A state machine system for insider threat detection

Haozhe Zhang, Ioannis Agrafiotis, Arnau Erola, Sadie Creese, and Michael Goldsmith

Department of Computer Science
University of Oxford, UK
first.last@cs.ox.ac.uk

Abstract. The risk from insider threats is rising significantly, yet the majority of organizations are ill-prepared to detect and mitigate them. Research has focused on providing rule-based detection systems or anomaly detection tools which use features indicative of malicious insider activity. In this paper we propose a system complimentary to the aforementioned approaches. Based on theoretical advances in describing attack patterns for insider activity, we design and validate a state-machine system that can effectively combine policies from rule-based systems and alerts from anomaly detection systems to create attack patterns that insiders follow to execute an attack. We validate the system in terms of effectiveness and scalability by applying it on ten synthetic scenarios. Our results show that the proposed system allows analysts to craft novel attack patterns and detect insider activity while requiring minimum computational time and memory.

Keywords: Insider threat · Tripwires · Attack patterns.

1 Introduction

There is growing evidence suggesting that organisations face significant risks from insider threats. According to the Breach Level Index 40% of the publicly reported data breaches were attributed to insiders who either maliciously or accidentally caused harm to their organisations [8]. In a similar vein, a survey conducted by ISACA [23] demonstrated that roughly 60% of the cyber-attacks which organisations experienced in 2014 were attributed to insiders threats. Beside the increase in the number of insider attacks, the inner knowledge and legitimate access to the systems, security practices and sensitive company data that insiders possess render these types of attacks the most costly [3, 20, 22], with reports suggesting that average damage can exceeded seven million dollars [14].

The dire implications, the increase in frequency, as well as challenges in detecting and mitigating these threats have attracted the interest of the research community over the last 20 years. Research has focused on conceptualising the problem of insider threat [10, 17, 19] and proposing anomaly detection

systems [15, 11, 12, 22]. On the other hand, large organisations and stakeholders involved in the defence and intelligence community have tried to mitigate the risk of insider threat by training investigators to determine when employees may become a risk and by forming transparent security policies [26, 1].

In this paper we propose a novel system which complements current approaches in insider threat detection and combines information from anomaly detection tools and security policies from rule-based systems. More specifically, our system provides a visual interface to security analysts that enables them to design and detect attack pattens which insiders follow. It utilises knowledge from theoretical works describing behaviours that may indicate insider activity [3], parses different types of logs to create attack graphs, provides alerts when an attack pattern is complete and outputs statistical data and visualisations that describe employees' activity on real time. We validate our system in ten synthetic scenarios and report our results regarding the effectiveness and efficiency of the detection system as well as its scalability.

In what follows, Section 2 reviews the literature on insider threat with focus on theoretical advances and implementions of anomaly detection system and Section 3 describes the system architecture and the methodology we followed to capture the requirements for our system. Section 4 presents the synthetic scenarios and discusses our results, while Section 5 concludes the paper.

2 Literature review

Literature on insider threat can be dichotomised in theoretical works aiming to understand behavioural factors, identify attack patterns and create conceptual models, and in practical research where a variety of anomaly detection tools with different capabilities are proposed. Carnegie Mellon University were the pioneers in examining human aspects of insider threat. In the CERT project, they applied System Dynamics to examine a series of insider threat case studies [10, 17]. Their proposed framework comprises four broad categories of insider cases, namely Information Technology(IT) Sabotage, Intellectual Property (IP) theft, Data and Financial Fraud, and Espionage. For each category, further behavioural aspects are identified and critical paths which insiders tend to follow are revealed. Their comprehensive work led to a a series of "Management and Education of the Risk of Insider Threat" (MERIT) studies where emphasis was placed on understanding how qualitative characteristics such as disgruntlement and dissatisfaction can be early indicators of insider threat activity [16].

Sarkar [24] provides a different perspective by distinguishing two classes of insiders, namely malicious and accidental. He identifies capability, motivation and opportunity as the three key factors which prompt insiders to act maliciously while human mistakes, errors, carelessness and bad design of the systems were the main characteristics of accidental insiders. In a similar vein, Nurse et al. examined a number of insider threat cases and developed a framework which describes not only technical or behavioural indicators but focuses on the motivation of the attackers as well [19]. Agrafiotis et al., analysed more than 100 cases

of insider activity, identified unique atomic steps which insiders follow and reconstructed all the cases based on these steps. They examined the common steps in these cases and effectively revealed more than ten different attack patterns which insiders followed to execute their attack [3]. These attack patterns along with publicly available security policies are formalised in [1], where a grammar for insider threat detection is proposed.

