Journal of Universal Computer Science, vol. 30, no. 9 (2024), 1140-1162
submitted: 27/9/2023, accepted: 30/5/2024, appeared: 14/9/2024 CC BY-ND 4.0

Exploiting TTPs to Design an Extensible and Explainable Malware Detection System

Yashovardhan Sharma
(University of Oxford, Oxford, UK
https://orcid.org/0000-0003-0054-9610, yashovardhan.sharma@cs.ox.ac.uk)

Simon Birnbach
(University of Oxford, Oxford, UK
https://orcid.org/0000-0002-2275-7026, simon.birnbach@cs.ox.ac.uk)

Ivan Martinovic
(University of Oxford, Oxford, UK
https://orcid.org/0000-0003-2340-3040, ivan.martinovic@cs.ox.ac.uk)

Abstract: In recent years, numerous sophisticated malware detection systems have been proposed, many of which are based on machine learning. Though such systems attain impressive results, they are often designed having effectiveness as the main, if not only, requirement. As a result, the effectiveness of such systems, especially if based on deep learning models, often comes with (*i*) poor extensibility, being very difficult to adapt and/or extend to other settings, and (*ii*) poor explainability, since it is often not possible for humans to understand the reasons behind the model's predictions, making further analysis of threats a challenge. In this paper we show how it is possible to design an extensible and explainable yet effective malware detection system. Extensibility is obtained thanks to the exploitation of TTPs (Tactics, Techniques, and Procedures) from the popular MITRE ATT&CK framework, which is an ontology of adversarial behaviour that allows us to divide the general problem of malware detection into the smaller problems of detecting the different types of malicious activity that can be carried out. Explainability is obtained by returning (*i*) which TTPs have been detected and are responsible for the classification of the entire behaviour as malicious, and (*ii*) why such TTPs have been classified as malicious. To demonstrate the viability of this approach we implement these ideas in a system called RADAR. We evaluate RADAR on a very large dataset comprising of 2,286,907 malicious and benign samples, representing a total of 84,792,452 network flows. The experimental analysis confirms that the proposed methodology can be effectively exploited: RADAR's ability to detect malware is comparable to other state-of-the-art non-interpretable systems' capabilities. To the best of our knowledge, RADAR is the first TTP-based system for malware detection that uses machine learning while being extensible and explainable.

Keywords: Malware, Intrusion Detection, Network Security, MITRE ATT&CK, Explainable AI, Cyber Security
Categories: D.2.0, I.2, K.6.5, L.4
DOI: 10.3897/jucs.131753

1 Introduction

The capability and sophistication of malware authors has increased in recent decades and novel attacks are continuously being developed. According to the 2021 Data Breach

Investigation Report by Verizon, nearly 20% of data breaches are a result of system intrusions, of which over 80% are due to malware [Bassett et al., 2021]. This, in conjunction with the increasing number of networked devices, has made it commensurately harder to defend both networks and devices from malicious attacks. To keep up with these rapidly evolving threats, more and more sophisticated malware detection systems are being designed, many of which are based on machine learning (see, *e.g.* [Kwon et al., 2019, Gibert et al., 2020, Shaukat et al., 2020]). Though such systems attain impressive results, they are often conceived having effectiveness as the main, if not only, requirement. As a result, the effectiveness of such systems, especially when based on deep learning models, often comes with (*i*) poor extensibility, being monolithic systems they are very difficult to adapt and/or extend to other settings, and (*ii*) poor explainability, since it is not possible for humans to understand the reasons behind the model's predictions, and also given their poor interpretability, this due to their inner complexity which makes it impossible to link the result of detection with its root cause. This is unfortunate: extensibility, explainability, and interpretability are important properties of any system in general, and in particular in the field of cyber security. Indeed, malware detection systems need to be continuously updated in order to deal with novel attacks, which, when detected, often need to be further analysed and inspected by domain experts (see also [Atzmueller and Kanawati, 2022]). Despite this, systems for malware detection are usually not designed to be extensible, and the vast majority of them are neither explainable nor interpretable.

In this paper we show how it is possible to design an extensible and explainable yet effective malware detection system. This system, called RADAR, (*i*) maintains an internal graph-based representation of the network in order to effectively reason about the network traffic generated by a given malicious/benign sample (hereby referred to as just *sample*), (*ii*) exploits TTPs from the industry-standard MITRE ATT&CK framework which describes post-compromise adversarial behaviour, to unequivocally identify and classify the potential malicious network flows exhibited by each sample, (*iii*) utilises a dedicated decision tree for each TTP to label each network flow as potentially malicious, and, (*iv*) combines the results obtained for the different TTPs in order to finally label each sample as either benign or malicious. As a result, it is relatively easy to extend the system with new components for detecting novel malicious behaviour using an updated set of TTPs, and/or modify existing components in order to broaden their capabilities using a combination of existing TTPs. Further, thanks to the joint use of TTPs and decision trees, RADAR is both an explainable and interpretable system, being able to output in human-understandable terms, (*i*) which are the high-level TTPs (and not just the low-level features) being used in the detected malicious sample, and (*ii*) why a given sample has been labelled as malicious. Additionally, the internal graph-based representation of the network, together with the exploitation of TTPs, provide the means to give valuable aid to system analysts in their evaluation of the reported malicious activity.

We evaluate RADAR on a very large dataset comprising of 2,286,907 malicious and benign samples, representing a total of 84,792,452 network flows. The experimental analysis confirms that the proposed methodology can be effectively exploited: RADAR's ability to detect malware is comparable to other state-of-the-art systems' capabilities, as reported in the literature (see, *e.g.*, [Malaiya et al., 2019]). For instance, RADAR's performance as judged by an AUC score of 0.87 is similar to what has been reported in the very recent paper by Dyrmishi et al. [Dyrmishi et al., 2023]. To the best of our knowledge, RADAR is the first TTP-based system for malware detection that uses machine learning while also being extensible and explainable.

The rest of the paper is structured as follows. Section 2 provides a detailed description of the RADAR design, while also presenting some of the technological hypotheses and choices that guided its implementation. RADAR is then evaluated in Section 3 according to different metrics and in various settings. Section 4 contains a discussion about the results of the development and testing of RADAR. Section 5 presents the most relevant related work and details how they compare to our work. We end the paper with a section on the conclusions and future work.

This paper is an extension of the work originally presented at the ‘‘2023 European Interdisciplinary Cybersecurity Conference (EICC’2023)” [Sharma et al., 2023]. In comparison with [Sharma et al., 2023], this paper presents a more detailed description and discussion of the proposed approach. The main new points presented in this paper are (*a*) a more detailed description of the system architecture, (*b*) a detailed presentation of the data structures used for the graph representation of the network, the advantages it provides, and the rationale for choosing the adopted technological solution in comparison with possible alternatives, (*c*) a detailed description of the results of the TTP detection, including the advantages of detecting TTPs in general and the ones we selected in particular, (*d*) a thorough and critical discussion of the topics covered, including the advantages as well as the limitations of RADAR.

2 System design

RADAR is a system for network traffic analysis and malware detection, designed to be both extensible and explainable. Extensibility is a well known software engineering and systems design principle that enables future growth and adaptation. In the field of cyber security, and malware analysis in particular, extensibility is a necessary criterion to design systems that can adapt in the face of novel and constantly-evolving attacks. Explainability has recently emerged as an important requirement for ML and AI systems. As stated in [Gunning et al., 2019], explainability is essential for users to effectively understand, trust, and manage powerful artificial intelligence applications in every application domain, including cyber security (see, e.g., [Atzmueller and Kanawati, 2022, Capuano et al., 2022]). Explainability calls for producing high-level outputs which can be rooted back to the inputs, i.e., for exploiting ML interpretable models for the classification of malicious activities. With such goals, RADAR has been designed to

1. be extensible, given its modular software architecture in which different TTPs are detected and classified by different ML engines, each of which can be easily extended and/or modified independently from the others, and
2. provide high-level explanations, given its ability to classify malicious activity using TTPs from the industry-standard MITRE ATT&CK framework, and its use of interpretable decision trees for this classification.

