parametrised by S has a path from the initial state to d.

The logical network graph can be considered as a bipartite graph, with one set composed of networks and the other of devices. The corresponding attack graph is not necessarily bipartite itself: each device and network may correspond to multiple

vertices, which have edges between them[1]. We can use the bipartite nature of the topology graph to simplify the resulting analysis.

Each path, p, in the graph is of the following form:

$$p = (e_1^1, \cdots, e_{n_1}^1, e_1^2, \cdots, e_{n_2}^2, \cdots, e_1^m, \cdots, e_{n_m}^m)$$

With each subpath $(e_i^k)_{i \in [n_k]}$ corresponding to edges that all begin at vertices in the same device or network. Because the path is non-self-intersecting, we have that n_k is bounded above by the number of vertices that represent the corresponding device or network. In the case where we only model one compromise level, we further have that $n_k = 1$ for all k. Hence for this example we can write

$$p = (e^1, \cdots, e^m)$$

Which is as above but with the subscript omitted. We can construct the full set of paths by first considering all paths that begin and end in a network vertex but otherwise only contain device vertices. For a pair of networks N_1 and N_2, let $p(N_1, N_2)$ be this set of paths. There are $N(N-1)$ possible pairs of networks and therefore $N(N-1)$ sets of the form $p(N_1, N_2)$.

Each of these sets of paths can be calculated independently, and so problems of combinatorial explosion are contained to within the comparatively small subgraphs induced by these paths. With each of these sets calculated, a metric on the whole attack graph can be calculated by aggregating the metric across each subgraph induced by the sets $p(\cdot\,, \cdot)$ and then considering a simplified graph[2] consisting of only a vertex for each network and an edge between each pair of vertices labelled with the metric value for the subgraph induced by the corresponding $p(N_1, N_2)$.

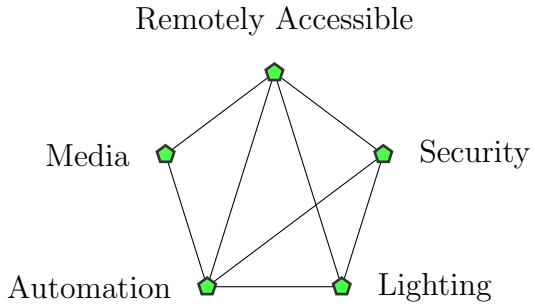
Figure 8.2 shows the networks of the example in Figure 8.1, with an edge between a pair of networks if there is an exploit path directly (i.e. without using another network) between those two networks in the attack graph. The network graphs can be considered as an attack graph[3].

For every path in the attack graph, we can find a corresponding simplified path in this simplified graph, and conversely for each path in this graph we can find a set of paths in the attack graph.

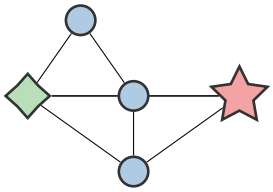
[1] In this example, the attack graph itself is also bipartite because we are only considering one compromise level for each device and network.

[2] The combination of this graph and the induced subgraphs has parallels with Hierarchical Attack Representations [HK12], which contain a topology graph and an attack tree for each pair of hosts.

[3] The graph is drawn as undirected because the symmetry of the definition means that each edge is actually a pair of identical-but-reversed directed edges.



(a) Network Topology



(b) Formatted as a typical attack graph, where the Security logical network is the goal

Figure 8.2: The network topology component of the attack graph, corresponding to the graph in Figure 8.1.

8.4 Static Analysis

With the attack graph constructed and decomposed into a network topology attack graph and a collection of network-pair graphs, we can consider the analysis of these components and how they can be combined.

8.4.1 Probability Information

Parameters in the model can be assigned probability distributions to reflect their likelihood of existence. Because these assignments are not system-specific, they can be estimated independently of the system and then reused in different applications. Parameters could be assigned probability based on prevalence of existing vulnerabilities, reputation, or maturity.

Because of the inherent challenges of probabilistic assignment, much of the analysis described here can be performed without these assignments.

8.4.2 Minimal Reaching Sets

We can derive the necessary parameters for a network, in the same way as a generic parametrised attack graph. These are the parameters that are part of every scenario that can reach a vertex of that network, so that any attack that can reach the network must have all the required parameters.

To reduce complexity, we can calculate the necessary parameters for each pair of networks as the set of parameters necessary for all paths directly between the two networks. The necessary parameters for a network is then the set of parameters necessary for all paths to it in the network attack graph.

If an important logical network has many necessary parameters then the number of distinct classes of vulnerabilities required to attack that network is large. Conversely, it may be that the network can be reached by any scenario that has one of two parameters. In this case, the network has no necessary parameters. This is especially problematic in the topology-inferred case, because each edge is enabled by one of many parameters. We can instead generalise necessary parameters to *minimal reaching (parameter) sets*.

> **Definition:** Minimal Reaching (Parameter) Sets
>
> For a network N in a parametrised attack graph, the minimal reaching set of N, written M_N, is the smallest set of scenarios such that a scenario S can reach a vertex of the network if and only if the scenario contains an element of M_N as a subset.

Using this concept, we can begin to quantify the security of a network (and hence of the devices connected to it), by examining its minimal reaching set.

We can define $m(N) := \min_{s \in M_N} |S|$ as the size of the smallest scenario in the minimal reaching set of N. This indicates the fewest distinct vulnerabilities that must be present in the system to enable the compromise of N. If this number is large, it suggests that the network is resilient to attack.

For a device d connected to networks $n_1, \cdots, n_k$ and with parameter set Ψ_d, the minimal reaching set for the device is the set:

$$M_d := \{S \mid S \in \bigcup_{i \in [k]} M_{n_i}, S \cap \Psi_d \neq \emptyset\}$$

8.4.3 Device Burden

Building on the concept of a parametrised graph's edge burden, we can consider the burden placed on attacks by a device by considering the sets of minimal scenarios that can reach the start vertex of an edge leading to the device, R_d, and the sets of minimal scenarios that can reach the vertex of the device, P_d.[4]

In the topology-inferred case, we can state that $P_d = \{S \in R_d \mid S \cap \Psi_d \neq \emptyset\}$, because any scenario enables the edge leading to the device when it contains any of the device's parameters (the set Ψ_d).

[4]The letters R and P are used to maintain analogy with the edge burden case, where they referred to the set that could *reach* the edge and the set that could *perform* the edge.

If $R_d \subset P_d$, the device places no burden on the attack; every scenario that can reach an edge leading to the device can perform that edge. This can be illustrated with an example: consider the case where a system consists of two logical networks: network A, which allows remote access and one other network, B. The system contains two devices which are identical except that the first, d_1, is part of A and B and the second, d_2, is part of only B.

Then we know that $\Psi_{d_1} = M_B$, because any parameter in Ψ_{d_1} is sufficient to compromise d_1 and therefore to access the networks it is part of. As a result, we have

$$R_{d_2} = P_{d_1} = 2^{\Psi_{d_1}} = 2^{\Psi_{d_2}} = P_{d_2}$$

Hence the device d_2 places no burden on the attack because any attack which can reach the device can compromise it.

This can be characterised as a property of an individual parameter on a device:

> **Definition:** Burden-free parameter
>
> A parameter α is burden-free for a device d if, for every network n that d is part of, we have: $\alpha \in M_n$

And equally

> **Definition:** Burden-free device
>
> A device d is burden-free if it contains a burden-free parameter.

The existence of a device with no burden essentially connects the two logical networks into one from the attack graph perspective. Any attack that reaches one network can use the zero-burden device as a bridge to reach the other. If the system aims to maintain defence-in-depth, then the discovery and removal of these devices is key to preventing system attackers from using a single parameter to penetrate multiple layers of defence.

8.4.4 Attack Surface

The attack surface of a network is the breadth of vulnerabilities that can be used to gain access to that network. We can define the (direct) attack surface between two networks in the same way:

Definition: Direct Attack Surface

The direct attack surface $A(n_1, n_2)$ between networks n_1 and n_2 is the set of minimal scenarios that enable a path from n_1 to n_2.

In the single compromise level case, this set is precisely the union of the parameter sets of their shared devices. It is also symmetric, with $A(n_1, n_2) = A(n_2, n_1)$ for any pair n_1, n_2. We can slightly overload this definition to specify the set of parameters which enable edges from the network to any of its devices: for a network n, the set $A(n) := A(n, n)$ is the set of minimal scenarios which enable an edge from n to a device and then back to n.

Of particular interest is the attack surface of the system as a whole; the set of parameters which enable an attacker to begin traversing into the system. In a general case, we can define $T(n)$ to be the union of the direct attack surfaces $A(n, n')$, for every network $n' \neq n$. This is similar to the set $A(n)$, although $A(n)$ also includes parameters that enable the compromise of devices that have no other networks. The attack surface of the whole system is then $T(n)$ when n is the initial network.

If the aim is to prevent any access to any device in the system, the set $A(n, n)$ (for initial network n) should be minimised, as each scenario in this set represents a means for an attack to compromise a device from the initial state. If the aim is to prevent any access that might lead to further compromise in the system, then the set $T(n)$ is relevant: each scenario in this set represents an attack that can reach a non-initial network.

8.4.5 Failure Sets

For a given scenario S, we can examine the reachable set $R(S)$ of the scenario, which is the set of devices (and networks) for which the scenario's attack graph has a path to from the initial state. We can also approach this from the other direction: given a set of devices, which scenarios can reach those devices. I term this the *failure set* of the devices. Given a set of devices $D' \subset D$, the failure set for those devices is given

$$F(D') := \{S \subset \Psi \mid D' \cap R(S) \neq \emptyset\}$$

We can define $F_x(D')$ to be the x-parameter failure set:

Definition: x-Parameter Failure Set

For a set of devices D', we define the failure set as

$$F(D') := \{S \subset \Psi \mid D' \cap R(S) \neq \emptyset\}$$

And the x-parameter failure set, for $x \in \mathbb{N}$:

$$F_x(D') := \{X \in F(D') \mid |X| = x\}$$

Thus the single-points-of-failure for the important devices is given by $F_1(D')$, and this set consists of parameters which, alone, are sufficient for an attacker to compromise the devices in D'. If this set is non-empty, then the system has a single parameter which is sufficient to compromise the important devices, and therefore has no depth in its defence. Where the attack surface sets measure the breadth of attacks possible on the system, the failure sets represent the depth of those attacks.