Another strand of literature emphasises on designing practical solutions to detect insider threats. Such works utilise different machine learning techniques to create profiles of employees based on their digital activity and try to distinguish abnormal actions [18, 5, 15]. Parveen et al. [21] focus on streaming data and use unsupervised learning techniques to identify changes in employees behaviour over time, a concept which they coined as "concept-drift". Chen et al. [11] proposed a user-relationship network for identifying collaborative insider attacks and introduced an unsupervised learning framework, the Community-based Anomaly Detection System (CADS). The underlying algorithm is a combination of kNN and PCA. Buford et al. [9] examined the concept of situation awareness in the automatic insider threat detection by designing an agent-based approach able to simulate insider behaviour and potentially detect changes in behaviour patterns Brdiczka et al. [7], explored the use of psychological profiling to reduce the number of false positive alerts in detection systems. Another interesting approach used Bayesian networks to infer the behavioural attributes of users based on sentiment analysis on text and social network analysis [4]. Lastly, [27] presents an unsupervised anomaly detection method using an ensemble of individual detectors to identify unknown attacks. Authors assert they can achieve or even improve the same performance of detectors that tackle specific scenarios.

Our review of the literature suggests that there is no unified effort to bring together the conceptual models which consider human aspects, the rule-based models which capture security-policy violations and the anomaly detection tools. A system proposed in [2] considers all these elements but has validated only the anomaly detection engine. In this paper we address this gap by designing a system which is able to capture information for attack patterns as presented in [3] using as a framework the grammar presented in [1] which can formalise policy violations and attack patterns which include alerts from anomaly detection systems. By effectively filling this gap, we enable analysts to design and test known or novel attack patterns, obtain a holistic perspective of employees' behaviours in real time and prevent insider attacks before being utilised.

3 System architecture and implementation

The detection system proposed in this paper is a state machine model and follows the tripwire grammar defined in [1], which is a formal language to clearly and unambiguously describe policy violations and attack patterns. Due to the fact that these violations can be triggered by a single system log and that no previous knowledge of the users profile is required, they are coined as *tripwires*.

An attack pattern P, as the Figure 1 shows, is a directed acyclic graph containing a finite number of *states* Σ and *transitions* Φ. Each transition is directed from one state to another, in such a way that there does not exist a consistently-directed sequence of transitions that starts from an arbitrary state σ and loops back to σ again. The states reflect the status of progress of P for a specific user and the transitions represent the attack steps that can be chosen from P. We use $S \in P$ and $\phi \in P$ to represent that the state S and the transition ϕ are in the pattern P.

Figure 1 shows an example of an attack pattern P describing an IP theft. The attack pattern captures the sequence of observed behaviour (observed behaviour is identified by parsing the necessary logs that organisations keep to monitor digital activity i.e., file logs, web logs, system logs within a specific time framework) and once complete (the S_4 state is reached) an alert is triggered. Formally, the attack pattern P is defined by the states $\Sigma_P = \{ S_0, S_1, S_2, S_3, S_4 \}$ and transitions $\Phi_P = \{ \phi_0, \phi_1, \phi_2, \phi_3, \phi_4 \}$ where S_i or ϕ_i represents a state or a transition with id i and belongs to attack pattern P. The S_{from}, S_{to}, *trigger* and *time* for any transition is defined below:

- $\phi_x : S_x \xrightarrow{<UserN, ActionY>, time} S_x + 1,$

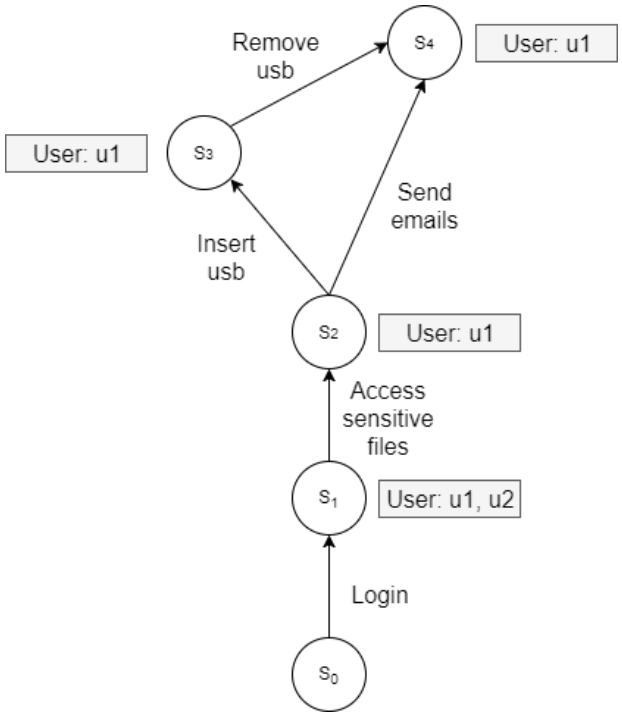


Fig. 1: The attacking progress of users u_1 and u_2

A key aspect of the transition that challenges the state machine model is the time within which a specific transition should be observed. The rationale behind being that an attack pattern must be executed within a specific time framework. It would be of no interest for example to observe a user accessing sensitive files and correlate this action to an email that is sent a month after. Once a completed attack is detected by the system, an alert is raised which comprises $\langle user, time, trace \rangle$; where *time* indicates when the attack happened and the trace is the sequence of attack steps committed by the user to complete the attack.

The proposed system allows analysts to create and edit attack patterns using an interface. It further enables analysts to match the organisation logs and alerts from detection systems to behaviours of interest as well as to monitor when there are accomplished by employees. The system parses the logs and tracks progress of all employees with respect to the created attack patterns in real-time and provides detailed statistics of the attacks through these interfaces. As Figure 2 shows, our detection system comprises two parts: the *front-end* and the *back-end*. The front-end refers to the presentation layer focusing on providing user-friendly interfaces, while the back-end refers to the data access layer which handles the logic of the detection system and storage of data.

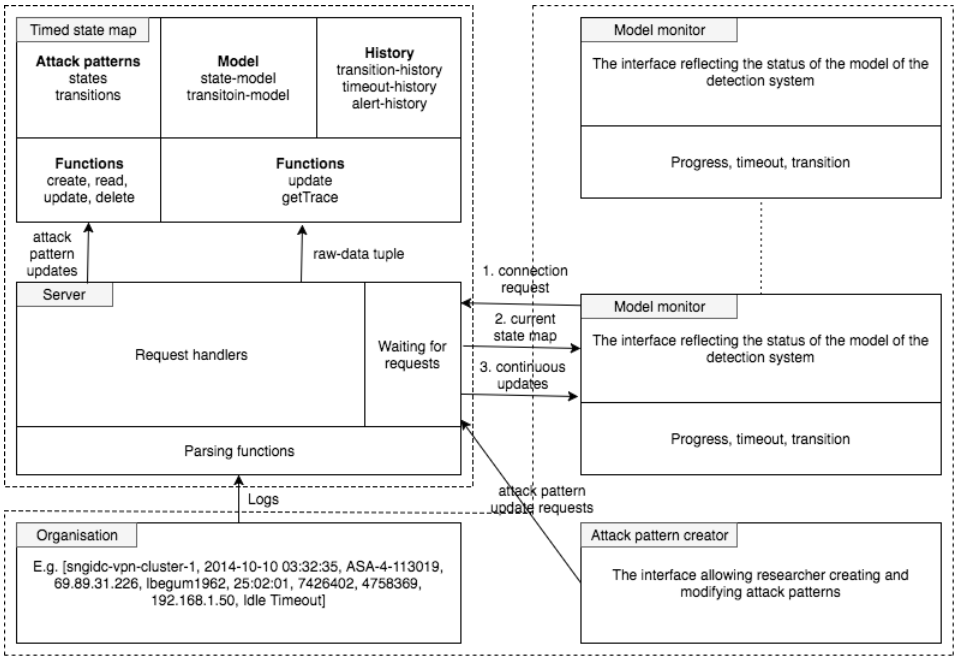


Fig. 2: The structure of state machine system

JavaScript is chosen as the primary programming language due to its highly increasing popularity for web based application developing. The community of JavaScript provides a powerful open-source libraries for building interfaces including D3.js [6] which is for creating dynamic, interactive data visualisations in browsers and React.js [13] which is for handling efficient updates of interfaces. The D3.js is applied for the construction of statistical charts in the interfaces, and the React.js is useful for the real-time update of the model monitor. The back end of detection system is developed using Node.js [25] which is an open source JavaScript run-time environment for executing JavaScript code server-side, while the front end interfaces are developed in HTML, CSS and JavaScript which can be executed by browsers.