RADAR’s data pipeline is designed to be able to take arbitrary network packet captures corresponding to a sample as input, and output:

1. the relevant matching TTPs for that sample, and
2. a prediction as to whether the sample is malicious.

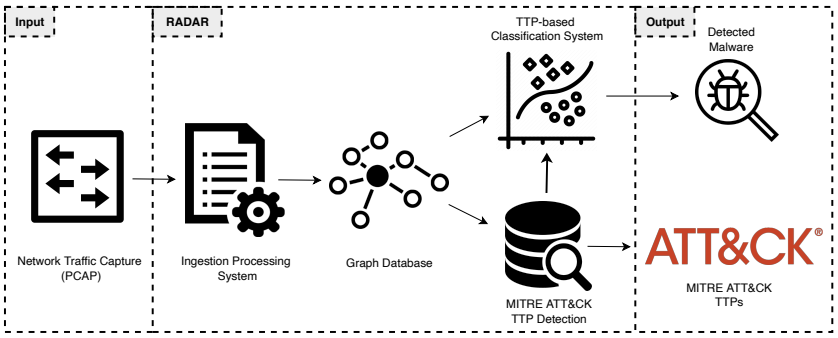


Figure 1: RADAR system overview.

This enables RADAR to be employed as a 'plug-and-play' technology wherein a packet capture can be sliced from a larger flow of network traffic and analysed. To further facilitate integrating RADAR into existing security infrastructures, it has been designed and implemented to also satisfy the following requirements:

1. Be lightweight and as efficient as possible to enable the storage and possible retention of large amounts of data.
2. Be content agnostic, *i.e.*, be based only on the analysis of the metadata, thus also working on encrypted data payloads and not violating possible privacy regulations that would otherwise limit the applicability of the system.
3. Have the ability to analyse arbitrary network captures, *e.g.*, independent of traffic type, protocol, or network topology.

The components of RADAR include an Ingestion Processing System that constructs a graph database of malware samples, a TTP Detection Engine based on the MITRE ATT&CK framework, and a TTP-based Classification System. We show an overview of our system architecture in Fig. 1. In the following subsections we shall cover each of the system components.

2.1 Data Ingestion and Graph Construction

The first stage in our data pipeline is the Ingestion Processing System that takes as input network traffic captures (PCAPs), and processes them into a format that is suitable for graph construction.

The possible data formats to implement the processing of packet captures broadly fall in one of the following three categories:

1. PCAPs. PCAPs contain the most granular level of detail with regards to extracting meaningful information. Given that our system input format is already PCAP, this would reduce one layer of processing and increase system efficiency. However, since these are raw network captures, these are large files that contain a lot of information, not all of which is required by our system. As a result, storage costs are likely to be higher and processing times per file would increase. In addition, PCAPs contain complete packet dumps, including data payloads and all other metadata, which could potentially cause ethical, legal, and privacy issues.

Field	Property
Flow Hash	The hash associated with each unique flow.
Sample Hash	The hash of the sample from which the flow is derived.
Unique ID	A unique ID associated with each flow.
Dataset	The dataset to which a flow belongs.
Source Port	The port from which the flow originates.
Destination Port	The port on which the flow connects to.
Start time, end time	Flow start or end time.
Duration	Flow duration in seconds.
Protocol	IP protocol identifier in decimal format.
Entropy	The Shannon Entropy for the flow payload.
Applabel	The application label, as identified by YAF.
Source IP, Dest. IP	Source and destination IPv4/IPv6 address.
Type, Code	ICMP type or code in decimal format.
Isn, Risn	Forward or reverse TCP sequence number.
Flags	4 properties representing various TCP flags.
Tag, Rtag	802.1q VLAN tag in forward/reverse direction of flow.
Pkt, Rpkt	No. of packets in forward/reverse direction of flow.
Oct, Roct	No. of bytes in forward/reverse direction of flow.
RTT	Round-trip time estimate in milliseconds.
End-reason	Reason for termination of flow.

Table 1: Flow properties.

2. NetFlows. NetFlows have also been used in a number of different existing solutions for network traffic analysis, as well as academic papers [Bilge et al., 2012, Grill et al., 2015]. Many Security Operations Centres (SOCs) also use NetFlows for the purpose of network traffic analysis. Their key limitation is the lack of detail about the underlying data. Typical NetFlows are only able to store basic information (Egress and Ingress port, protocol, and interface) about the network traffic. Due to the lack of information available from these flows, the analysis that can be performed on them is more shallow. This lack of data conversely means that the storage of NetFlows is compact—their processing is faster than comparable storage formats. Additionally, they are privacy-preserving since they do not contain any data payloads and rely mostly on metadata. NetFlows also have the unique performance advantage that they are usually generated directly by the networking hardware present in the network, thus not requiring any additional software, steps, or time to create them. This combined with their small size makes them suitable for real-time network traffic analysis, especially when dealing with large volumes of network traffic data.

3. Network logs. Network logs, like those generated by Zeek, are somewhere between PCAPs and NetFlows in terms of the granularity of data they contain. Zeek in particular has been used in academic research and existing solutions to detect malware, including the extraction of MITRE ATT&CK techniques. These logs are not comparable to PCAPs in terms of precision, but contain more information than a typical NetFlow while maintaining some of the storage and performance benefits. They are also content agnostic and thus preserve the privacy of the underlying network communications. However, these logs are typically application-specific and hence have no standard format. Additionally, they are wholly dependent on the application that generates them (e.g., Zeek), which forces the introduction of the corresponding software that creates these logs into the overall system pipeline, which may not be desirable from the perspective of security or flexibility. Lastly, they are typically generated only after raw data (e.g., PCAP files) is given as input to the application. This is an additional step that can only be performed after the network traffic in question has already been captured, thus making them unsuitable for real-time network traffic analysis.

Among the above network data formats, we selected NetFlows which rely solely on metadata, thus meeting the requirements of being content-agnostic and lightweight. In particular, we opted for a specific type of NetFlow, Yet Another Flowmeter (YAF) [Inacio and Trammell, 2010]. YAF generates bidirectional flows in IPFIX standard [Claise et al., 2008], which is based on the Cisco NetFlow V9 export protocol. Our primary motivation for selecting YAF is that it is designed for high performance and scalability across large networks. By design, it also takes into consideration the balance between capturing information and maintaining privacy on the network, and takes a hybrid approach between typical NetFlows and PCAPs. It is also more feature-rich than the typical NetFlow or network log, since it contains support for functionality such as application labelling (*i.e.*, port independent protocol checking) and entropy analysis (to determine if traffic is encrypted or compressed). The complete list of features, including those extracted using YAF, are presented in Table 1.

As can be seen from the table, these features are a combination of properties that we can extract from YAF (such as IPs, ports, and protocol) and custom features (such as flow hash, sample hash, and UID) that enable us to analyse a sample at different levels of granularity. For instance, a flow hash is computed on the basis of certain key values of a flow [`flow_hash: hash(src_ip, dest_ip, src_port, dest_port, protocol)`], and is not necessarily unique in the larger set of flows, whereas UID is unique per flow by design. Combined with the sample hash, this allows us to track similar flows and unique flows of interest, both within a given sample and in completely separate malware samples. This variation in granularity and aggregation helps make our TTP detection engine more robust in identifying sequences of flows that represent malicious behaviour.

Having selected and created the relevant features for our network traffic, we then map them into a graph, which represents the network traffic we want to analyse.