The size of the smallest element of $F(D')$ (equivalently the least x such that $F_x(D')$ is non-empty) is the number of parameters necessary for a compromise on the set D', and is equal to the value of $m(D')$, the size of the smallest scenario in the minimal reaching set of D'.

8.5 Dynamic Analysis

In addition to considerations of system design, we can analyse an attack that is already in progress. To do so, we assume some knowledge of the attack: that the attack has been (reliably) detected as having compromised a particular device. The problem for the system defender is now to consider what the consequences of this are, both in terms of what may have happened before this device was compromised and what can be anticipated to happen next.

I describe this attack as being in-progress, as though the attacker had just performed the detected compromise of the device. This is simply as a convenience, and in practice the detection could have happened in the past.

8.5.1 Parameter Inference

Given an in-progress attack, there exists a particular "true" scenario, S_A, which represents the actual known parameters of the attack. Our first task in understanding

the attack is to define conditions on S_A that we know to hold (under the assumptions of the model, at least).

We know that the attacker must have a path from the initial state to the compromised device d. We can therefore say that there exists a scenario $S \in M_d$ with $S \subset S_A$.

In some cases, this S will be unique: the only way to reach device d is through that particular set of properties. In others, it may be that there is a small set of choices for S. In these instances, we can proceed with this information.

However, it may be the case that there is a large number of possible sub-scenarios that could have enabled S_A to reach d. Without making assumptions about how the attacker chooses exploits in the network, we cannot make precise claims about which of these is the most reasonable; such assumptions are outside the scope of this thesis, although it is worth noting that in some cases they may be fairly apparent (for instance, it may be the case that the device could be reached directly with a single parameter or via a complex chain of parameters in a long series of exploits).

Instead, we can use the probability distribution of parameters, if defined, to determine the (a priori) most likely possible scenario, although this method risks making probabilistic inference whilst ignoring important evidence.

8.5.2 Compromise Inference

With a (potentially probabilistically-weighted) set of estimates of S_A, we can make claims about which devices and networks the attacker could have reached before compromising the device. We can also make claims about which other devices and networks may be compromised after the detected compromise.

First, we can infer that the attacker must have compromised any vertices that are part of every path between the initial state and the detected compromise. This (and weaker versions, in cases where there are no such specific shared vertices) can be used to direct us towards discovering vulnerable devices, deciding which devices should be considered compromised and re-secured, and to estimate the potential impact of the attack.

Further, we can update our estimations of the risk of the system. We can consider the compromised device as the "initial state" for the remainder of the attack, and find the possible consequences of the rest of the attack.

8.6 Potential Use Cases

The work so far in this section presents a toolkit of theoretical methods with which a parametrised topology-induced attack graph can be analysed. I now provide a collection illustrative use cases to serve as a demonstration of situations in which these techniques could be applied. A further use case based on the implementation is presented in Chapter 9.

In the smart home situation, the home owner may wish to make security-conscious decisions without any knowledge of how to perform a security risk assessment. As a result, such an assessment must be performed independently of any specific expert knowledge. The attack graph analysis presented here is well-suited to making such analysis, and to providing automatically-implementable security decisions.

Device selection

In this static use case, the user is presented with a choice between two new devices, d_1 and d_2, to deploy on the network; the two devices are functionally identical, and so the user would like to deploy the device which presents the best security choice. The two devices require the same connectivity and so are part of the same logical networks.

The security of this device can be considered in two regards: how much impact to the user there is if the device is compromised and to what degree the device can be used as part of an attack on another part of the system. Given the two devices are functionally identical, it is reasonable to conclude that they have the same impact when compromised. As a result, both variants of the security of the device are reduced to the question of how easy the device is to compromise.

In this model, this is equivalent to asking what the burden of the device is on scenarios that can reach it. Because the two devices are part of the same networks, we have $R_{d_1} = R_{d_2}$, and so when considering the sets of scenarios that can compromise these devices, we can consider $P_{d_1} = \{S \in R_{d_1} \mid S \cap \Psi_{d_1}\}$ and $P_{d_2} = \{S \in R_{d_1} \mid S \cap \Psi_{d_2}\}$.

If $P_{d_1} = P_{d_2}$, then the two sets are compromised by precisely the same scenarios and there is no security justification to choose one over the other. This does not necessarily imply the two devices are identical – if both devices are burden-free then their other parameters are irrelevant and could be distinct.

If $P_{d_1} \subset P_{d_2}$ (and vice versa), then every attacker that can compromise d_1 can compromise d_2, so there is no scenario in which d_2 is secure when d_1 is not. As a

result, d_1 is the better choice from a security perspective. This (non-strict) inequality can be found immediately in the special case when $\Psi_{d_1} \subset \Psi_{d_2}$.

Outside of these clear cases, we are left with two sets, $E_{d_1} := P_{d_1} \setminus P_{d_2}$ and $E_{d_2} := P_{d_2} \setminus P_{d_1}$, which we have to make a security comparison between. Returning to the principles of Chapter 4, we cannot directly compare these sets without the means to compare individual parameters. Such a comparison can be made when a probability distribution is defined, where we can assess the relative risk of the two devices by examining which of the two sets E_{d_1} and E_{d_2} is more likely.

Performing this comparison is made slightly more complex in cases where the devices are used as a transition between logical networks, and in cases where parameter probabilities are not independent. In either case, we cannot assume that $\mathbb{P}[E_{d_1}] < \mathbb{P}[E_{d_2}]$ implies that d_1 is the better choice. Instead we have to perform the full analysis on a per-profile basis, and calculate the reachability sets for each profile. In a typical parametrised topology-inferred attack graph this should not be prohibitively computationally expensive (and especially so in single-compromise level cases).

Network selection

In this static use case, the user has a single device but would like to determine the logical networks that the device should be part of.

In this instance, we can use controlled parameters to condition the connections the device makes to each of the possible logical networks. The analysis proceeds by following the methods for assessing controlled properties as in Chapter 7.

Of particular interest in this case are sets such as the attack surface. If the device is to be connected to multiple networks, then we can consider the direct attack surfaces with and without the device between each pair of these networks. If the device does not increase the direct attack surface between a pair of networks, then it can be connected to that pair without increasing the number of scenarios that can traverse the network.

Dynamic Reconfiguration

As an example dynamic use case, consider a system which is capable of rapidly and reliably detecting compromise on a particular device. In this case, we would like the system to automatically adapt to this detection, without user input, to ensure that the attack cannot proceed towards important devices.

As detailed in the dynamic analysis section above, the first step is to make inferences about the possible scenario of the in-progress attack. With such inferences, the system can then perform inference of possible future compromises. In the case where this includes the important devices, dynamic reconfiguration can occur (such as disabling devices, or temporarily removing a device from a logical network). To choose which reconfigurations are necessary, they can be specified as controlled parameters and the most suitable controlled profile can be determined and enacted.

8.7 Critical Reflection

In this chapter, I combined the work of the previous chapters, to demonstrate how the analysis techniques presented in parametrised attack graphs (Chapter 7) could be applied to topology-inferred graphs (Chapter 6).

Parameters on the topology-inferred graph were created from host information, so that each edge was possible if one of its host's parameters was true. Host parameters can then be selected to represent possible sources of exploits, which would be shared between hosts with similar characteristics.

As a result, the model captures the interactions between vulnerabilities, both in terms of the combination of different vulnerabilities (as in typical attack graphs) and in terms of how a single vulnerability can affect multiple devices.

This two-level interaction modelling enables the attack graph to indicate which sets of vulnerability are the most consequential for the system, helping the end-user avoid creating unintentional single-points-of-failure, and indicating the anticipated levels of compromise from a possible set of vulnerability discoveries.

8.7.1 Benefits

Performing the analysis through a combination of these two techniques avoids reliance on known vulnerability datasets, gaining the benefits of topology-inferred graphs. Particularly in the smart home use case, this is important as the system can have many devices from a variety of sources, which may have no known vulnerability information. It has been shown that even devices considered secure can contain easily-discovered weaknesses [CdR16].

Parametrising the graph enables conclusions to be drawn about the consequences of vulnerability discovery, by considering the impact of the relevant parameters. This enables the system to be configured in such a way that it minimises the consequences of both single vulnerabilities being discovered, and of sets of vulnerabilities.

Modelling vulnerabilities as being present in potentially multiple devices means that the system can be designed to avoid small failure sets, resulting in defence-in-depth, with attackers requiring multiple, distinct vulnerabilities to be possible in order to compromise valuable components of the system.

Topology-inferred graphs can be rebuilt rapidly upon system changes, so that the analysis can always be performed with the most up-to-date version of the attack graph. Attack graphs built from vulnerability information cannot necessarily be updated quickly, and may require a long rebuilding process on each update.

8.7.2 Drawbacks

The analysis process is made more complex, both by the use of topology-inferred graphs and by parametrisation. Under the assumption that the smart home consists of hundreds of devices, the model should still be feasible to analyse, particularly if duplicated devices are grouped together.

The lack of vulnerability information means that existing vulnerabilities are not flagged as being especially problematic to the security of the system. While the analysis could be adapted to include such information, doing so would require vulnerability data sources which are unreliable.

While the method presented here does not require vulnerability information, it is dependent on properties of the hosts, such as the device model, manufacturer, and ideally additional shared dependencies it may have. This is a much less strenuous requirement than vulnerability information (indeed, the first step in finding vulnerability information is finding these properties so they can be matched against vulnerability databases). However, finding some of this information may be difficult for end users. Despite this, much of the data can be automatically gathered by scanners.

8.7.3 Conclusion

The analysis in this chapter demonstrates a means by which an automated tool could build and analyse attack graphs in order to answer security questions that a smart home user may have. For instance, a user might be faced with two near-identical devices from different manufacturers. The methods presented in this chapter could automatically determine, from a small amount of comparatively easily-acquired device information, which device is the best for the security of that particular system. Equally, other decisions can be made, such as which network a particular device should be connected to.

While the technique here is presented for the smart home, this technique could equally be applied to any other system, such as corporate networks as in most attack graph examples. Another applicable use-case would be industrial IoT systems, which share some common features with the smart home, but can be applied with expert input, enabling more carefully constructed parameter sets, and more sophisticated analysis that requires manual input.

Chapter 9

Implementation and Use Case

This chapter presents a validation of the smart home use case, through an implementation of work of the previous chapters. This demonstrates the applicability of the work in this thesis to realistic cases, using readily-available information about the smart home to provide security assessments and actionable suggestions to a smart home user.