We have designed the system with a modular approach meaning that the interfaces and subsystems are independent: they are encapsulated and have different duties. This was motivated by the fact that it is easier to modify the system to adapt to different datasets and attack patterns. We can modify or turn off one sub-system without changing others to meet different requirements. For example, when connecting a new organisation with a different log format, the system requires different parsing functions to translate the logs to the formatted inputs that can be recognised by the *timed state map*. The *timed state map* component maintains progress for attack patterns for every user. It is the subsystem server's duty to parse the logs, so we need to replace the old server with a new one which contains the specific parsing functions for the new organisation.

3.1 Main components of the system

The system comprises three sub-models which are *state-model*, *transition-model* and *history*. The state-model and transition-model are defined as set of tuples which are named *state tuples* and *transition tuples* respectively, or s-tuple and t-tuple for short. The *history* comprises different types of tuples recording the history of processing raw logs. Storing every possible instantiation of an attack pattern for every employee is not feasible, as will lead to a state explosion. We create a single instance for every attack pattern per user and we aggregate in their graph all the necessary information and present the most advanced state. All actions are recorded and once the time frame for observing a transition has elapsed, the system is able to back track to the second most advanced state based on the historical data. This is achieved with the use of s-tuples and t-tuples.

More specifically, an s-tuple is the record indicating the state a user is at and it contains the components *user*, *state* and *time*; where the *time* indicates the time the user reaches this state. A t-tuple is the record of a transition commenced by a user and it contains the components *user*, *transition* and *time*. All users are in the initial states for all the attack patterns by default. A transition ϕ can be triggered by a user when the $\phi.S_{from}$ has already been reached by the user, and the user is carrying out the $\phi.trigger$ and satisfies the $\phi.time$. We name the set of states reached by a user the *territory* of the user and users can expand their territories on the attack patterns by triggering attack steps leading from the states in their territory to unoccupied states. Figure 1 shows the attack progress

of users u_1 and u_2 with respect to the attack pattern P. The territory of u_1 is $\{ S_0, S_1, S_2, S_3, S_4 \}$ and the territory of u_2 is $\{ S_0, S_1 \}$.

The ***trace*** of an s-tuple s is defined as the sequence of history states the user $s.user$ has reached from the initial state S_0 to the $s.state$. For example, the trace of tuple $\langle u_1, S_4, time \rangle$ is $[S_0, S_1, S_2, S_3, S_4]$ and the trace of tuple $\langle u_2, S_1, time \rangle$ is $[S_0, S_1]$. We store the progress of a user in an attack pattern by keeping only one record for a state or transition in the attack pattern tree. Formally, this is ensured by the two *model-rules*:

1. at the certain time, a single state can be reached by a single user at most one time, i.e. there cannot be two s-tuples s_1 and s_2 such that $s_1.user = s_2.user \wedge s_1.state = s_2.state$.
2. at the certain time, a single transition can be commenced by a single user at most one time, i.e. there cannot be two t-tuples t_1 and t_2 such that $t_1.user = t_2.user \wedge t_1.S_{from} = t_2.S_{from} \wedge t_1.S_{to} = t_2.S_{to}$.

The history is used to keep track of the statistical information generated during the processing. This data is useful for the evaluation of the attack patterns and provides useful insights on common routes that attackers follow, allowing an analyst to act before the final steps of an attack are executed. The history of logs processed by the detection is shown in the *Processing Log* panel on the top left corner in Figure 3.

To update the model, every time the system parses a log it matches it to the territory of the *user* which is all the states the *user* is currently in. Then, the system calculates all the transitions that the *user* can trigger based on the current state; this set of transitions is named *candidate transitions*: $\Phi_{candidate}$. Since each transition refers to only one attack step, we only need to consider the transitions extended from the territory, which means the transitions directed from the states occupied by the *user*, i.e. $\Phi_{candidate} = \{ \phi \mid \phi \in \Phi, \phi.S_{from} \in S_{user} \}$.

Next, since the calculation of candidate transitions does not consider the triggering events and timeout constraints, we know $\Phi_{target} \subseteq \Phi_{candidate}$. So, we need to filter the $\Phi_{candidate}$ by applying the trigger and timeout constraints. Also, if a transition expires, we need to create a timeout event which represents a transition from the current state to S_0.