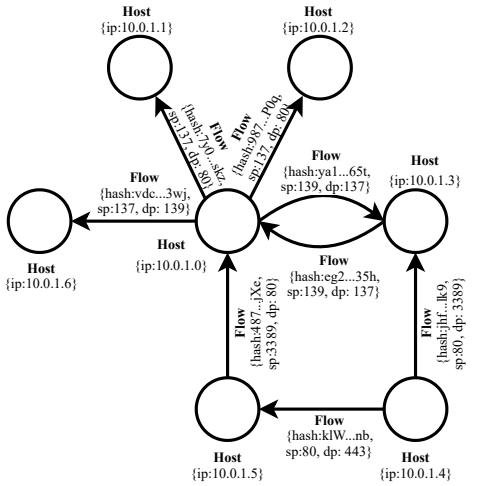


Figure 2: Graph schema for network traffic.

Formally, we create an annotated graph $G = (V, A, E)$, where:

- V is the set of vertices. Each vertex represents a host and is uniquely identified by its IP address in the network traffic.
- A is the set of edge annotations. Each annotation corresponds to an instantiation of the set of properties listed in Table 1.
- $E \subseteq V \times V \rightarrow A$ is the set of directed edges. Each edge corresponds to a flow between the two hosts.

Fig. 2 highlights this graph schema with the help of an example set of flows from a given sample. We choose this generalised form of graph entities and relationships precisely because it is applicable to all network traffic, and makes writing TTP detection logic independent of network protocol and packet formats. Having a universal form of expressing network data is also much more desirable from the perspective of easily extending and integrating the system.

We express the graph schema explained above in our system using the Neo4j Graph Database [Neo4j, 2022], which we can query using the Cypher query language. Features from the network captures are collated and processed together in a format that conforms with our schema. Subsequently, all flows associated with a given malware sample are inserted in this form into our graph database, at which point they can be analysed. The analysis of the data in the database takes advantage of its graph structure when querying the underlying network traffic to detect various host and flow configurations. We demonstrate this point by means of a simple example presented below.

Example 1. Let M be the number of flows in our dataset and d be the maximum out-degree among all the nodes. Suppose that we want to determine for a given sample, the total number of hosts a that when communicating with another host b, also result in b establishing a connection with another host c.
This network traffic is represented as a graph in RADAR, and using a Cypher-based notation, we write a query for this graph as follows:

```
MATCH (a:Host)-[f1:Flow]->(b:Host)-[f2:Flow]->(c:Host)
WHERE f1.sample_hash = f2.sample_hash
RETURN COUNT(a, b, c);
```

The first matching operation for $a \rightarrow b$ requires traversing all possible flows, and has time complexity $O(M)$. The second match operation, $b \rightarrow c$, on the other hand is a constant time operation since only the existence of an outgoing edge from b needs to be established. This is repeated at most d times, for each of the outgoing flows from b. The remaining filtering operations in the `WHERE` clause operate on a subset of the remaining flows, and are thus guaranteed to have an upper bound of $O(M)$. Hence, owing to the graph representation, the overall time complexity of this query is $O(Md)$.

Now let us consider the case where the same network traffic data is stored in a tabular format, such as in a standard relational database (RDB), as shown in Table 2. Each row in the table represents a flow, and each column denotes the flow properties as described

sample_hash	sip	dip	sp	dp	flow_hash	uid	...
xyz	10.0.1.10	10.0.1.1	88	80	eg123ya	47def75	...
abc	10.0.1.1	10.0.1.2	88	80	fm849ag	09pqr23	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Table 2: Tabular representation of network traffic data.

in Table 1. Then, using a SQL-based notation, we write this query as follows:

```
SELECT COUNT(DISTINCT ft1.sip)
FROM flow_table AS ft1, flow_table AS ft2
WHERE ft1.sample_hash = ft2.sample_hash AND ft1.dip = ft2.sip;
```

Here, the Cartesian Product of the table with itself in the `FROM` clause, results in a time complexity of $O(M^2)$. The overall complexity of the query remains $O(M^2)$ since the subsequent `WHERE` and `SELECT` clauses operate on a subset of the result of the Cartesian Product. □

Operation	Visualisation	Complexity (Graph)	Complexity (RDB)
Detection of all traffic traversing at least 3 hosts		$O(Md)$	$O(M^2)$
Detection of all traffic traversing at least $k > 3$ hosts		$O(Md^{k-2})$	$O(M^{k-1})$
Detection of all bidirectional flows between 2 hosts		$O(Md)$	$O(M^2)$
Detection of all bidirectional flows traversing $k > 2$ hosts		$O(Md^{k-1})$	$O(M^k)$

Table 3: From left to right: examples of possible operations on network traffic data, their graph visualisations, time complexity of their associated query if data is represented as a graph, and time complexity of their associated query if data is represented using a relational database.

The benefits in query performance in terms of time complexity given by the graph representation is not limited to the query in Example 1. This and other similar operations useful in analysing network traffic data can be generalised further and the overall benefit of the graph-based modelling can be realised much more concretely. We present some of these operations, their graph visualisation, and their graph-based and RDB-based time complexities in Table 3. As can be seen in the table, the graph representation used by RADAR is more efficient than standard RDB-based modelling.

2.2 TTP Detection

The TTP Detection Engine analyses all flows in the graph database and executes detection rules on the graph entities and relationships to identify any matching MITRE ATT&CK technique. The first two columns of Table 4 show the complete list of TTPs that our system currently supports (notice that T1021 and T1563 have two sub-techniques each).

The set of rules utilised by our TTP Detection Engine has been selected being a good representative of the standard malicious activity carried out on networks, diverse enough in order to stress the ability of the system to detect different kinds of attacks. The TTPs we consider broadly falls into two categories:

1. **Feature-based detection rules.** This type of detection relies on specific features of the flows described in Table 1. For instance, properties like `src_port`, `dest_port`, `protocol`, and `applabel` are used to identify flows with specific values for these properties that correspond to those exhibited by specific TTPs.
2. **Heuristic-based detection rules.** This type of detection relies on the analysis of multiple flows and uses heuristics based on their properties. For instance, comparing properties such as `pkt` and `rpkt`, looking at the order and timing of connections made to `dest_port`, or analysing the `entropy` corresponding to a data payload, can provide useful information about the specific TTPs being used.

Tactics	Techniques	Malicious	Benign	Class.
Reconn.	T1590 - Gather Victim Netw. Inf.	4	4	✗
Cred.Access	T1557.001 - Man-in-the-Middle	7	0	✗
Discovery	T1124 - System Time Discovery	1,645,252	103,461	✓
	T1135 - Network Share Discovery	8224	41	✓
Lateral Movement	T1021.001/4 - Remote Services(RDP/SSH)	666	20	✗
	T1550.003 - Use Alt. Auth. Material	241	1	✗
	T1563.001/2 - Rem.Serv.Sess.H.(RDP/SSH)	3	0	✗
	T1570 - Lateral Tool Transfer	1	0	✗
Command and Control	T1071 - Application Layer Protocol	1997	15	✓
	T1090 - Proxy	965	2	✗
	T1105 - Ingress Tool Transfer	938	163	✓
	T1571 - Non-Standard Port	1,970,237	119,501	✓
Execution	T1053 - Scheduled Task/Job	10	5	✗

Table 4: Supported MITRE ATT&CK TTPs. The first column shows the tactical goals of the adversary. The second column shows the (sub-)techniques used to achieve them. The third and fourth columns indicate the absolute number of malicious and benign samples per TTP in our dataset. The last column indicates whether it is feasible to train a classifier for the given TTP.

We demonstrate the utility of both types of rules and the specific logic required with the help of two examples.

Example 2. For the feature-based rules, we consider the MITRE ATT&CK technique T1571 (Non-Standard Port). This technique is part of the "Command and Control" (C&C) tactic and is applicable to the case when a host attempts to communicate with another host using standard network protocols but on a non-default port. Malware can attempt such a connection for several reasons—to communicate using a random port, evade firewall rules blocking certain default ports, or to find an open port for the purpose of establishing a C&C channel, among others. In order to detect this technique we rely on the features of the flow the two hosts use to communicate with each other. Specifically, we compare the network protocol in the flow with the port(s) used during communication and determine if these correspond to their defaults. In order to do this, we utilise YAF during the Ingestion Processing phase to determine the application layer protocol of every

flow independent of the port used. This feature is extracted and stored in our graph as the flow property `applabel`. Our detection logic maintains a list of the 50 most popular application layer protocols and their corresponding default ports, and compares these to the features of `applabel` and `port` present in every flow. A mismatch between these features is a positive match for T1571.