To support this I provide a worked use case, showing the full extent of the information required by the model, and demonstrating some of the analysis techniques presented so far, together with how the smart home user could react and update their system, both to architectural weaknesses discovered during the analysis, and to new vulnerabilities reported by other sources.

9.1 Implementation

I developed an implementation of the combined technique (as presented in Chapter 8) in Python[1]. The implementation takes the form of a tool that analyses the security of a system, which is provided as a set of hosts (together with their associated parameters), a set of logical networks on which the devices are able to communicate, and the list of devices communicating over each network. It can also be provided with (independent) probabilities for each parameter, so that it can perform probabilistic analysis (although deterministic analysis can be performed regardless of whether probabilities are available or not). As in the previous chapter, the implementation considers the case when devices and hosts have a single compromise level.

[1]Python 3.6, https://www.python.org/

9.1.1 Functionality

The implementation works by maintaining a model of the (base) attack graph, the attributes of which are calculated on demand and stored. This means each calculation is only performed when required and is stored so as to avoid being unnecessarily recalculated. Specifically, for the given system, the implementation can calculate the following:

- **Reachability**: reachability in the graph, or given a particular scenario, can be calculated, determining if a given parametrised (or base) attack graph contains a path to a given device.
- **Direct Attack Surface of networks A and B**: the set of parameters which enable transitions between two networks, representing the set of possible parameters for which the corresponding parametrised attack graph contains a path from the vertices of network A to the vertices of network B (or vice versa) without passing through any other networks.
- **(Indirect) Attack Surface of networks A and B**: as with the direct attack surface, but where the paths can pass through other networks before reaching B. This is calculated efficiently by first finding the direct attack surfaces and then finding paths on the graph formed with a vertex for each network and edges representing the direct attack surfaces between each network. This includes calculation of all possible paths between the two networks.
- **Device Attack Surface of network A**: For a given network, the implementation can find the sets of parameters which enable an attacker with access to network A the ability to compromise one of the devices on that network. This can further be used to find the external attack surface, which is the sets of parameters that enable an attacker with no prior access to compromise one of the devices in the system.
- **x-Parameter Failure Sets of device D**: For a given device, the x-parameter failure sets for that device can be calculated, which are the sets of parameters which are sufficient for any parametrised attack graph with those parameters to have a path from the initial state to that device. In other words, these are the parameters necessary for an attack on that device. Of particular interest are these sets for which $x = 1$, which represent single points of failure, i.e., single parameters which are sufficient for the attack to compromise that device.

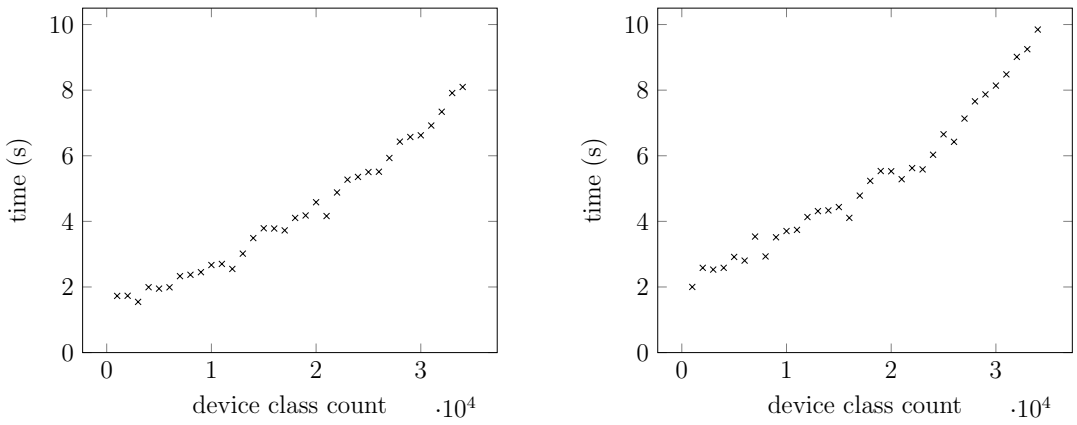
- **Most Likely Exploit of device** D: As an example, the most likely exploit path to the device D can be calculated, returning the failure set of D with the highest probability. Note that this is not necessarily the same as the minimal failure set, because the minimal failure set can contain unlikely vulnerabilities.

9.1.2 Performance

The implementation performs well on smart home-scale datasets. For the use case, containing 17 classes of device on 5 logical networks, all calculations were performed in a few milliseconds on a standard desktop computer. Constructing the graph was, as expected, efficient on all sizes of system due to the way the graph is constructed.

Subsequent analysis was typically efficient, and calculating attributes such as reachability or finding direct attack surface between networks could be performed without significant computational overhead.

Validating theoretical expectations, enumerating all possible paths was more expensive, but not prohibitively so, particularly on the scale of the smart home use case. This acceptable performance was maintained as the number of device classes increased greatly, with sets of as many as 35,000 classes of device being computed in a manageable amount of time (see Figure 9.1).



(a) Each device on two networks.

(b) Each device on three networks.

Figure 9.1: Time taken to find all attack surfaces and the full failure set for a randomly-chosen device, as **the set of device classes increases**. System contains 5 logical networks, and each device has 2 parameters drawn randomly from a set of 20

Increasing the number of logical networks has significant performance penalties,

although the model is still feasible for smart home-scale numbers of devices. Figure 9.2 clearly demonstrates the exponential complexity of finding all paths on the set of networks.

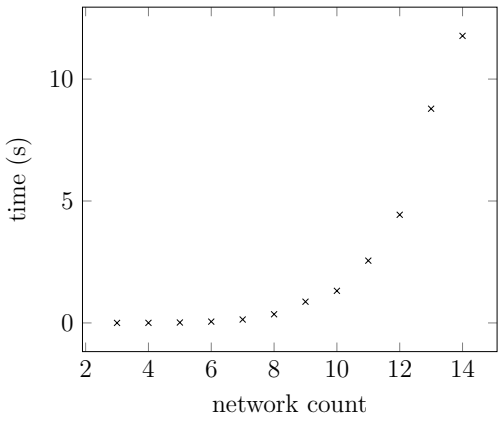


Figure 9.2: Time taken to find all attack surfaces and the full failure set for a randomly-chosen device, as **the number of logical networks increases**. System contains 30 device classes, and each device connects to 2 networks has 2 parameters drawn randomly from a set of 20

This performance penalty is primarily caused by the number of interlinked logical networks; in this randomly generated example, every network will very likely have a connection to every other, through a shared device. As a result, when the implementation attempts to find indirect attack surfaces between each pair of n networks, it is equivalent to enumerating all paths on the complete graph K_n, causing exponential growth.

This performance is greatly improved when the number of direct connections between networks is reduced. In the case where the logical networks are arranged in a tree, so that there is exactly one path between any pair of networks, we have considerably better performance. This can be seen in Figure 9.3, which is running the same simulation as Figure 9.2, but where the networks are arranged as a tree.

In this case, the total time taken is less than a tenth of a second in most cases. While there is some exponential growth, this is unavoidable because it represents the enumeration of the possible combinations of parameters along each discovered path.

Varying the number of parameters, both in terms of the total population of parameters and the number of parameters given to each device class, does not significantly impact performance, as demonstrated in Figure 9.4.

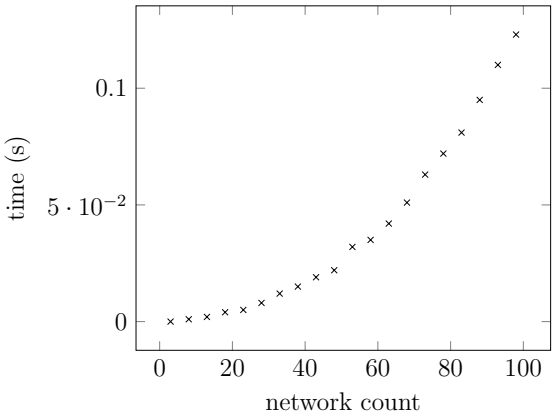


Figure 9.3: Time taken to find all attack surfaces and the full failure set for a randomly-chosen device, as the number of logical networks increases, where **the networks are arranged as a tree**. System contains 30 device classes, and each device connects to 2 networks has 2 parameters drawn randomly from a set of 20

Overall, the implementation performed well in every realistic case, building and enumerating attack paths in less than a second, except in cases with large numbers of device classes or on many logical networks that could all communicate (≥ 10). Even in these cases, the runtime was still manageable, on the order of seconds, provided the number of logical networks was not too large.

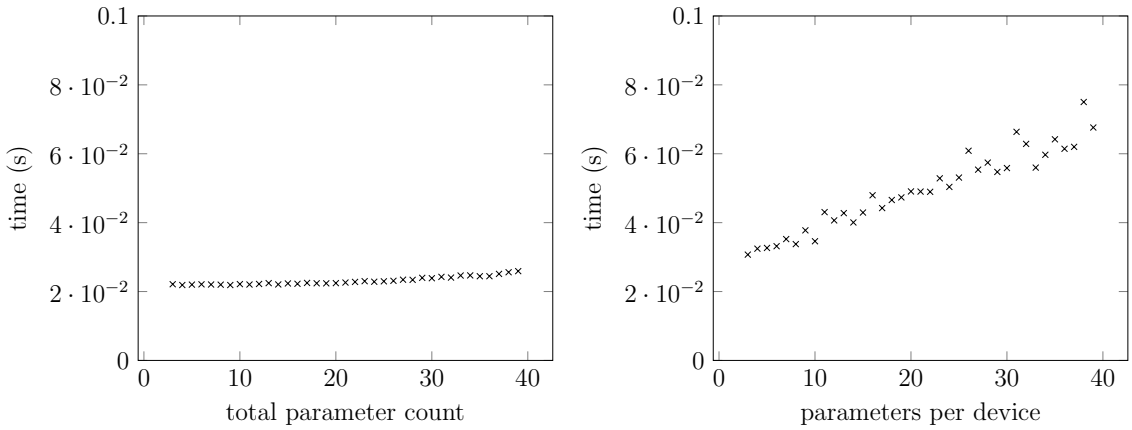
For practical purposes, such performance is more than suitable for dynamically assessing the security of the system as devices leave and join it, even without further optimisation[2]. Further, the implementation could be applied to many enterprise-scale networks with acceptable performance.

9.2 Smart Home Use Case

This implementation was tested against a set of devices synthesised based on the smart home dataset. By demonstrating the implementation, I validate the techniques presented in this thesis and show that the methods presented can be readily applied to realistic examples.