Once the set of target transitions is determined, we need to update the state-model, transition-model and history accordingly. For each timeout event $\phi_{timeout} \in \Phi_{target}$, we remove the s-tuple with value $\langle Tuple.user, \phi_{timeout}.S_{from} \rangle$ from the state-model which indicates that the user is no longer at the state and we add or update the timeout-history tuple with value $\langle Tuple.user, \phi_{timeout}, times \rangle$: if the tuple already exists, we set the initial *times* to one, else we increase *times* by one. Next, for each transition $\phi \in \Phi_{target}$, we add an s-tuple $\langle Tuple.user, \phi.S_{to}$ $, Tuple.time \rangle$ and a t-tuple $\langle Tuple.user, \phi.S_{from}, \phi.S_{to}, Tuple.time \rangle$ to the state-model and transition-model. Similar to the timeout transition, we add or update the transition-history tuple with value $\langle Tuple.user, \phi, times \rangle$. According to the

two model-rules introduced above, if there are s-tuples and t-tuples with the same $Tuple.user$, $\phi.S_{from}$ and $\phi.S_{to}$ values, we just update the time of the tuples without adding a new one. So, the model of the state map keeps only the latest behaviours of the users. Finally, when the output state is reached, i.e. $\phi_{S_{to}} \in \Sigma_f$, an alert-history tuple with value $\langle Tuple.user, trace, Tuple.time \rangle$ will be created and add to the history.

During the calculation of target transitions, all transitions should be commenced once the input raw-data tuple are parsed. However, there is a special case that there can be multiple transitions $\Phi_{duplicate}$ moving to the same state, i.e. $\Phi_{duplicate} \subseteq \Phi_{target}, s.t. \forall \phi_i, \phi_j \in \Phi_{duplicate}, \phi_i.S_{to} = \phi_j.S_{to}$. For this case, we remove other transitions and only leave a transition ϕ_{latest} which is directed from the state with the latest reaching time, i.e. $\phi_{latest} \in \{ \forall \phi_i \in \Phi_{duplicate}, \phi_{latest}.S_{from}.time \geq \phi_i.S_{from}.time \}$. Note the ϕ_{latest} is selected randomly from the transitions directed from the latest occupied state and this ensures there will be at most one transition directed to a state at one time.

Removing some of the transitions in $\Phi_{duplicate}$ will change the resulting t-tuples but the result of s-tuples would not be affected because the ϕ_{latest} who has the same target state with removed transitions is persistent: the target state would be reached anyway, which means the upcoming updates would not be affected. Commencing only the ϕ_{latest} ensures the trace of an s-tuple is deterministic (can be represented by an array) and trace follows the latest behaviours.

Visual Interfaces are the GUIs developed to allow researchers to access and manipulate the data of the state map, as well as to visualise statistics about attacks. Several visual interfaces with different functions are developed in this project.

The first visualisation of the interface is the *progress view*, which reflects the real-time progress of attacks for a selected attack pattern. Consider that the attack pattern *pattern2* is selected, as Figure 3 shows; the attack pattern is shown by a DAG on the right of the interface. The s-tuples are reflected by labelling the users beside the states that they belong to. The size of a state S in the DAG reflects the number of users at S, i.e. a larger state means there are more users at S. The history of logs processed by the detection is shown in the *Processing Log* panel on the top left corner.

As Figure 4 shows, the data for transitions are grouped by their transition id and presented by tables where vertical columns depict the number of times that the user commenced a transition. In addition, the interfaces provide a bar chart showing the comparison of frequencies of these transitions. The horizontal axis shows the transitions and different colours with different proportions in a bar refer to the number of times that the user commenced this transition. A DAG of a selected attack pattern is also provided, which reflects the frequencies of these transitions: the transitions with higher frequency have thicker links in the DAG. These views can reflect on design problems of the attack patterns, such as the interval time of a transition is too short or too long.