Example 3. For the heuristic-based rules, we consider the MITRE ATT&CK technique T1071 (Application Layer Protocol). This technique is also part of the C&C tactic, and is applicable when an application layer protocol is used to hide malicious traffic. Specifically, we attempt to identify this technique by classifying flows as anomalous on the basis of their incoming and outgoing traffic relative to other flows. For each malware sample, we isolate its flows and establish a mean packet outflow to packet inflow ratio through their directed edges, and determine a unique baseline for each sample. Then by defining a threshold over which there is far more data egress than ingress as compared to the baseline, we identify flows and hosts that are potentially communicating via a C&C channel or exfiltrating data from the network. Fig. 3 shows the distribution of this ratio against a variable threshold over a subset of our data. This threshold is designed to be dynamically adjusted based on the distribution of the network traffic and the characteristics of what is considered "normal" in a given network or environment.

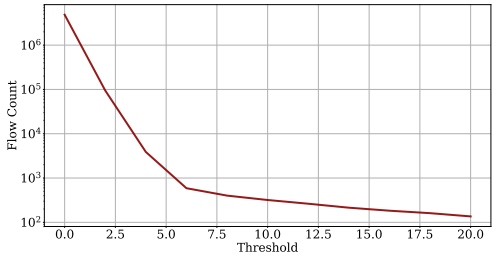


Figure 3: Distribution of flows with a varying threshold of packet outflow/inflow ratio, with a logarithmic Y axis.

At the end of this process, the system outputs a list of flows and their corresponding TTPs. This information is further aggregated by combining all the flows belonging to the same sample and then passed on to the TTP-based Classification system.

2.3 TTP-based Classification System

The TTP-based Classification System checks whether the flows in each sample exhibit malicious behaviour when compared to other flows matching the same TTP, and then flags a sample as malicious if ''enough'' flows exhibit malicious behaviour. The TTP-based Classification System works at deployment time following the steps in Figure 4.

As shown in the figure, the TTP-based Classification System takes as input: (*i*) the flows' properties (as listed in Table 1), and (*ii*) the TTPs detected in each flow, as returned by the TTP Detection Engine. Following a ''divide-and-conquer'' approach, it consists of one base classifier for each analysed TTP. This choice allows us to compare

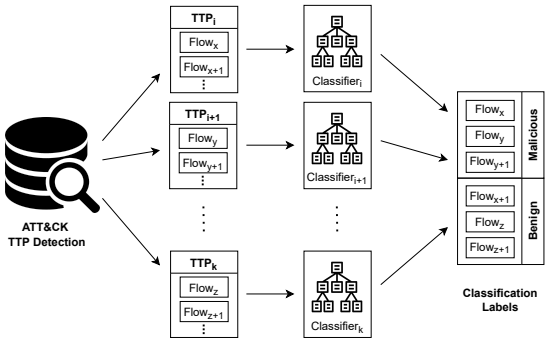


Figure 4: TTP-based Classification System Pipeline

the behaviour of each flow with only those flows that match the same TTPs.

For each flow f, we create a vector x containing all the features listed in Table 1. Then, x is passed to all the base classifiers that have been trained on the flows that matched the same TTPs as f. Thus if, for example, f matched TTP_i, TTP_{i+1} and TTP_k then x will be passed just to the ith, $(i + 1)$th and kth classifiers. Each of the selected classifiers then independently decides whether f presents malicious behaviour when compared to the other flows flagged with the same TTP. Notice that in general any classifier can be used for this task. However, in order to preserve the interpretability and explainability of the system, we opt to use decision trees [Wu et al., 2008]. Indeed, a decision tree is a transparent model whose predictions can be easily explained via Boolean logic rules. This is not true for most classifiers (*e.g.*, based on artificial neural networks) which are often completely opaque models whose predictions are difficult to interpret. As a result of this choice, it is possible to see exactly which network properties are used to make a classification decision, and this information can be quite useful to SOC analysts by helping them triage alerts in a more informed manner.

Once we have the prediction for each flow in the sample, we have to find a way to aggregate them and decide whether the sample is malicious or not. To this end, we devise three policies:

1. P1: the first policy relies on n_t, the number of unique TTPs matched by the malicious flows in the sample. If $n_t \geq \theta_t$ then the sample is considered malicious. θ_t is a parameter that the user can tune.
2. P2: the second policy relies on n_f, the number of malicious flows in the sample. If $n_f \geq \theta_f$ then the sample is considered malicious. θ_f is a parameter that the user can tune.
3. P3: the third policy relies on p, the percentage of malicious flows in the sample. If $p \geq \theta_p$ then the sample is considered malicious. θ_p is a parameter that the user can tune.

The first policy, P1, is devised on the ground of the hypothesis that the more unique TTPs a sample maliciously uses, the more likely it is that the sample is engaged in malicious activity. However, this policy can become quite strict very quickly, as it is unlikely that

the maliciousness of a sample will increase linearly with the number of TTPs after a certain point. We thus devise an alternate policy, P2, which is based on the number of malicious flows in a sample. The benefit of such a policy is that it does not rely solely on the presence of unique TTPs, and instead takes into consideration the fact that the overall repeated malicious usage of TTPs might be a good indicator of the sample's maliciousness. However, this policy's reliance on absolute values makes it susceptible to missing samples with lower numbers of malicious flows. Hence we devise our third policy, P3, which takes into account the percentage of malicious flows in the sample. It has similar benefits as P2 by virtue of utilising the repeated usage of TTPs and not relying solely on the presence of unique TTPs, however it uses the relative proportion of malicious flows in the overall sample to determine maliciousness. Notice that integrating other custom policies is quite straightforward, *e.g.*, assigning different weights to the TTPs and/or to the multiple uses of the same TTP. Thus, by designing and tuning policies for specific contexts, further performance gains can be obtained.

3 Evaluation

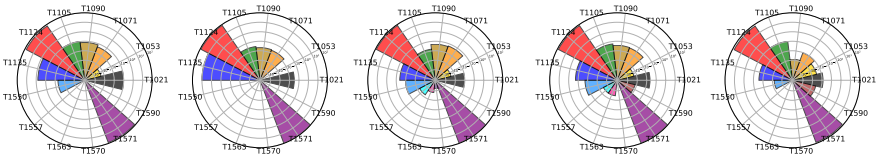
We take a similar approach to many well-known systems in the literature when evaluating RADAR, wherein we test our detection capabilities against real-world malware datasets [Paxson, 1998]. In doing so, we use an especially large dataset of network flows collected from malware samples for testing. We execute our samples within a Cuckoo sandbox on VirusTotal. All samples are allowed to freely create any network connections they wish, and the network traces of their actions are collected. Each sample is executed for two minutes within the Cuckoo sandbox, as research has shown this is sufficient for observing the majority of a sample's behaviour [Kuechler et al., 2021]. The resultant PCAPs corresponding to each sample are then given as input to RADAR in order to commence analysis.

Dataset	Samples	Flows
Ember Malicious [Anderson and Roth, 2018]	435,741	26,567,527
MalRec [Severi et al., 2018]	37,763	12,273,502
MalShare [Project, 2022]	1,268,923	32,310,907
VirusShare [VirusShare.com, 2022]	595,098	12,217,493
Ember Benign [Anderson and Roth, 2018]	155,432	1,423,023
Total (unique)	2,286,907	84,792,452

Table 5: Overview of all the datasets. The first four datasets contain only malicious samples, while the last one contains only benign samples.