This device set was intended to represent a realistic smart home of the near-future (containing devices that would be available today, but perhaps in greater quantity than is likely today). The smart home being modelled contained 17 classes of device,

[2]Such optimisations could include, for example, maintaining as much of the graph as possible when changes to the system occur. In particular, devices that only join a single logical network would not invalidate any of the previous calculations.



(a) Varying the total number of parameters (2 per device).

(b) Varying the number of parameters per device (50 total parameters).

Figure 9.4: Time taken to find all attack surfaces and the full failure set for a randomly-chosen device, as **the set of parameters is varied**. System contains 6 logical networks, and 1000 devices.

arranged across 5 logical networks. Devices were assigned properties based on their manufacturer (with some devices sharing a manufacturer) and some relevant properties such as shared standards. Note that these are device *classes* as opposed to specific devices. The presence of multiple, identical devices does not affect the analysis, so in a smart home containing many identical lights, for example, these lights can be modelled as a single device class.

Table 9.1 and Table 9.2 show the full data input to the model, including the list of hosts, logical networks, properties and the example probabilities[3]. These probabilities are intended as approximate estimates, and are split so that some manufacturers are considered more likely to cause vulnerabilities than others. Most of the parameters, starting with M, indicate a manufacturer of the device, but one, starting with P, indicates a shared library between multiple devices. In practice, as demonstrated in the implementation section above, more parameters could readily be added to the model without significant performance penalties.

Construction of an attack graph of this scale using the topology-inferred technique is highly efficient taking less than a millisecond on standard desktop hardware. This use case analysis proceeds by examining potential risks to the home, and how these

[3]Derivation of these probabilities is not straightforward to justify. Because of the relatively constrained set of parameters (compared with the set of vulnerabilities in typical analysis), it is more tractable than other probability assignments in existing work. Analysis can equally proceed either without these probabilities, or by assuming they are all equal.

	Type	E	M	A	L	S	Parameters
1	Media	-	✓	✓	-	-	M1
2	Doorbell	-	-	-	-	✓	M3 P1
3	Locks	-	-	-	-	✓	M9 M13
4	Heating	✓	-	✓	-	-	M10 M13
5	Sensors	-	-	✓	✓	-	M11 M13
6	Controls	-	-	-	-	✓	M12 M13
7	Lights	-	-	✓	✓	-	M2 P1
8	Lights	-	-	-	-	✓	M4 P1
9	Locks	-	-	-	-	✓	M5
10	Media	-	✓	✓	-	-	M5
11	Security Hub	✓	-	-	-	✓	M6
12	Camera	✓	-	✓	-	-	M2
13	Camera	-	-	-	-	✓	M7
14	Appliances	✓	-	✓	-	-	M1
15	Solar	✓	-	✓	-	-	M8
16	Automation Hub	✓	✓	✓	-	-	M1 P1
17	Lighting Hub	✓	-	✓	✓	-	M2

Table 9.1: Devices in the fictional smart home. Logical networks are listed as **E**xternal, **M**edia, **A**utomation, **L**ighting and **S**ecurity.

Parameter	Probability
M1	0.01
M2	0.01
M3	0.01
M4	0.01
M5	0.01
M6	0.01
M7	0.2
M8	0.2
M9	0.2
M10	0.2
M11	0.2
M12	0.2
M13	0.3
P1	0.1

Table 9.2: Parameters in the smart home example, with their associated probabilities.

can be mitigated.

9.2.1 Preventing Break-in

The smart home contains two smart locks, which are responsible for controlling access to the building from the outside. If compromised, these would enable the attacker to gain unauthorized entry to the building, with potential serious consequences. As a result, the first part of this analysis seeks to examine the likelihood of such a breach occurring.

We can first construct the full failure sets for both locks, indicating the sets of parameters which are sufficient for an attacker to compromise one or other of the two locks. This process takes approximately 10 milliseconds, and results in a set of 408 possible profiles which can reach the locks.

Of these, one is a single-point-of-failure for one of the locks; i.e., it is a failure set of size 1. This means it is a single parameter which, when present, enables an attack on the lock. In this case, the parameter relates to the manufacturer of electrical components present in a number of the devices. This indicates that a compromise stemming from this manufacturer would have significant consequences for the owner of the smart home, allowing an attacker to gain physical access to their home. In fact, the vulnerability present in these devices which led to them being included in the data set was a vulnerability from this manufacturer; the components had all been left with default hard-coded credentials.

This path, despite representing a single-point-of-failure, consists of multiple edges and traverses multiple networks. The path consisted of compromising a heating controller that had internet access, using this to compromise a security control panel and gain access to the secure logical network, and finally using this access to compromise the lock itself. This was made possible because of the shared dependencies between the devices.

Aside from this single parameter failure, the locks were susceptible to 7 different 2-failure sets, where a pair of parameters enabled their compromise.

Assigning probabilities to the parameter set, we can extend our analysis to look at the risk posed by attacks on the lock. Examining the shortest number of exploits required to reach the lock under each profile, we can plot a cumulative probability function for this metric, as shown in Figure 9.5 [4] . This example examined the metric on all $16,384$ profiles in approximately 3 seconds.

From the diagram, we can see that a significant number of profiles cannot compromise the locks; in fact, 67% of profiles (likelihood-weighted) cannot reach either lock.

[4]The implementation generated the values for all the cumulative probability plots in this section directly. Plotting was performed using PGFPlots http://pgfplots.sourceforge.net/.

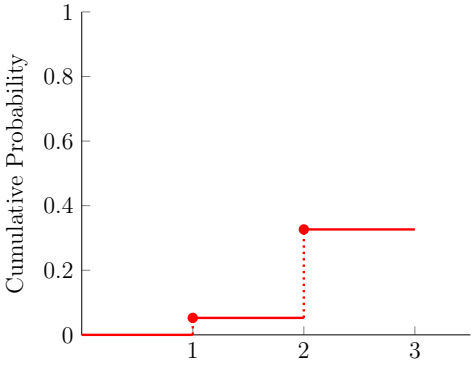


Figure 9.5: The cumulative distribution of the **shortest path to lock compromise** metric.

Very few profiles (5%) can perform the exploit in a single step, and the remaining profiles that can require two.

Given the nature of the single parameter failure set, we might recommend that the user take action to avoid this weakness. For instance, the user could diversify away from that manufacturer, and opt for alternative devices that do not share potential weaknesses with other devices in the system. In some cases, it may be possible to enact a control that prevents this threat, therefore mitigating a particular parameter.

Here we assume the user chooses to opt for a different lock, replacing it with an otherwise-identical model from a different manufacturer. The analysis can then be repeated with the new lock.

With this new lock, the full analysis described above took approximately 2 seconds, and found no single parameter failure sets. We can plot the new cumulative probability function alongside the old one, to examine the differences, as shown in Figure 9.6. The new system has considerably better security; without the shared vulnerability between the lock and the other devices, the ability for an attacker to compromise the lock depends on multiple parameters being present, which is significantly less likely. In this system, the probability of compromise being possible has dropped to 0.4%. This difference can readily be seen on the cumulative probability plot.

For the purpose of this example, we now assume that a public and unpatched vulnerability is discovered in the newly-replaced smart lock. This can be modelled in the system as a parameter with probability 1, so that it can always be performed. Running the analysis again provides us with Figure 9.7. This returns to a similar plot as with the original lock. The lock with the known vulnerability is slightly more likely to be compromised than the original (43% of attackers can compromise it, compared

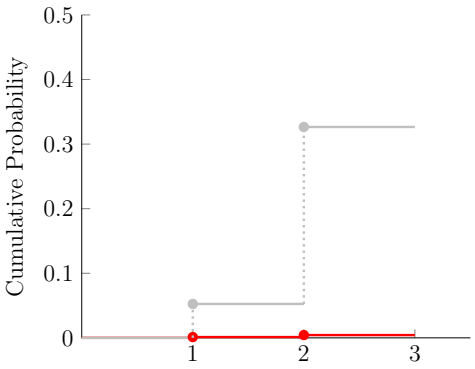


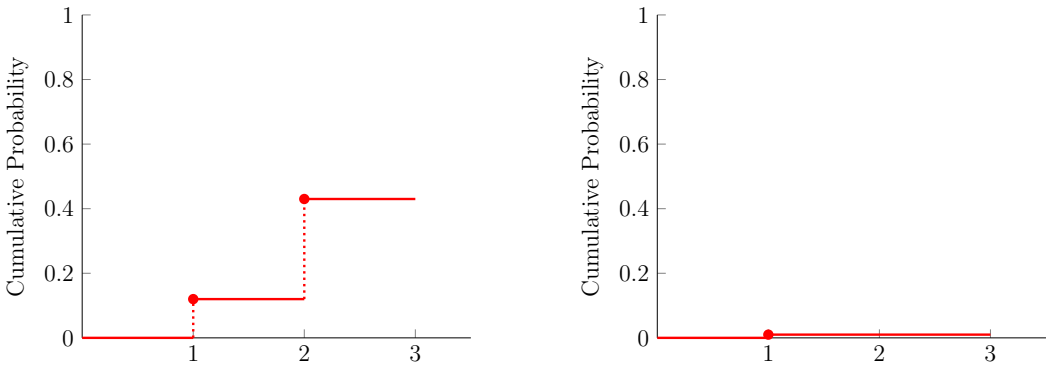
Figure 9.6: The cumulative distribution of the **shortest path to lock compromise** metric, after the lock has been replaced. The previous plot with the other lock is shown in grey.

to 33% who can compromise the original lock). Note that the difference between this lock, which is known to be compromised, and a lock that shares weaknesses with other devices on the network, is not very large – certainly, not as significant as the difference between these and the lock without the known weakness.

This highlights the importance of shared weaknesses in the system. Even with a known-vulnerable device present, it can only be compromised by attackers who can reach it. Hence, unless attackers can also find a path to the device, the known weakness is not important. In light of this, and faced with the alternative prospect of buying (yet another) new smart lock, the smart home user instead chooses to be more stringent with what can access the vulnerable lock, only allowing devices on that network to communicate via their security hub device.

Consequently, as shown in Figure 9.7, the lock becomes very secure, with only 1% of attackers able to compromise it, despite containing a weakness exploitable by every attacker. This is because the security hub acts as a gateway, not only for the devices themselves, but necessarily now for any attack on the lock. The security of the lock is entirely dependent on the security of the hub, because a compromise of the hub naturally leads to a compromise of the lock.