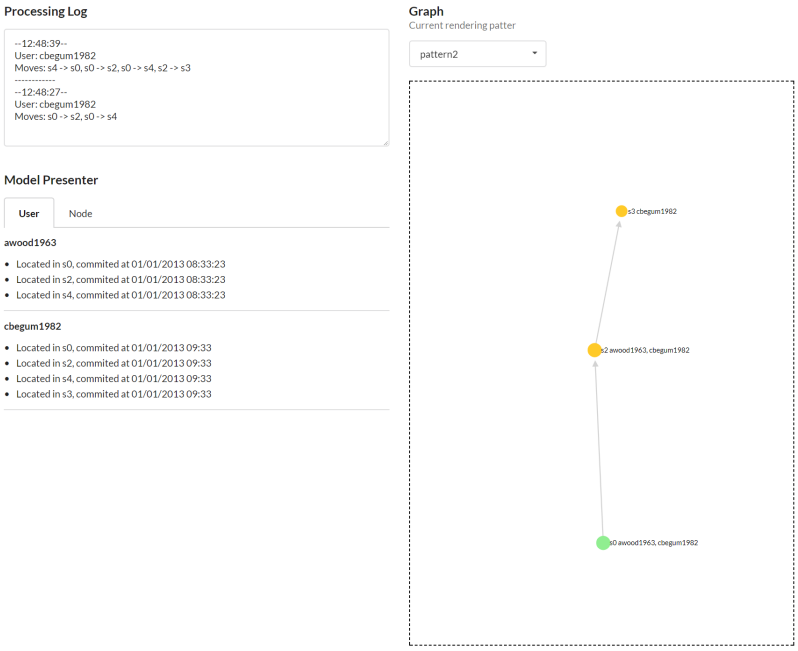


Fig. 3: The progress view of a model monitor

Alert view presents statistics on alerts that are flagged. With a similar interface as the transition view, the alerts are grouped by their traces and are represented in tables with columns named user and time as shown in Figure 5a. The interface also provides a bar chart indicating the number of these alerts. In addition, by comparing the number and composition of different groups of alerts, analysts can get statistical results such as "which trace of a given attack pattern is more frequent'" and "which employees are more likely to commit a given attack."

Pattern editor enables analysts to design and implement attack patterns in the state map, i.e. creating and modifying attack patterns through the buttons and input boxes without coding. The interface enables analysts to input the values for the attributes of Σ_P and Φ_P . These attributes are based on the data available and the system supports a predetermined number of attributes. The data that are currently supported are raw data from logs (file, web, email, login) and alerts from the CITD anomaly detection system. These alerts can be either unusual deviations from a normal behaviour or policy violations [2]. When special attributes are required it is straightforward to manually denote these into the parser module which will then update the pattern editor. Further details can be found in Appendix 1.C

Log importer sends logs to the server so it can be considered as the *organisation* part shown in Figure 2. As Figure 5b shows, it allows analysts to edit and send

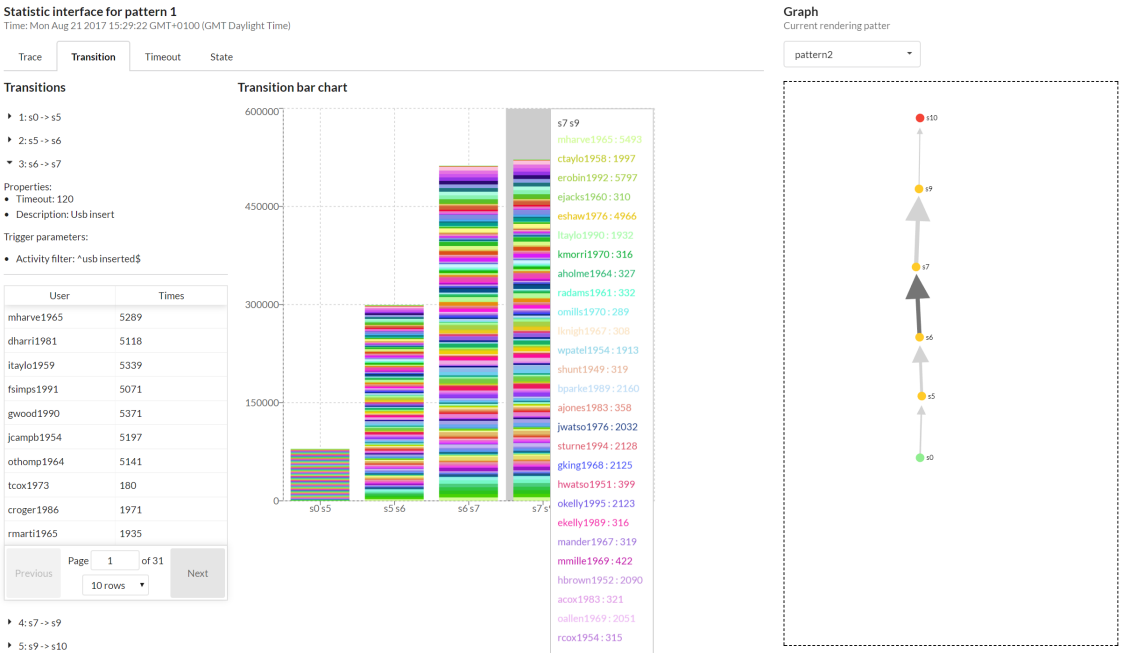
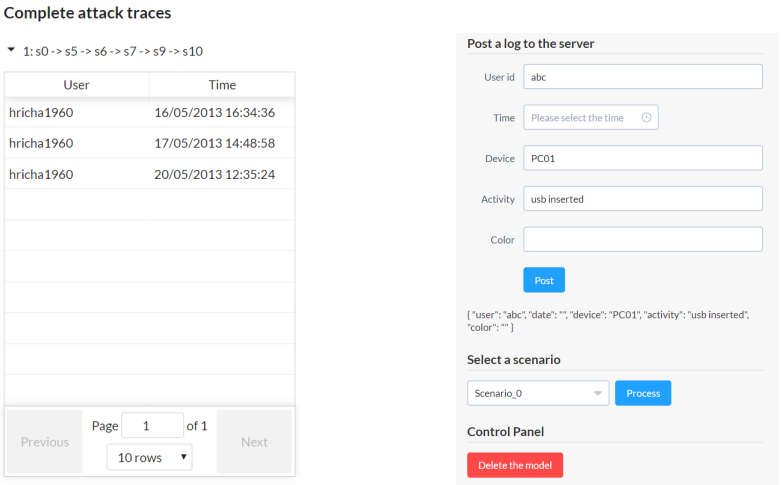