Table 5 shows the complete list of datasets, along with their properties, that we use to test our system. Ember is an open dataset of hashes of malicious Windows executables created as a benchmark for machine learning models. It contains 3 types of hashes: malicious, benign and unlabelled. For our purposes, we utilise both the malicious hashes (as Ember Malicious) and the benign hashes (as Ember Benign) in our dataset. MalRec is a small dataset created as a result of a running malware on a dynamic analysis platform, which was collected over a two-year period. MalShare is a community-driven open repository of malware samples, which features over a million hashes, from which the corresponding dataset is derived. Similarly, VirusShare is a repository that aims to

help security researchers analyse selected strains of malware, and the source of our corresponding dataset. We select these datasets as they are representative of modern commodity malware and allow for the reproducibility of results. It is important to note that we do not utilise the entirety of each dataset in our experiments, since we filter out samples that do not exhibit any network activity. Overall, we analyse a total of 84,792,452 flows from 2,286,907 malicious and benign samples.



(a) Ember Mal. *(b) MalRec* *(c) MalShare* *(d) VirusShare* *(e) Ember Benign*

Figure 5: Percentage of samples matching various MITRE ATT&CK TTPs across each dataset, with a logarithmic Y-axis.

3.1 TTP Extraction

Detected TTPs and Significance. We deploy RADAR against the datasets shown in Table 5 to detect the TTPs our system supports. Table 4 shows the instances of the various MITRE ATT&CK TTPs that RADAR is able to detect in both our malicious and benign datasets and the total number of samples for which the corresponding TTP is detected in the datasets. As seen in Table 4, we are able to detect nearly every type of MITRE ATT&CK TTP that RADAR supports in our dataset, though there is large variance in their frequency. We also analyse the frequency of distinct TTPs per malware sample in our datasets. We observe that only 8.62% of samples are using no detectable techniques while for almost all the other samples RADAR is able to detect 3 or more techniques, indicating that our broad selection of TTPs is significant being able to identify malicious activity within the vast majority of malware samples within our datasets.

Variance of TTPs Across Datasets. We further explore the distribution of different techniques across all our datasets. Figure 5 shows the percentage of samples matching each MITRE ATT&CK TTP in every dataset. We find that certain TTPs have a high prevalence in all our datasets (*e.g.*, T1124 and T1571), while others occur with varying frequency depending on the specific dataset. These heterogeneous distributions are to be expected since each of these datasets is from different sources, and indicate that for the best performance, the system has to be tuned depending on the specific operational context. While the individual prevalence of certain techniques in our malware datasets shows the unique composition of the types of malware in each dataset, the frequency of occurrence of certain techniques in our benign dataset showcases some of the differences between the TTPs utilised by malicious and benign samples. For instance, T1550 (Use Alternate Authentication Material), T1557 (Man-in-the-Middle), and T1563 (Remote Service Session Hijacking) can be found in the MalShare and VirusShare datasets, while of these only T1550 is detected in Ember Malicious, while MalRec contains none of these TTPs. On the other hand for instance, the occurrence of T1090 (Proxy) is relatively infrequent in the benign dataset when compared to the malicious datasets. This would

logically follow since benign samples are unlikely to utilise proxy services like Tor as frequently as malware samples.

Malicious Usage of TTPs. Our results highlight that techniques such as T1571 (Non-Standard port), T1124 (System Time Discovery) and T1135 (Network Share Discovery) are the most frequently occurring across all our malicious datasets. Other techniques used by malware in significant numbers include T1071 (Application Layer Protocol), T1021 (Remote Services), T1105 (Ingress Tool Transfer), and T1090 (Proxy). Whilst we discover evidence of other TTPs being used by malware, they are far less prevalent in their usage, but still indicate usage by a non-negligible amount of samples. The most commonly occurring TTP in our dataset is T1571 (Non-Standard Port). This shows a high prevalence of malware making use of ports that are not the default original source or destination port for a given application protocol. Whilst this technique has been written about in various papers and threat intelligence reports by antivirus firms (see, *e.g.*, [SophosLabs Research Team, 2019, Rafique and Caballero, 2013]), our pervasive detection of T1571 in all the different datasets affirms that this is a technique used by a large variety of different malware samples 'in-the-wild'.

Benign Usage of TTPs. We also observe a high number of TTPs being detected in our benign dataset. Specifically, T1571 (Non-Standard Port) and T1124 (System Time Discovery) are by far the most frequently occurring TTPs in our benign dataset. This is not surprising considering that benign software can be easily responsible for both these behaviours: (*i*) many applications can use non-standard ports to establish connections via commonly-used protocols (such as connecting via HTTPS on a port other than 443) and, (*ii*) several applications rely on having an accurate measure of time (say by contacting an Network Time Protocol server) to function correctly. This highlights the challenge of using solely TTPs to determine the maliciousness of a sample.

3.2 Classification performance

In this subsection, we evaluate how well RADAR can distinguish between malicious and benign use of TTPs on the dataset obtained by merging the datasets in Table 5.

Experimental Setup. *Dataset Creation per TTP.* Given our a dataset, we first need to assess whether enough flows/samples are matching a TTP to train the corresponding classifier. The results of this analysis are reported in the last column of Table 4. If we do not have enough flows/samples matching a TTP to train the corresponding classifier, we just classify each flow matching the TTP as malicious, and then we use the desired policy to conclude whether the sample is malicious or not. For those TTPs for which training a classifier is feasible (*i.e.*, TTPs: T1071, T1105, T1124, T1135 and T1571) we then need: (*i*) a "benign"/"malicious" label for each flow, and (*ii*) a training, validation and test set for each classifier. Regarding the labels, we simply assign a flow the label "malicious" (resp. "benign") if the sample to which it belongs is malicious (resp. benign). Regarding the creation of the datasets splits, we need to make sure that: (*i*) flows from the same sample do not belong to different splits, (*ii*) flows matching multiple TTPs belong to the same split for the different classifiers. To achieve the above we perform two steps. Firstly, we split the dataset obtained as the union of the datasets in Table 5 in train, validation and test sets, such that: the train set contains 60% of the samples, the validation set 10% of the samples, and the test set 30% of the samples. Secondly, for each flow f in the training set and for each TTP T, we check whether f matched T,

and if that is the case, then f is added to the training set of the classifier for the TTP T. The same procedure is then repeated for all the flows in the validation and test set. This ensures that the splits for each classifier retain the desired properties.

Decision Tree Training. Each decision tree is then trained on the training set of the relevant TTP by maximising the Gini gain at each splitting step. In order to alleviate the imbalance problem across our malicious and benign datasets, we associate weights to each of the two classes that are inversely proportional to the class frequency in the training set. Further, for each decision tree associated to a TTP, we optimise two hyper-parameters: (*i*) the maximum depth of the decision tree, and (*ii*) the minimum number of samples in each leaf. For the maximum depth we try all the possible depths between 1 and 50, together with the option of not setting a maximum depth at all, while for the minimum number of samples we test values 1, 100 and 1,000. For each TTP, we then return the hyper-parameters that achieved the highest F1-score for the given TTP. While any other metric can be chosen, we pick F1-score in order to strike a balance between precision and recall (*i.e.*, true positive rate). To ensure the reproducibility and robustness of our results, we initialise the training of each of our decision trees with five different random states, and then subsequently proceed with validation and testing for each state. The random state parameter controls the randomness of the decision tree. We average out the results from each random state for every decision tree and report only the averaged values going forth. We also compute 95% confidence intervals for each metric, however we do not plot these intervals as the range of the standard deviations is quite small. Indeed, the maximum standard deviation we register across all policies and thresholds is equal to 1.1×10^{-4}, thus indicating the stability of our models.