This use case demonstrates how the analysis presented in this thesis can be used to evaluate the security of the network, provide recommendations and provide updated evaluations. Through this, the fictional smart home user was able to discover a weakness in the architecture of their network, and to react to a vulnerability discovered elsewhere. In both cases, the system was able to be updated in order to make the relevant compromise considerably more difficult for attackers to perform.



(a) With usual restrictions on the security network.

(b) Once the security network is strictly restricted.

Figure 9.7: The cumulative distribution of the **shortest path to lock compromise** metric, after a public vulnerability has been disclosed.

9.3 Comparison to other methods

In this section, I provide a comparison between my proposed combined technique and other existing attack graph methods.

The most notable difference between my technique and that of other attack graph methods is the reduced data requirements. By not tying the model directly to vulnerability information, this requirement, present in almost all other surveyed attack graph work (see Section 2.5.1). The problems highlighted by others with this data, particularly with CVSS data (e.g. [AM14]), are therefore avoided. My technique can still benefit from the wealth of information present in databases such as the NVD, but indirectly, and so with less stringent requirements; for instance, through statistical derivation of vulnerability likelihood across time.

This benefit is exemplified by approximately half of the vulnerabilities, present in the Smart Home dataset yet not appearing in the NVD. These have no corresponding CVE or CVSS information, and so would not appear in an analysis based on typical vulnerability scanners (e.g. [AJ18, HK08, FWSJ08, ILP06]).

Avoiding reliance on specific vulnerabilities also circumvents the need for manual vulnerability enumeration (such as in [NPN17]), which is a difficult task on small networks and impractical on large networks.

The most similar work to my approach are existing approaches to modelling zero-day attacks. Existing work in this area also attempts to model vulnerabilities for which no specific definition is provided. *Sun et al.* [SDL+16] propose a method for identifying zero-days based on retrospective observations of attacks. While this gives

an opportunity for zero-day attacks to be assessed, it does so after the attack has happened, in contrast to the predictive benefits of my method (and most other attack graph techniques).

Wang et al. [WJS+14] propose an attack graph method that combines a typical, known-vulnerability attack graph with a second layer consisting of connectivity and running services. This results in a model that has a similar purpose to that proposed here. It is able to capture the possible existence of zero-day exploits through topology information. However, in contrast to my proposed technique, it still depends on vulnerability information to construct the underlying attack graph. It also holds zero-day vulnerabilities to be either equivalent or entirely distinct; as a result, removing the dependency on existing vulnerability information would result in a highly-connected graph of zero-days. Conversely, my proposed technique allows for modelling of the similarity between unknown vulnerabilities and hence more precise and practical modelling of the likelihood that a given set of vulnerabilities exist.

The proposed k-zero-day metric [WJS+14] is a valuable measure of security, and could equally be calculated from my technique. It is roughly analogous to the calculation of the smallest non-empty x-Parameter Failure Set; although this does not take into account which of these are zero-days and which are not.

A key difference between my work and existing probabilistic attack graph methods is the decreased number of probabilistic assignments that must be made. For the worked example above, the model required 14 assignments (see 9.2); this is one probability per parameter. Conversely, typical methods require one assignment per vulnerability (e.g. [WIL+08]), and Bayesian methods require assignments for each vulnerability and for each combination of vulnerabilities or artefacts that vulnerability depends on ([MGSPL17, PDR12, FWSJ08]). Capturing the interdependency between vulnerabilities in my technique is performed through parameters, and so does not require additional assignments; capturing the interdependency between parameters requires assigning probabilities based on combinations of parameters, which is considerably smaller than on combinations of vulnerabilities, particularly in large graphs.

This reduced number of assignments benefits both the feasibility of graph construction and the accuracy of the probabilities. With a significantly-restricted set of assignments, expert knowledge can be used to populate the dataset. Further, the parameters of the model can be fine-tuned and adjusted after construction to explore the effects of varying them, leading to better understanding of their sensitivity and allowing the exploration of "what-if" scenarios.

Considering vulnerabilities through parameters also enables a greater variety of analysis techniques. In addition to the "what-if" explorations mentioned above, parametrisation enables exploration of different attacker profiles and adapts analysis to take into account the attacker profiles. Performing attack graph analysis with consideration of the attacker was a stated goal of the original attack graph work [PS98], yet most attack graph methods do not attempt to model the attacker.

Existing methods of attacker modelling on attack graphs involve limited profiles (e.g. [DKC09]) or generic attacker properties (e.g. [NPN17]). By explicitly considering parameters, both of the attacker and of the scenario, my technique enables analysts to examine the consequences of particular attackers, scenarios, or parameters. Further, combined with probabilistic assignments, analysts can examine risk profiles in varied situations, and across spectra of possibilities through conditional probability graphs. By tailoring the analysis to different scenarios, my technique enables analysts to better understand the risk profile of their system, both in general and in regard to specific weaknesses. This stands in contrast to techniques that model a single, implicit attacker (see Section 2.5.3) which can only give a generic, often worst-case, prediction.

Chapter 10

Conclusion

The aim of this thesis was to examine and address problems faced by attack graphs, and to provide novel techniques for generating and analysing those graphs. The key problems identified in existing attack graph work were data quality, computational performance and difficulties with graph analysis, and theoretical solutions have been proposed with the aim of addressing each of these.

To validate these contributions, the combined technique was implemented and tested against a smart home use case, demonstrating the practicality of the method, in terms of the data required to use the method, the computational performance and in terms of its ability to provide actionable results.

10.1 Contributions

The primary novel contributions of the thesis are as follows:

- A method, presented in Chapter 4, to compare certain pairs of attack graphs in an objective fashion, under given assumptions. This is used as a means by which to evaluate subjective metrics, find counterexamples and propose improvements.
- The explicit assumptions of single-preconditions and partitioned-preconditions, as presented in Chapter 5. While both have been implicitly part of attack graph work to various degrees, making them explicitly and precisely enables them to be consciously considered and understood.
- A method, presented in Chapter 6, to build attack graphs directly from topology information, avoiding reliance on vulnerability information.

- A method, presented in Chapter 7, to parametrise attack graphs using a parameter set, enabling easier and more detailed modelling of both attackers and defenders.

This thesis also makes the following further contributions:

- I present a formalisation of the template-matching attack graph generation technique (Section 3.1.1).
- The smart home dataset (Section 3.2.1) contains a set of vulnerabilities selected that represent real-world vulnerabilities in a set of smart home devices. This was used throughout the thesis as a motivating example.
- Chapter 8 presents the combination of the previous chapters together with a collection of analysis methods, which are applied to the smart home example but equally can be applied to other systems.
- Chapter 9 provides a demonstration of the combined technique on a smart home use case, together with details for my implementation of the technique in Python, validating both the previous work in this thesis.

10.2 Solutions to Stated Challenges

At the beginning of this thesis, I stated three primary challenges faced by attack graph methods. These were:

Data. The data sources used in much of attack graph work often appear unsuitable and incomplete. Techniques that use this data are therefore unreliable.

Performance. Attack graph techniques have become significantly more efficient over time, but it is not clear or well-established how these performance improvements are achieved, or what the full consequences of these improvements are.

Analysis. Many attack graph analysis methods have been proposed, but few have clear justifications, and many lead to questionable security assessments in certain cases. Analysis often requires manual interpretation, making it slow and subjective.

I now examine each of these challenges with respect to the contributions of the thesis.

10.2.1 Data

Attack graphs are constructed from vulnerability data, which is gathered from data sources such as the National Vulnerability Database [NVD]. These vulnerabilities are used to construct the structure of the graph and, in some methods, to augment the graph further with probability information (such as from the Common Vulnerability Scoring System [CVS]).

These data sources are frequently incomplete, and only contain vulnerabilities that have been discovered and disclosed. This incompleteness is particularly in the case of Internet of Things devices[1]. Probability assignments, where an exploit is given a likelihood of success, must be made algorithmically from only partially-related data sources due to the high number of vulnerabilities, and such assignments are difficult to explain or justify.

In Chapter 6 I presented **Topology-Inferred Graphs**, a method for building graphs without using vulnerability information. This method, either alone or as part of the combined technique, enables attack graphs to be constructed without depending on known vulnerabilities. Instead, these graphs rely on simpler, easier-to-acquire information to give an architecture-based view of the security of the system, instead of a single snapshot of the security with respect to the vulnerabilities that are known at the time. This is especially beneficial in situations where little is known about devices in the system, such as the smart home example.

In Chapter 7 I presented **Parametrised Attack Graphs**, which allow probabilistic information to be incorporated into the graph through *parameters*. This set of parameters is considerably smaller than the set of vulnerabilities, and so can readily be considered directly by practitioners, avoiding the necessity of algorithmic probability assignments. This makes them easier to justify theoretically, especially in light of the difficulty of achieving empirical validation. This reduction in necessary assignments is especially significant when contrasted with other methods that attempt to capture the dependencies between exploit probabilities, such as Bayesian Attack Graphs.

The combined technique presented in Chapter 8 therefore presents a means for practitioners to build models of their systems that are based on the network topology and minimal host information – two requirements that are needed by other attack graph methods in order to find vulnerability information. The resulting graph can be

[1]See, for example, *Chothia and de Ruiter* [CdR16], where student teams were able to find serious unreported vulnerabilities in multiple devices.

augmented with probabilistic information through parametrisation, giving the benefits of typical risk-based graph methods but requiring only a small fraction of the probability assignments.

These graphs are not reliant on the discovery of vulnerabilities, making them highly-suitable for situations such as the smart home, in which there is rapid device turnover of devices from a wide variety of potentially untrustworthy sources. In other environments, such as industrial Internet of Things deployments and traditional enterprise networks, proprietary hardware and software may never have publicly available vulnerability reporting, and so will always be considered perfectly secure by traditional attack graph methods. In the combined technique, these devices are considered as vulnerable as any others, making this technique applicable to these situations.

10.2.2 Performance

Attack graph performance has progressed greatly from the early techniques, which were impractical on any more than a handful of hosts. Dependency attack graphs solve this problem by returning to attack-tree-like operator-based edges, while transition attack graphs rely on restricting the edge or vertex sets, often basing each vertex on a single host.

Both methods attribute much of their performance benefits to the assumption of monotonicity, which is necessary for dependency attack graphs. However, it is not entirely clear that dependency or transition graphs can gain their demonstrated performance improvements purely from the assumption of monotonicity.