Fig. 4: The transition view of the model monitor



(a) The table in the alert view (b) Log importer

Fig. 5: The table in the alert view & Log importer

test logs to the system. The *server* of the system will consume the logs and start processing them.

4 System validation

We validate our system in terms of efficiency and scalability on ten synthetic scenarios. These scenarios were desinged as part of the Corporate Insider Threat Detection (CITD) project, which was sponsored by the UK National Cyber Security Programme in conjunction with the Centre for the Protection of National Infrastructure. The ground truth of these scenarios was made available to the authors only after all the results for all the scenarios was generated.

Each scenario contained login, file, website, email and usb activity logs for a period of one year (01/01/2013 - 31/12/2013), the number of employees varied from 12 to 300 and captured cases from IP theft to sabotage and financial gains. To evaluate the detection system, the attack patterns presented in [3] were implemented using the attack pattern editor interface. The system returned alerts on employees, indicating the paths of the attack which the employees followed and the time when the last step occurred. More details about the dataset for each scenario can be found in Appendix 1.A. The specific attack patterns with the time frameworks can be found in Appendix 1.B

4.1 Efficiency of the system

Due to space limitations, we will present in detail the results and visual outputs from one scenario and we will discuss the overall effectiveness of the tool on the rest of the scenarios in Section 5. The chosen scenario describes a disgruntled software developer who had recently being offered a position in a rival company. The developer had then used their company email address to send source code files to their own personal webmail address. There are particular folders of interest in the file log data which start from the path /svn or /development. Only one attack pattern was triggered when we run the system that pertained to IP data exfiltration via email and indicated the perpetrator of the attack. The single and correct output alert in this scenario was $nricha1989$, $19/12/201319:25:03$, $S0- > S1- > S2- > S3- > S5- > S6$.

Statistical data on transitions of all employees is shown in Figure 6. The data of transitions on the left side of the image enable analysts to select a specific transition to identify its characteristics and the number of times this transition occurred. The bar chart on the right shows on the x axis the transitions and on y axis the number of times this transition occurred. The bars are coloured differently and each colour represents different employees in the organisation. Analysts can hoover over a bar to elicit further details. The transparency of the colour is proportional to the number of times this employee has triggered the transition.