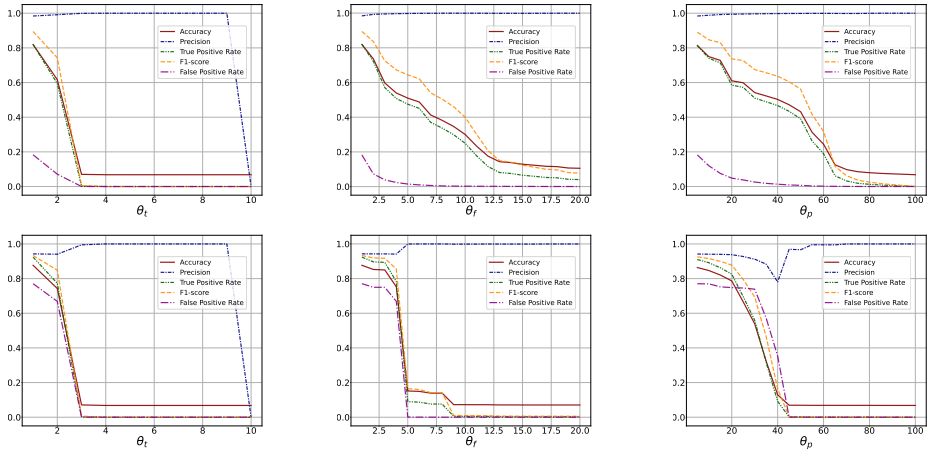


Figure 6: RADAR's (top figures) and RB-RADAR's (bottom figures) performances when using the 3 aggregation policies P1 (left), P2 (middle) and P3 (right) while varying the user defined threshold.

Comparison of Different Aggregation Policies. The first analysis we conduct is a comparison of the different aggregation policies described in Section 2.3, considering 5 metrics: (*i*) accuracy, (*ii*) precision, (*iii*) true positive rate, (*iv*) false positive rate, and

(v) F1-score. Each of these metric is necessary to better understand the capabilities of the system and to tune the policy parameters as necessary. For example, choosing a higher threshold will probably minimise the false positive rate, however, it is equally important to understand the impact this will have on the true positive rate. In Figure 6/top we plot the performance of each of the policies according to the listed metrics while varying the decision thresholds. As we can see from Figure 6/top/left, P1 has good performance for low θ_t (*i.e.*, $\theta_t = 1$ and $\theta_t = 2$), however, as expected, it soon becomes too strict and its true positive rate drops below 0.01 for $\theta_t = 3$. On the other hand, as we can see from Figure 6/top/middle and Figure 6/top/right, this does not happen for either P2 or P3. Further, we observe that all the policies start at almost the same values for our metrics (notice that P1 and P2 are equivalent for $\theta_t = \theta_f = 1$), but then different policies guarantee different levels of sensitivity when tuning their respective parameters, with P3 presenting the smoothest trend.

Comparison with Exclusively TTP-based Approach. The second analysis we conduct is to evaluate the need of the decision trees. To this end, we compare the results obtained by RADAR with Rule Based RADAR (RB-RADAR), which is RADAR without the TTP-based classification system. Exactly like RADAR, RB-RADAR is able to match each flow to the TTPs listed in Table 4. However, instead of using decision trees, RB-RADAR considers each flow matching at least one TTP as malicious. Then, in order to conclude whether a sample is malicious or not, it uses the same 3 policies—P1, P2, P3—as RADAR. For this analysis we use all the same metrics as before, and we plot the results in Figure 6/bottom.

Firstly, we notice that for $\theta_t = \theta_f = \theta_p = 1$, RB-RADAR achieves not only high accuracy, precision, true positive rate, and F1-score, but also a very high false positive rate (nearly 0.8). In contrast, for the same thresholds, RADAR obtains a false positive rate below 0.2 while keeping the value associated to the other metrics high. Secondly, we observe that when RB-RADAR achieves a false positive rate less than 0.2 (with P1 for $\theta_t = 3$, with P2 for $\theta_f = 5$, and with P3 for $\theta_p = 45$), this results in a drastic reduction in accuracy, true positive rate and F1-score, which all become less than 0.2 as well (the precision, similar to RADAR, is very high no matter the chosen threshold). This shows that using only TTPs for malware classification is an unsuitable approach, and using a two-step classification system based on decision trees, is a far better strategy.

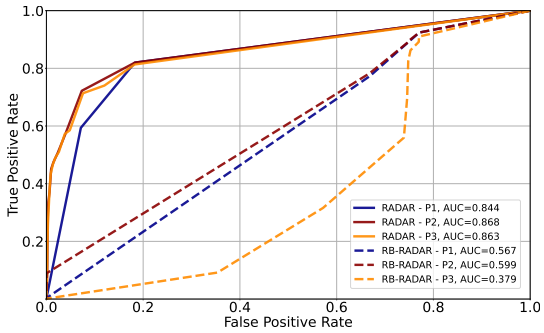


Figure 7: ROC Curves comparing RADAR and RB-RADAR.

Finally, in order to have a threshold independent comparison we plot the receiver operating characteristic (ROC) curve for each system and each of the three policies. We further compute the area under the curve (AUC) for each of these ROC curves. The results of this comparison are plotted in Figure 7. As we can see from the figure, while RADAR is able to achieve an AUC always greater than 0.8, the performance of RB-RADAR is close to random classification when using P1 and P2, and worse than random classification when using P3. This further highlights that using decision trees to classify malware on the basis of TTPs improves performance while maintaining a balance between true positive and false positive rate.

4 Discussion

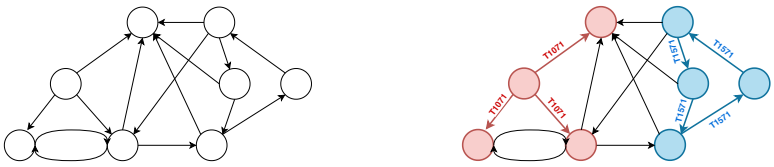
The development and testing of our system has interesting applications for malware analysis and the measurement of malware behaviour. In this section we discuss the results of our system development and testing.

Prevalence of Techniques. Based on our TTP extraction results, we find that only 8.62% of samples across all our datasets had no MITRE ATT&CK TTPs detected within them. While for just our malicious datasets, this number drops further to 7.56%. This suggests that TTPs are highly prevalent in malware and thus can be exploited for malware detection. The ATT&CK TTPs that we discovered to be prevalent within our datasets are also interesting. For instance, the use of non-standard ports to mask or blend-in malicious flows has been described by many threat intelligence reports and papers in isolated incidents and small datasets. However our research concludes that this is a common technique across the vast majority of malware in all of our datasets.

Variance of Techniques. Our results show that there is variance among the techniques used within each dataset. The frequency of these techniques is different since the underlying nature of the commodity malware datasets differs from one another. We find that some of these TTPs are more common across datasets, but also that the distribution of these TTPs is not uniform. This variation can be used to evaluate the TTPs that are currently being employed by malware authors and concentrate resources for mitigation accordingly. At the same time, the different usage of TTPs across benign and malicious samples, are strong indicators of malicious activity if the TTPs are primarily used by malware.

False Negatives. Previous work has shown that evasion of detection systems by changing features of network traffic has been a long-standing technique used by different malware families to evade network-based detection [Gorecki et al., 2011, Bayer et al., 2009]. We take steps to mitigate false negatives like this by detecting divergence from the standard port usage, as detailed in Section 2.2. This is a validation step not taken by comparable systems and a practical technique for threat intelligence analysts in a SOC.

Detection Sensitivity and Thresholds. During the course of system development, we found that selecting the appropriate detection sensitivity for various techniques was crucial. An improperly configured system which is not tuned to a specific environment could result in a high number of false positives. Network activity is not uniform or consistent across different environments, and requires tuning detection thresholds to match that environment. Our detection functionality offers configurable thresholds wherever



(a) (b)

Figure 8: Network traffic graph views before (fig. (a)) and after (fig. (b)) the TTP Detection phase. The coloured nodes and edges represent hosts and flows respectively, which correspond to a detected TTP. In this example, the red flows represent the presence of T1071 - Application Layer Protocol while the blue flows represent T1571 - Non-Standard Port.

possible, as demonstrated in Section 2.2. The experience and domain-knowledge of an analyst can be leveraged whilst utilising such a system in a SOC.