Explicitly stating a definition of monotonicity enabled future research to be precise about the assumptions being made in their work, and enabled others to consider specifically if the assumption was suitable for them[2].

In Chapter 5 I proposed two further assumptions, the **single-precondition assumption** and the **partitioned-precondition assumption**. The first of these is being made implicitly in much attack graph work, particularly that of host-centric models, and their performance improvements can be attributed to it. Making this assumption explicitly enables future work to consider its validity and make a conscious decision of whether it is true in their particular use case.

The single-precondition assumption greatly reduces the possible state-space of the graph, reducing it from $O(2^n)$ to $O(n)$. This has clear benefits for the performance

[2]For example, the work of *Zhuang et al.* [ZZD+12] which deliberately does not make this assumption, in order to enable the modelling of dynamic defences.

of graph models, which suffer twice from large numbers of states; firstly during construction and secondly during analysis. The partitioned-precondition assumption also reduces the number of possible states depending on p the size of the partitions chosen, to $O(\frac{2^p}{p}n)$, resulting in (rough) equivalence to the single-precondition assumption for singleton partitions and reversion to $O(2^n)$ when the entire state space is partitioned into a single partition.

The **Topology-Inferred Graphs** in Chapter 6 use this assumption to provide a method of constructing the graph directly from the topology, which can be done very efficiently from topology data. This circumvents the potentially-expensive template-matching methods that vulnerability-based attack graphs require. This also enables further performance improvements during analysis which exploit the structure of the resulting graph.

10.2.3 Analysis

Attack graphs suffer greatly from the difficulty of providing empirical valuation in security. Models of attackers, particularly probabilistic models, require infeasible amounts of data to verify. Consequently, analysis is often poorly-grounded and based only on the best estimate of the analyst or researcher, with no evident means by which to justify it.

To address this, in Chapter 4 provides **Relative Attack-Graph Security** comparisons, which are theoretically-grounded comparisons between the security of attack graphs based on clearly-stated assumptions. These comparisons can therefore be said to be objectively true, given the assumptions. Because of the strict requirements and avoidance of any subjective comparison, these comparisons can be made without requiring any evaluation of individual exploits, which is considered out of scope.

While these comparisons only apply to very specific pairs of attack graphs, they can be used as a benchmark for other, subjective metrics. Because the objective comparison is considered to be true given the assumptions, any reliable metric should agree with the assessment of the objective comparison on any pair of graphs for which it is defined. By applying this technique, I found metrics which did not agree with the objective comparison, and was able to construct examples which exemplify this discrepancy and demonstrate realistic examples on which the subjective metrics give unintended results.

By providing this theoretical assessment of subjective metrics, researchers developing metrics have a means to evaluate them without needing difficult-to-find empirical

data, and subsequently avoid problems that may otherwise be unknown. The techniques presented in this chapter can also be applied to other sets of assumptions, which may be application-specific, so that metrics intended for specific use cases can be evaluated under suitable assumptions.

Chapter 7 introduces **Parametrised Attack Graphs**, which enable analysis that incorporates both defender choices and attacker properties. Network defenders can examine the consequences not only of individual choices, but of the combinations of those choices. This can be done using any existing attack graph metrics, so that each decision can be made by drawing on the considerable existing work in this area. Parametrised graphs also model the attacker probabilistically, again examining every possible combination of attacker capabilities instead of relying on preconceived profiles as in other methods.

By constructing a set of attack graphs from the parameter set, the structure of the parameter set can be used to develop novel attack graph analysis methods, such as using cumulative distribution plots to examine the variance of a metric across possible attackers. Security assessment can also be expressed in terms of parameters, enabling conclusions to be drawn regarding the parameters themselves, such as finding the failure sets of parameters that, alone, are sufficient to enable compromise, and measuring the attack surface of the system precisely in terms of the parameters which enable attackers to gain entry to the network.

10.3 Limitations

The scope of this thesis is specifically concerned with theoretical validation and modelling. This choice was made, in part, due to the difficulty of acquiring data with which to perform empirical validation. Setting aside the confidential nature of much of the data required to construct and use attack graphs, the amount of data required to perform a statistically-significant validation is prohibitively large, particularly when validating probabilistic claims.

A particular system might be the victim of a sophisticated, targeted attack only very rarely, and so even a long-term study on a system might detect only a small handful of attacks, if any. Performing validation with such a small set of attacks would be unreliable, and probably little better than no validation at all. Particularly, it is especially difficult to claim that a system is more secure; even if it suffers no attacks at all it is impossible to say whether this was as a result of improvements made or simply by chance.

This further supports the focus of this thesis on making clear assumptions that can be tested and validated across multiple systems and by multiple groups. From these assumptions, theoretically-valid conclusions can be drawn that can be trusted as much as the assumptions that underlie them.

10.4 Future Work

The work presented in this thesis offers a number of potential avenues for future work. Specifically:

- The work of Chapter 4 can be extended and applied to specific use cases. Environments that have particular requirements can be considered with distinct sets of assumptions, to provide metric evaluations that apply to that case. Additionally, this chapter specifically avoided comparisons between exploits in the graph, because of the inherent difficulty and subjectivity of doing so. It may be possible to provide some situations in which particular pairs of exploits can be compared, similar to how attack graphs are compared currently.

- Chapter 5 enables researchers to consider the single-precondition assumption explicitly, and it would be beneficial for future work to examine the consequences of making this assumption specifically. In cases where this assumption does not hold, it may be useful to apply the partitioned-precondition assumption instead. Aside from this, future work in attack graph would benefit from clearer, more well-defined assumptions, and the work in this chapter can be viewed as a step towards more strongly theoretically-grounded attack graph work.

- Chapter 6 represents a shift from considered attack graphs as maps of known vulnerability to maps of architectural weakness. Both these approaches have merit; existing methods are directly applicable to the task of directing patches with maximal impact, and topology-inferred graphs avoid the strict data requirements and handle zero-day vulnerabilities. Alongside further work developing attack graph models of architectural weakness, future work could benefit from combining these two models into a single method, that both captures potential and known weakness.

- Chapter 7 provides a new area of attack graph analysis based on parameters. Because of the significant reductions in the number of probability assignments required, placing more focus on the accuracy and validity of these assignments

will result in better returns that similar work on other probabilistic attack graph models. Consequently, future work could gather data from multiple sources and attempt to synthesise general parameters and probabilities of attackers, which could be reused by different applications of the model.

10.5 Final Remarks

The work in this thesis is intended as a step towards stronger theoretical grounding of attack graph analysis. While the benefits of accurate and reliable attack graph modelling seem clear, their practical difficulties make them hard to justify and apply.

Faced with a lack of empirical data, and no clear methods by which such data could be acquired, it becomes necessary for us to rely on purely theoretical work with clear assumptions. By making assumptions clearly and proceeding from those assumptions with provable techniques, we can separate out and make explicit the subjectivity of attack graph work.

Once this solid grounding is established, it can be expanded upon more reliably and more valuably, so that attack graphs can be presented as a practical technique for better understanding and evaluating our systems' ability to withstand and prevent attack.

Returning to the context of cyber security presented at the beginning of this thesis, the techniques presented here aim to move attack graphs closer to viability as a means by which network defenders can understand the landscape of vulnerability present on their system.

By mitigating the existing problems regarding the quality of data, the computational performance, and the analytical power of graphs through a variety of approaches, I hope that my contributions enable practitioners to design networks that are resilient to unknown threats, and accurately estimate the impact of vulnerabilities without dependence on unobtainable data.

Appendix A

Glossary and List of Terms

$\mathcal{P}$ – Properties – The set of properties of the system. Each is an atomic fact about the system being modelled.

$\mathcal{S}$ – States – The set of states the system being modelled can be in. Each state is a collection of properties, so that the set of states $\mathcal{S}$ is the power set of the property set $\mathcal{P}$.

$\mathcal{T}$ – Templates – The set of templates. Each template is a map from a subset of the set of spaces to the set of states, and each is associated with an exploit

$\mathcal{E}$ – Exploits – The set of all exploits, which relate to templates.

S – Attack Sequences – The set of all attack sequences, each of which is an ordered, finite sequence of exploits. These are represent steps of exploits that can be made during an attack, although it is possible that an attack sequence contains exploits that cannot be performed in that order, and are thus impossible attack sequences.

S' – Attack Suites – The set of all attack suites, each of which is a set of attack sequences, representing multiple, distinct methods by which a particular target can be reached.

`SHV-`N – Smart Home Vulnerability N – Refers to the Nth vulnerability from the smart home vulnerability dataset, as described in Table 3.1.

`CVE-Y-N` – CVE ID – The standardised Common Vulnerability and Exposures ID of a vulnerability. These can be used to find vulnerabilities in relevant database, e.g. [NVD]

Ψ – Parameters – The set of parameters for a parametrised attack graph. Each parameter represents a fact about the system or attacker that does not change over the course of the attack.

Ψ_c – Controlled Parameters – The set of *controlled* parameters, which represent those parameters which can be selected as a consequence of choices by the defenders, and hence are known certainly. Such parameters might include the deployment of particular software, or whether a given policy is enacted or not.

Ψ_u – Uncontrolled Parameters – The set of *uncontrolled* parameters, which represent the parameters which are not selected by the defenders, and therefore represent aspects of the scenario that are not (necessarily) known. Such parameters might include the capabilities of the attacker, or precise conditions of the network.

Bibliography

[AA13] Jaafar Almasizadeh and Mohammad Abdollahi Azgomi. A stochastic model of attack process for the evaluation of security metrics. *Computer Networks*, 57(10):2159–2180, July 2013.

[ABD+18] M. Ugur Aksu, Kemal Bicakci, M. Hadi Dilek, A. Murat Ozbayoglu, and E. slam Tatli. Automated Generation of Attack Graphs Using NVD. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, pages 135–142, New York, NY, USA, 2018. ACM.

[Abr16] S. M. Abraham. Estimating Mean Time to Compromise Using Non-homogenous Continuous-Time Markov Models. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 467–472, June 2016.

[AEIE13] Hande Alemdar, Halil Ertan, Ozlem Durmaz Incel, and Cem Ersoy. ARAS Human Activity Datasets in Multiple Homes with Multiple Residents. page 4, 2013.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010.

[AJ18] Massimiliano Albanese and Sushil Jajodia. A Graphical Model to Assess the Impact of Multi-Step Attacks. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 15(1):79–93, January 2018.