Opportunity for Improving Malware Detection. Our system is designed to integrate cohesively with existing IDS and EDR systems. The goal of our system is to improve detection of malicious behaviour by operating in conjunction with other systems. For instance, an IDS that utilises host-based analysis of malware can be paired with the network-based analysis of our system to more comprehensively analyse a given sample. Additionally, our system's focus on broader tactics and techniques make it useful across a broad spectrum of malware, the effectiveness of which can be increased by combining it with other NIDS that focus on the detection of specific malware families.

Decision Trees and Effectiveness. Our results highlight the importance of exploiting decision trees in our ML pipeline. We find that a system relying solely on TTPs can have a false positive rate almost as high as 0.8, if tuned to maximise other metrics such as true positive rate, precision, accuracy, and F1-score. On the other hand, using a classification pipeline with decision trees reduces the false positive rate to less than 0.2 in the worst case, where each of the other metrics is maximised. As discussed earlier, any classifier can be used for the classification of the malicious activity in RADAR. However, when we performed some experiments using other popular ML models (in particular, Adaptive Boosting, Logistic Regression, Deep Neural Networks, Random Forest, EXtreme Gradient Boosting, and Support Vector Machines), we found that there was no significant variation in the performance, and indeed resulted in a less transparent and thus less explainable system.

Extensibility and Explainability. RADAR is clearly an extensible, interpretable and explainable system thanks to its modular architecture, the exploitation of decision trees, the definition of clearly defined procedures in each stage of the decision process. Here, we stress the importance of our graph-based representation of the network and of exploiting TTPs for explainability. Indeed, whenever a sample is labelled as malicious, the relevant malicious TTPs detected can be visually represented by our system, allowing for further visual inspection by a SOC analyst. Fig. 8a shows the state of the graph representing the network before the TTP Detection phase commences. Despite containing all the network traffic data, this graph is not particularly informative to a SOC analyst during analysis. However as Fig. 8b shows, once the final phases of RADAR have completed, there is a lot more visual information for a SOC analyst to use for further investigation. The flagged hosts and flows corresponding to a TTP detection can be clearly marked, coloured, and annotated with their respective TTP. An analyst can then immediately take

action based on the information presented in the graph.

Scalability. The modular nature of RADAR is particularly beneficial when dealing with larger and more complex networks. Firstly, since individual TTPs can be implemented and their specific decision trees added, the system's capabilities can be expanded as per the requirements of the network. Secondly, while we evaluate our system using a network based in a dynamic analysis environment, RADAR is designed to handle traffic from much larger and more complex networks. The graph construction we utilise in Section 2.1 demonstrates the benefits of using our approach when the network is scaled beyond just a few hosts, and the nature and topology of the connections being made becomes far more complex.

Limitations. Our system is a network-based detection system and as such will not be effective against malware that only executes locally and has no network footprint. We are also restricted to analysing the features we extract from our source data. Whilst it may be possible to have a more granular analysis by using all possible information captured within a PCAP, we have attempted to strike a balance between the granularity of our data and the privacy and efficiency of our system by selecting YAF NetFlows, as discussed in Section 2.1. Other limitations of general dynamic analysis systems also apply to us since we analyse samples whose network activity traces have been captured in dynamic analysis environments. The network traces captured are time-limited and may not necessarily represent the entire range of malware activity, and as such any analysis on them may not be exhaustive. We address this by selecting an execution cutoff-time parameter that allows for the majority of a sample's behaviour to be observed, as elaborated in Section 3. Additionally, malware may specifically evade detection systems by changing the features of its network traffic [Polychronakis et al., 2008, Gorecki et al., 2011, Bayer et al., 2009]. We mitigate this by detecting divergence from the standard port usage, as detailed in Section 2.2. Lastly, RADAR is only as effective as the set of TTPs it can detect, and as such enlarging this set can increase coverage. We address this by selecting a broad variety of TTPs that encompass a large subset of the behaviours exhibited by network-based malware—our results show that we were able to detect TTPs in 91.38% of all samples and 92.44% of malicious samples, thus indicating that our chosen subset of TTPs is quite effective at detecting malicious behaviour generally.

5 Related Work

In the literature, research related to utilising TTPs has so far focused on: using TTPs to track the evolution of malware and describing emerging trends [Chierzi and Mercês, 2021, Oosthoek and Doerr, 2019], locating TTPs in the control flow graph describing the execution flow of a malware executable [Fairbanks et al., 2021], automatically extracting TTPs from Cyber Threat Intelligence (CTI) reports [Mendsaikhan et al., 2020, Legoy et al., 2020, Husari et al., 2017], discovering existing inter-dependencies in a TTP chain [Al-Shaer et al., 2020], or detecting Living-Off-the-Land (LotL) malware techniques and Advanced Persistent Threat (APT) attack campaigns [Kuppa et al., 2019, Hassan et al., 2020]. These works highlight the varied manner in which TTPs have been utilised within the broad domain of cyber security.

More closely related to our problem domain is the work by [MchPhee, 2020], who use the network IDS Zeek [Paxson, 1998] to extract MITRE ATT&CK TTPs from network

traffic. Their paper discusses methods to identify several TTPs and provides prototypical analysis scripts to test a few of these TTPs. BZAR [MITRE Corporation, 2020] is another such example which has similar capabilities, as it enables the extraction of certain MITRE ATT&CK TTPs from Zeek log files. While RADAR also has a TTP Extraction phase, it goes significantly further than these systems in terms of its functionality and capabilities. For instance, just comparing this phase alone, McPhee's scripts identify 3 techniques, while BZAR can identify 12 techniques and 8 sub-techniques across 9 tactics. RADAR on the other hand utilises a selection of 9 techniques and 6 sub-techniques suited to NetFlow traffic, that altogether encompass 11 tactics from the ATT&CK framework. Compared to BZAR where a majority of techniques (13/20) cover just 3 tactics, RADAR is thus more suitable to detecting a wider variety of malicious activity. Additionally, unlike BZAR where most TTPs (18/20) rely primarily on DCE-RPC protocol messages to enable detection, RADAR does not depend upon any one protocol for its detection capabilities and hence can handle more versatile network traffic. Furthermore, while BZAR only allows for the configuration of the 'epoch' threshold used to set the period over which it analyses data to detect TTPs, RADAR supports a higher degree of configurability by virtue of: (*i*) supporting thresholds for specific TTPs, (*ii*) allowing the selection of different detection policies for the overall system, and (*iii*) having policy-specific thresholds, which altogether enable a granular balance between the true-positive and the false-positive rate in the TTP detection phase. Lastly, we go even further by using these TTPs to effectively design an extensible system able to classify malware based on NetFlows. While doing so, our system explicitly separates malicious from benign usage of TTPs, and exploits interpretable models to provide an explainable classification of maliciousness to an analyst using it. Such functionality is not present in any comparable work, and to the best of our knowledge, is proposed for the first time in RADAR.

6 Conclusions and Future Work

In this paper we propose RADAR, a system that is capable of extracting MITRE ATT&CK TTPs from arbitrary network captures and using them to determine if the network traffic represents malicious behaviour. RADAR has been designed to be extensible and explainable. We demonstrate the effectiveness of RADAR by testing it against a dataset comprising of over 2 million samples, showing that RADAR is able to detect malware with an AUC score of 0.868. Our experimental results also highlight that TTPs occur frequently in both malware and benign samples. As a result, TTPs alone are insufficient to detect malicious activity and decision trees are fundamental in achieving these results. To the best of our knowledge, RADAR is the first system that is able to effectively and automatically separate malicious usage of TTPs from benign usage, while also using interpretable models to provide an explainable classification of maliciousness to an analyst using the system. Given the modularity of RADAR, other TTPs, machine learning models, and/or policies can easily be implemented and tuned for the specific intended use-case of the system. Indeed in future work, we plan to exploit these properties to further increase performance by (*i*) adding more TTPs, since this should correspond to a better profiling of each sample, and (*ii*) experimenting with other policies which are tuned for a specific use-case, e.g., by assigning weights to the TTPs and tuning them on the basis of the behaviours we want to detect. As per some preliminary experiments, we do not expect significant improvements in performance by using other machine learning (or even deep learning) models, which in any case come at the high price of sacrificing the interpretability of the system.