[ALC+17] Ioannis Agadakos, Ulf Lindqvist, Chien-Ying Chen, Matteo Campanelli, Prashant Anantharaman, Monowar Hasan, Bogdan Copos, Tancrde Lepoint, Michael Locasto, and Gabriela F. Ciocarlie. Jumping the Air

Gap: Modeling Cyber-Physical Attack Paths in the Internet-of-Things. pages 37–48. ACM Press, 2017.

[AM14] Luca Allodi and Fabio Massacci. Comparing Vulnerability Severity and Exploits Using Case-Control Studies. *ACM Transactions on Information and System Security*, 17(1):1–20, August 2014.

[APRS05] Paul Ammann, Joseph Pamula, Ronald Ritchey, and J. des Street. A host-based approach to network attack chaining analysis. In *21st Annual Computer Security Applications Conference (ACSAC'05)*, pages 10–pp. IEEE, 2005.

[Arc03] I. Arce. The weakest link revisited. *IEEE Security & Privacy Magazine*, 1(2):72–76, March 2003.

[AWK02] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224. ACM, 2002.

[Bay13] Jennifer L. Bayuk. Security as a theoretical attribute construct. *Computers & Security*, 37:155–175, September 2013.

[BD12] Leyla Bilge and Tudor Dumitras. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, page 833, Raleigh, North Carolina, USA, 2012. ACM Press.

[BG12] P. Bhattacharya and S. K. Ghosh. Analytical framework for measuring network security using exploit dependency graph. *IET Information Security*, 6(4):264–270, December 2012.

[BODBW13] Nazmiye Balta-Ozkan, Rosemary Davidson, Martha Bicket, and Lorraine Whitmarsh. Social barriers to the adoption of smart homes. *Energy Policy*, 63:363–374, December 2013.

[BWJS16] Daniel Borbor, Lingyu Wang, Sushil Jajodia, and Anoop Singhal. Diversifying Network Services Under Cost Constraints for Better Resilience Against Unknown Attacks. In Silvio Ranise and Vipin Swarup, editors, *Data and Applications Security and Privacy XXX*, volume 9766, pages 295–312. Springer International Publishing, Cham, 2016.

[CCZ08] Hasan Cavusoglu, Huseyin Cavusoglu, and Jun Zhang. Security Patch Management: Share the Burden or Share the Damage? *Management Science*, 54(4):657–670, April 2008.

[CdR16] Tom Chothia and Joeri de Ruiter. Learning From Others Mistakes: Penetration Testing IoT Devices in the Classroom. page 8, 2016.

[Cha14] Alex Chapman. Hacking into Internet Connected Light Bulbs, 2014.

[CKY18] Dove Chiu, Lu Kenney, and Tim Yeh. Device Vulnerabilities in the Connected Home: Uncovering Remote Code Execution and More - Trend-Labs Security Intelligence Blog, April 2018.

[CSHH16] Kyle Cook, Thomas Shaw, Peter Hawrylak, and John Hale. Scalable Attack Graph Generation. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, CISRC '16, pages 21:1–21:4, New York, NY, USA, 2016. ACM.

[CVS] CVSS. CVSS - Common Vulnerability Scoring System - https://www.first.org/cvss.

[CWE] CWE. CWE - Common Weakness Enumeration - https://cwe.mitre.org/.

[CWG+14] Min Chen, Jiafu Wan, Sergio Gonzalez, Xiaofei Liao, and Victor C.M. Leung. A Survey of Recent Developments in Home M2m Networks. *IEEE Communications Surveys & Tutorials*, 16(1):98–114, 2014.

[Dar] Darktrace. Darktrace - https://www.darktrace.com/.

[Dav09] M. A. Davidson. The Good, the Bad, And the Ugly: Stepping on the Security Scale. In *2009 Annual Computer Security Applications Conference*, pages 187–195, December 2009.

[DD94] Marc Dacier and Yves Deswarte. Privilege graph: An extension to the typed access matrix model. In Dieter Gollmann, editor, *Computer Security ESORICS 94*, Lecture Notes in Computer Science, pages 319–334. Springer Berlin Heidelberg, 1994.

[DDK96] Marc Dacier, Yves Deswarte, and Mohamed Kaniche. *Quantitative Assessment of Operational Security: Models and Tools*. 1996.

[DKC09] Ram Dantu, Prakash Kolan, and Joao Cangussu. Network risk management using attacker profiling. *Security and Communication Networks*, 2(1):83–96, 2009.

[DLK+14] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, and Michael Bailey. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488. ACM, 2014.

[Dol83] Danny Dolev. On the Security of Public Key Protocols. *IEEE TRANSACTIONS ON INFORMATION THEORY*, (2):11, 1983.

[DPRW07] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 204–213. ACM, 2007.

[FJP16] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security Analysis of Emerging Smart Home Applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 636–654, San Jose, CA, May 2016. IEEE.

[Fri11] Marcel Frigault. *Measuring network security using Bayesian Network-based attack graphs*. PhD thesis, Library and Archives Canada = Bibliothque et Archives Canada, Ottawa, 2011.

[FW08] M. Frigault and L. Wang. Measuring Network Security Using Bayesian Network-Based Attack Graphs. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 698–703, July 2008.

[FWSJ08] Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring Network Security Using Dynamic Bayesian Network. In *Proceedings of the 4th ACM Workshop on Quality of Protection*, QoP '08, pages 23–30, New York, NY, USA, 2008. ACM.

[GHGK17] Mengmeng Ge, Jin B. Hong, Walter Guttmann, and Dong Seong Kim. A framework for automating security analysis of the internet of things. *Journal of Network and Computer Applications*, 83:12–27, April 2017.

[Her17] Cormac Herley. Justifying Security Measures a Position Paper. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security ESORICS 2017*, volume 10492, pages 11–17. Springer International Publishing, Cham, 2017.

[HK08] R. Hewett and P. Kijsanayothin. Host-Centric Model Checking for Network Vulnerability Analysis. In *2008 Annual Computer Security Applications Conference (ACSAC)*, pages 225–234, December 2008.

[HK12] Jin Hong and Dong-Seong Kim. Harms: Hierarchical attack representation models for network security analysis. 2012.

[HKCH17] Jin B. Hong, Dong Seong Kim, Chun-Jen Chung, and Dijiang Huang. A survey on the usability and practical applications of Graphical Security Models. *Computer Science Review*, 26:1–16, November 2017.

[HOS09] John Homer, Xinming Ou, and David Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. *Kansas State University Technical Report*, pages 1–15, 2009.

[HvO17] Cormac Herley and P. C. van Oorschot. SoK: Science, Security and the Elusive Goal of Security as a Scientific Pursuit. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 99–120, San Jose, CA, USA, May 2017. IEEE.

[HWMK17] Allen D. Householder, Garret Wassermann, Art Manion, and Chris King. The CERT Guide to Coordinated Vulnerability Disclosure. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA PITTSBURGH United States, 2017.

[HZO+13] John Homer, Su Zhang, Xinming Ou, David Schmidt, Yanhui Du, S. Raj Rajagopalan, and Anoop Singhal. Aggregating vulnerability metrics in enterprise networks using attack graphs. *Journal of Computer Security*, 21(4):561–597, 2013.

[IB12] Nwokedi Idika and Bharat Bhargava. Extending Attack Graph-Based Security Metrics and Aggregating Their Application. *IEEE Transactions on Dependable and Secure Computing*, 9(1):75–85, January 2012.

[ILP06] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical Attack Graph Generation for Network Defense. In *ACSAC*, volume 6, pages 121–130, 2006.

[IPHK16] Marieta Georgieva Ivanova, Christian W. Probst, Ren Rydhof Hansen, and Florian Kammller. Transforming Graphical System Models to Graphical Attack Models. In Sjouke Mauw, Barbara Kordy, and Sushil Jajodia, editors, *Graphical Models for Security*, volume 9390, pages 82–96. Springer International Publishing, Cham, 2016.

[Jan09] Wayne Jansen. Directions in security metrics research. Technical Report NIST IR 7564, National Institute of Standards and Technology, Gaithersburg, MD, 2009.

[JN10] Sushil Jajodia and Steven Noel. Topological vulnerability analysis. In *Cyber Situational Awareness*, pages 139–154. Springer, 2010.

[JNO05] Sushil Jajodia, Steven Noel, and Brian OBerry. Topological analysis of network attack vulnerability. In *Managing Cyber Threats*, pages 247–266. Springer, 2005.

[JSW02] Somesh Jha, Oleg Sheyner, and Jeannette Wing. Two formal analyses of attack graphs. In *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pages 49–63. IEEE, 2002.

[JvRNG16] A Janse van Rensburg, JRC Nurse, and M Goldsmith. Attacker-parametrised attack graphs. ThinkMind Digital Library, 2016.

[Kay16] Kerem Kaynar. A taxonomy for attack graph generation and usage in network security. *Journal of Information Security and Applications*, 29:27–56, August 2016.

[KPCS14] Barbara Kordy, Ludovic Pitre-Cambacds, and Patrick Schweitzer. DAG-based attack and defense modeling: Dont miss the forest for the attack trees. *Computer Science Review*, 13-14:1–38, November 2014.

[LIS+05] R. P. Lippmann, K. W. Ingols, C. Scott, K. Piwowarski, K. J. Kratkiewicz, M. Artz, and R. K. Cunningham. Evaluating and Strengthening Enterprise Network Security Using Attack Graphs. Technical report, 2005.

[LIS+06] Richard Lippmann, Kyle Ingols, Chris Scott, Keith Piwowarski, Kendra Kratkiewicz, Mike Artz, and Robert Cunningham. Validating and Restoring Defense in Depth Using Attack Graphs. pages 1–10. IEEE, October 2006.

[LM] John Lyle and Andrew Martin. Trusted Computing and Provenance: Better Together. page 10.

[LS14] Zach Lanier and Mark Stanislav. The Internet of Fails, 2014.

[McD01] James P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 workshop on New security paradigms*, pages 15–21. ACM, 2001.

[Met] Metasploit. Metasploit - https://www.metasploit.com/.

[MGPVT04] Bharat B. Madan, Katerina Goeva-Popstojanova, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. A method for modeling and quantifying the security attributes of intrusion tolerant systems. *Performance Evaluation*, 56(1):167–186, 2004.

[MGSPL17] Luis Muoz-Gonzalez, Daniele Sgandurra, Andrea Paudice, and Emil C. Lupu. Efficient Attack Graph Analysis through Approximate Inference. *ACM Transactions on Privacy and Security*, 20(3):1–30, July 2017.

[MMBC09] Miles A. McQueen, Trevor A. McQueen, Wayne F. Boyer, and May R. Chaffin. Empirical estimates and observations of 0day vulnerabilities. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–12. IEEE, 2009.

[MMW14] Patrick Morrison, David Moye, and Laurie Williams. Mapping the Field of Software Security Metrics. page 24, 2014.

[MO06] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *Information Security and Cryptology-ICISC 2005*, pages 186–198. Springer, 2006.

[Nar18] Tom Nardi. Fix Your Insecure Amazon Fire TV Stick, April 2018.

[NCR17] J. R. C. Nurse, S. Creese, and D. De Roure. Security Risk Assessment in Internet of Things Systems. *IT Professional*, 19(5):20–26, 2017.

[Nes15] Nessus. Nessus - https://www.tenable.com/products/nessus/nessus-professional, February 2015.

[NJ04] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 109–118. ACM, 2004.

[NJOJ03] S. Noel, Sushil Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 86–95, December 2003.

[nma] nmap. nmap - https://nmap.org/.

[NPN17] Hoang Hai Nguyen, Kartik Palani, and David M. Nicol. An Approach to Incorporating Uncertainty in Network Security Analysis. In *Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp*, pages 74–84. ACM, 2017.

[NVD] NVD. NVD - National Vulnerability Database - https://nvd.nist.gov/.

[OA17] Hamed Orojloo and Mohammad Abdollahi Azgomi. A game-theoretic approach to model and quantify the security of cyber-physical systems. *Computers in Industry*, 88:44–57, June 2017.

[OBM06] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345. ACM, 2006.

[ODK99] Rodolphe Ortalo, Yves Deswarte, and Mohamed Kaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *Software Engineering, IEEE Transactions on*, 25(5):633–650, 1999.

[OK14] Yossef Oren and Angelos D. Keromytis. Attacking the Internet Using Broadcast Digital Television. *ACM Transactions on Information and System Security*, 17(4):1–27, 2014.

[Pau14] Stphane Paul. Towards Automating the Construction & Maintenance of Attack Trees: a Feasibility Study. *Electronic Proceedings in Theoretical Computer Science*, 148:31–46, April 2014. arXiv: 1404.1986.

[Pay06] Shirley C. Payne. A Guide to Security Metrics. page 11, 2006.

[PDR12] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic Security Risk Management Using Bayesian Attack Graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1):61–74, January 2012.

[PGLCX16] Marcus Pendleton, Richard Garcia-Lebron, Jin-Hee Cho, and Shouhuai Xu. A Survey on Systems Security Metrics. *ACM Computing Surveys*, 49(4):1–35, December 2016.

[PJAS06] Joseph Pamula, Sushil Jajodia, Paul Ammann, and Vipin Swarup. A weakest-adversary security metric for network configuration security analysis. In *Proceedings of the 2nd ACM workshop on Quality of protection*, pages 31–38. ACM, 2006.

[Pro] Niels Provos. Honeyd: A Virtual Honeypot Daemon (Extended Abstract). page 8.

[PS98] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

[RDRN+18] Petar Radanliev, David Charles De Roure, Razvan Nicolescu, Michael Huth, Rafael Mantilla Montalvo, Stacy Cannady, and Peter Burnap. Future developments in cyber risk assessment for the internet of things. *Computers in Industry*, 102:14–22, November 2018.

[RLFR17] A. Ramos, M. Lazar, R. H. Filho, and J. J. P. C. Rodrigues. Model-Based Quantitative Network Security Metrics: A Survey. *IEEE Communications Surveys Tutorials*, 19(4):2704–2734, 2017.

[RP05] Indrajit Ray and Nayot Poolsapassit. Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. In Sabrina de Capitani di Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Computer Security ESORICS 2005*, Lecture Notes in Computer Science, pages 231–246. Springer Berlin Heidelberg, 2005.

[RS12a] M. Rudolph and R. Schwarz. A Critical Survey of Security Indicator Approaches. In *2012 Seventh International Conference on Availability, Reliability and Security*, pages 291–300, August 2012.

[RS12b] Manuel Rudolph and Dr Reinhard Schwarz. Security Indicators A State of the Art Survey Public Report. page 69, 2012.

[RSD07] E. Ramirez-Silva and M. Dacier. Empirical Study of the Impact of Metasploit-Related Attacks in 4 Years of Attack Traces. In Iliano Cervesato, editor, *Advances in Computer Science ASIAN 2007. Computer and Network Security*, Lecture Notes in Computer Science, pages 198–211. Springer Berlin Heidelberg, 2007.

[San14] W. H. Sanders. Quantitative Security Metrics: Unattainable Holy Grail or a Vital Breakthrough within Our Reach? *IEEE Security Privacy*, 12(2):67–69, March 2014.

[SAT17] Georgios Spanos, Lefteris Angelis, and Dimitrios Toloudis. Assessment of Vulnerability Severity using Text Mining. In *Proceedings of the 21st Pan-Hellenic Conference on Informatics - PCI 2017*, pages 1–6, Larissa, Greece, 2017. ACM Press.

[Sch99] Bruce Schneier. Attack trees. *Dr. Dobbs journal*, 24(12):21–29, 1999.

[SDL+16] Xiaoyan Sun, Jun Dai, Peng Liu, Anoop Singhal, and John Yen. Towards probabilistic identification of zero-day attack paths. In *Communications and Network Security (CNS), 2016 IEEE Conference on*, pages 64–72. IEEE, 2016.

[Sha16] Scott J. Shackelford. Protecting intellectual property and privacy in the digital age: the use of national cybersecurity strategies to mitigate cyber risk. *Chap. L. Rev.*, 19:445, 2016.

[SHJ+02] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.

[Sno] Snort. Snort - https://www.snort.org/.

[SPEC01] Laura P. Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 2, pages 307–321. IEEE, 2001.

[SS15] Teodor Sommestad and Fredrik Sandstrm. An empirical test of the accuracy of an attack graph analysis tool. *Information and Computer Security*, 23(5):516–531, November 2015.

[SW04] Oleg Sheyner and Jeannette Wing. Tools for generating and analyzing attack graphs. In *Formal methods for components and objects*, pages 344–371. Springer, 2004.

[Sym19] Symantec. Internet Security Threat Report 2019. page 5, 2019.

[TCC17] Guilherme Mussi Toschi, Leonardo Barreto Campos, and Carlos Eduardo Cugnasca. Home automation networks: A survey. *Computer Standards & Interfaces*, 50:42–54, February 2017.

[TL01] Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 workshop on New security paradigms*, pages 31–38. ACM, 2001.

[TO18] Lawrence J Trautman and Peter C Ormerod. Industrial Cyber Vulnerabilities: Lessons from Stuxnet and the Internet of Things. *U. Miami L. Rev.*, 72:67, 2018.

[Ven15] Pedro Venda. Hacking DefCon 23's IoT Village Samsung fridge | Pen Test Partners, 2015.

[Ver09] Vilhelm Verendel. Quantified security is a weak hypothesis: a critical survey of results and assumptions. In *Proc. 2009 workshop on New security paradigms workshop, Sept.08-11, 2009*, pages 37–49, 2009.

[WIL+08] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *Data and applications security XXII*, pages 283–296. Springer, 2008.

[WJS+14] L. Wang, S. Jajodia, A. Singhal, P. Cheng, and S. Noel. k-Zero Day Safety: A Network Security Metric for Measuring the Risk of Unknown Vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, 11(1):30–44, January 2014.

[WLJ06] Lingyu Wang, Anyi Liu, and Sushil Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15):2917–2933, September 2006.

[WMS+17] Ryan Williams, Emma McMahon, Sagar Samtani, Mark Patton, and Hsinchun Chen. Identifying vulnerabilities of consumer Internet of Things (IoT) devices: A scalable approach. In *Intelligence and Security Informatics (ISI), 2017 IEEE International Conference on*, pages 179–181. IEEE, 2017.

[Wol16] Josephine Wolff. Perverse Effects in Defense of Computer Systems: When More Is Less. *Journal of Management Information Systems*, 33(2):597–620, April 2016.

[WSJ07a] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring the overall security of network configurations using attack graphs. In *Data and Applications Security XXI*, pages 98–112. Springer, 2007.

[WSJ07b] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Toward measuring network security using attack graphs. In *Proceedings of the 2007 ACM workshop on Quality of protection*, pages 49–54. ACM, 2007.

[WTFB08] Marc Weber Tobias, Matt Fiddler, and Toby Bluzmanis. Backdooring the Frontdoor, 2008.

[XCT+09] A. Xie, Z. Cai, C. Tang, J. Hu, and Z. Chen. Evaluating Network Security With Two-Layer Attack Graphs. In *2009 Annual Computer Security Applications Conference*, pages 127–136, December 2009.

[XLO+10] Peng Xie, Jason H. Li, Xinming Ou, Peng Liu, and Renato Levy. Using Bayesian networks for cyber security analysis. In *Dependable Systems*

and Networks (DSN), 2010 IEEE/IFIP International Conference on, pages 211–220. IEEE, 2010.

[Yag06] Ronald R. Yager. OWA trees and their role in security modeling using attack trees. *Information Sciences*, 176(20):2933–2959, October 2006.

[YMR16] Awad Younis, Yashwant K. Malaiya, and Indrajit Ray. Assessing vulnerability exploitability risk using software properties. *Software Quality Journal*, 24(1):159–202, March 2016.

[ZCO11] Su Zhang, Doina Caragea, and Xinming Ou. An empirical study on using the national vulnerability database to predict software vulnerabilities. In *Database and Expert Systems Applications*, pages 217–231. Springer, 2011.

[ZDO14] Rui Zhuang, Scott A. Deloach, and Xinming Ou. Towards a theory of moving target defense. In *In Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40. ACM, 2014.

[Zig18] Zigbee. Zigbee 3.0 - https://www.zigbee.org/zigbee-for-developers/zigbee-3-0/, January 2018.

[ZWJ+16] M. Zhang, L. Wang, S. Jajodia, A. Singhal, and M. Albanese. Network Diversity: A Security Metric for Evaluating the Resilience of Networks Against Zero-Day Attacks. *IEEE Transactions on Information Forensics and Security*, 11(5):1071–1086, May 2016.

[ZZD+12] Rui Zhuang, Su Zhang, Scott DeLoach, Xinming Ou, and Anoop Singhal. Simulation-based Approaches to Studying Effectiveness of Moving-Target Network Defense. *NIST*, November 2012.