7 Datasets

To further facilitate research in this domain, we have publicly released some of the datasets we have utilised in our research. The repository with our datasets can be found here: https://github.com/baddymaster/BOF24.

References

[Al-Shaer et al., 2020] Al-Shaer, R., Spring, J. M., and Christou, E. (2020). Learning the Associations of MITRE ATT&CK Adversarial Techniques. In *IEEE Conference on Communications and Network Security (CNS)*. IEEE.

[Anderson and Roth, 2018] Anderson, H. S. and Roth, P. (2018). Ember: An open dataset for training static pe malware machine learning models.

[Atzmueller and Kanawati, 2022] Atzmueller, M. and Kanawati, R. (2022). Explainability in cyber security using complex network analysis: A brief methodological overview. In *European Interdisciplinary Cybersecurity Conference*. ACM.

[Bassett et al., 2021] Bassett, G., Hylender, D., Langlois, P., Pinto, A., and Widup, S. (2021). Data breach investigations report.

[Bayer et al., 2009] Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., and Kruegel, C. (2009). A View on Current Malware Behaviors.

[Bilge et al., 2012] Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., and Kruegel, C. (2012). Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138.

[Capuano et al., 2022] Capuano, N., Fenza, G., Loia, V., and Stanzione, C. (2022). Explainable artificial intelligence in cybersecurity: A survey. *IEEE Access*.

[Chierzi and Mercês, 2021] Chierzi, V. and Mercês, F. (2021). Evolution of IoT Linux Malware: A MITRE ATT&CK TTP Based Approach. In *APWG Symposium on eCrime*.

[Claise et al., 2008] Claise, B., Bryant, S., Leinen, S., Dietz, T., and Trammell, B. (2008). Specification of the ip flow information export protocol. *RFC 5101*.

[Dyrmishi et al., 2023] Dyrmishi, S., Ghamizi, S., Simonetto, T., Le Traon, Y., and Cordy, M. (2023). On the empirical effectiveness of unrealistic adversarial hardening against realistic adversarial attacks. In *IEEE Symp. on Security and Privacy (SP)*.

[Fairbanks et al., 2021] Fairbanks, J., Orbe, A., Patterson, C., Layne, J., Serra, E., and Scheepers, M. (2021). Identifying ATT&CK Tactics in Android Malware Control Flow Graph Through Graph Representation Learning and Interpretability. In *Proc. of IEEE Big Data*.

[Gibert et al., 2020] Gibert, D., Mateu, C., and Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.*, 153.

[Gorecki et al., 2011] Gorecki, C., Freiling, F. C., Kührer, M., and Holz, T. (2011). TrumanBox: Improving dynamic malware analysis by emulating the internet. *Lecture Notes in Computer Science*, 6976 LNCS:208–222.

[Grill et al., 2015] Grill, M., Nikolaev, I., Valeros, V., and Rehak, M. (2015). Detecting dga malware using netflow. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1304–1309. IEEE.

[Gunning et al., 2019] Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., and Yang, G. (2019). XAI - explainable artificial intelligence. *Sci. Robotics*.

[Hassan et al., 2020] Hassan, W. U., Bates, A., and Marino, D. (2020). Tactical provenance analysis for endpoint detection and response systems. In *IEEE Symp. on S&P*.

[Husari et al., 2017] Husari, G., Al-Shaer, E., Ahmed, M., Chu, B., and Niu, X. (2017). TTPDrill: Automatic and Accurate Extraction of Threat Actions From Unstructured Text of CTI Sources. In *Proc. of ACSAC*.

[Inacio and Trammell, 2010] Inacio, C. M. and Trammell, B. (2010). YAF: Yet another flowmeter. *LISA 2010 - 24th Large Installation System Administration Conference*.

[Kuechler et al., 2021] Kuechler, A., Mantovani, A., Han, Y., Bilge, L., and Balzarotti, D. (2021). Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes. In *Proc. of NDSS*, NDSS 21.

[Kuppa et al., 2019] Kuppa, A., Grzonkowski, S., Asghar, M. R., and Le-Khac, N. A. (2019). Finding rats in cats: Detecting stealthy attacks using group anomaly detection. *Proc. of IEEE TrustCom*.

[Kwon et al., 2019] Kwon, D., Kim, H., Kim, J., Suh, S., Kim, I., and Kim, K. (2019). A survey of deep learning-based network anomaly detection. *Clust. Comput.*

[Legoy et al., 2020] Legoy, V., Caselli, M., Seifert, C., and Peter, A. (2020). Automated Retrieval of ATT&CK Tactics and Techniques for Cyber Threat Reports. *preprint arXiv:2004.14322*.

[Malaiya et al., 2019] Malaiya, R., Kwon, D., Suh, S., Kim, H., Kim, I., and Kim, J. (2019). An empirical evaluation of deep learning for network anomaly detection. *IEEE Access*.

[MchPhee, 2020] MchPhee, M. (2020). Methods to Employ Zeek in Detecting MITRE ATT&CK Techniques. Technical report, SANS Institute.

[Mendsaikhan et al., 2020] Mendsaikhan, O., Hasegawa, H., Yamaguchi, Y., and Shimada, H. (2020). Automatic Mapping of Vulnerability Information to Adversary Techniques. In *The 14th Intl. Conf. on Emerging Security Info., Systems and Tech.*

[MITRE Corporation, 2020] MITRE Corporation (2020). BZAR: A set of Zeek scripts to detect ATT&CK techniques.

[Neo4j, 2022] Neo4j (2022). Neo4J Graph Database.

[Oosthoek and Doerr, 2019] Oosthoek, K. and Doerr, C. (2019). SoK: ATT&CK Techniques and Trends in Windows Malware. In *Security and Privacy in Comm. Netw.*

[Paxson, 1998] Paxson, V. (1998). Bro: A System for Detecting Network Intruders in Real-Time. In *7th USENIX Security Symposium (USENIX Security 98)*.

[Polychronakis et al., 2008] Polychronakis, M., Mavrommatis, P., and Provos, N. (2008). Ghost turns Zombie: Exploring the Life Cycle of Web-based Malware. *LEET 2008 - 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*.

[Project, 2022] Project, M. (2022). MalShare.

[Rafique and Caballero, 2013] Rafique, M. Z. and Caballero, J. (2013). FIRMA: Malware clustering and network signature generation with mixed network behaviors. LNCS 8145.

[Severi et al., 2018] Severi, G., Leek, T., and Dolan-Gavitt, B. (2018). Malrec: Compact Full-Trace Malware Recording for Retrospective Deep Analysis. In *Proc. of DIMVA*.

[Sharma et al., 2023] Sharma, Y., Birnbach, S., and Martinovic, I. (2023). RADAR: A TTP-based extensible, explainable, and effective system for network traffic analysis and malware detection. In Mileva, A., Wendzel, S., and Franqueira, V. N. L., editors, *Proc. EICC 2023*, pages 159–166. ACM.

[Shaukat et al., 2020] Shaukat, K., Luo, S., Varadharajan, V., Hameed, I. A., and Xu, M. (2020). A survey on machine learning techniques for cyber security in the last decade. *IEEE Access*, 8.

[SophosLabs Research Team, 2019] SophosLabs Research Team (2019). Emotet exposed: looking inside highly destructive malware. *Network Security*.

[VirusShare.com, 2022] VirusShare.com (2022). VirusShare.

[Wu et al., 2008] Wu, X., Kumar, V., et al. (2008). Top 10 algorithms in data mining. *Knowledge and information systems*.