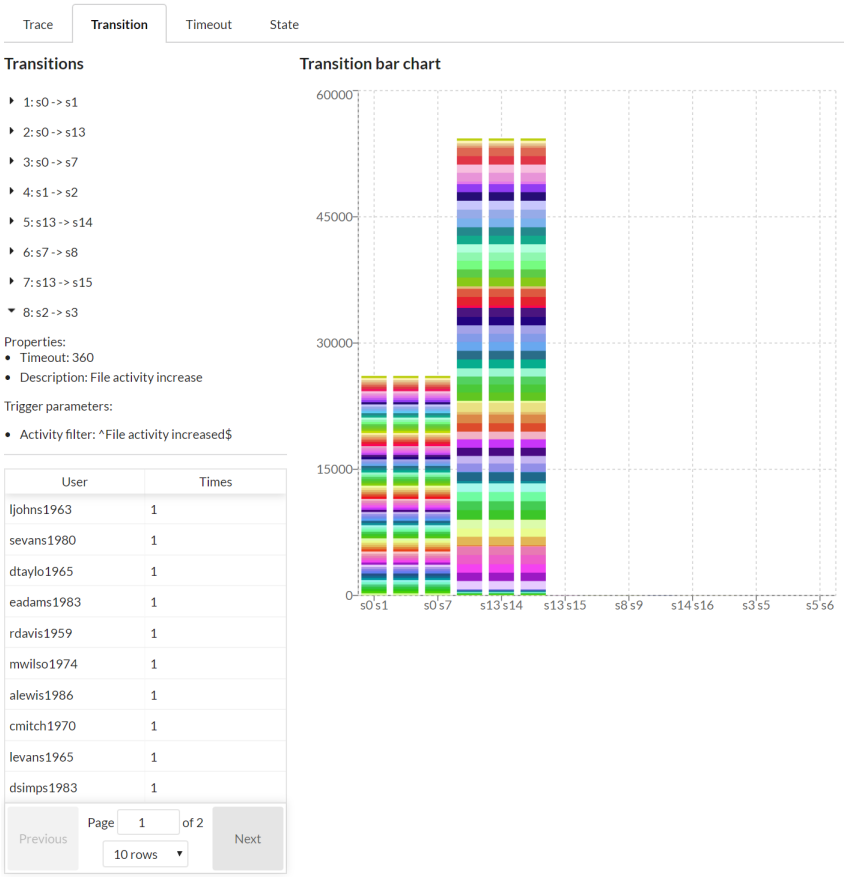


Fig. 6: The transition view of the scenario 1

From Figure 6 we can see that the vast majority (over 95%) of transitions refer to the first two steps of the attack pattern which are login and access to sensitive files. Due to the fact that the number of transitions for the next states is rather small, the colour indicating their frequency is not visible in the figure but the number be retrieved from the statistical data. It is expected to observe an overwhelming number of transitions for the first two steps and an equally overwhelming number of time out events in Figure 7 for these transitions. Employees login to the organisation's system to access files as part of their daily routine. They normally need to access multiple files, so the "sensitive file accessing" steps are more frequent than the "login" steps. The frequencies of the next steps in the progression of the attack pattern are much lower because the CITD system generated a small number of cmss alerts. The number of transitions to the next step is reduced to one. Only the insider cmss accessed an unusual

volume of files and sent emails with big attachments to their personal email address. This behaviour is denoted by the transitions cmss.

As Figure 7 shows, setting appropriate time frameworks for the transitions between states is of paramount importance. In our case, most of the transitions after the first two steps expired because the employees did not proceed further. The timeout events ensured that reasonably short intervals, e.g. in within a day, are required for an attack to be accomplished. Our results here highlight how valuable a state machine system can be in insider threat detection. An anomaly detection system would only provide indicators of abnormal behaviour and would probably increase the number of false positive alerts. Furthermore, these systems tend to be oblivious to the sequence with which certain actions take place. When alerts from such systems are combined with alerts from rule-based tools then behavioural aspects of an attack can be inferred as shown in our example. Furthermore, it is possible for an analyst to intervene before the final step of an attack is executed and mitigate the harms for an organisation.

Fig. 7: The timeout view of the scenario 1

Similar conclusions can be drawn for all the other scenarios, where our system has generated no false positives but failed to identify the perpetrator in three scenarios. In all cases the frequency of the steps decrease exponentially the further we proceed in the attack pattern.This is an indication that the attack patterns presented in [3] effectively indicate insider activity. Furthermore, our

system can complement an anomaly detections system as well as a rule-based tool and decrease the number of false positive alerts generated by these systems. On the other hand, the lack of an alert for three scenarios reveals that there is further work to be conducted in the design of attack patterns. The attacks in for the scenarios which the system did not generate an alert were subtle and the perpetrators were amongst the users who reached the higher states of certain attack patterns. We believe that the statistical data provided by our system can enable analysts to refine and tailor attack patterns to specific organisations. The key success criterion being that the more advanced the state is the less frequent its transitions should be.

4.2 Scalability of the system

Focusing on memory usage, the three components which are stored in memory are the state-model, the transition-model and the history. The sizes of the state and transition models are $O(|S|)$ and $O(|T|)$ respectively, where $|S|$ and $|T|$ are the number of states and transitions in the attack patterns. The complexity of the history is the sum of transition-history, alert-history and timeout-history. The size required by the alert-history is based on the number of logs. However, since the number of alerts raised by the detection system is fairly small, the memory required by the alert-history can be omitted. So, the space complexity of the history is $|transition - history| + |timeout - history| = O(2|T|)$. As the model maintains a behaviour profile for each user, by summing up the complexities calculated above, the space complexity of the detection system is $O(|U|(3|T| + |S|))$, where $|U|$ is the number of employees in the organisation.

The rather small complexity is due to the fact that the system does not create an instantiation of an attack pattern for every initial step for every employee. We have managed to trim the number of attack pattern instances per employee by updating the state in the attack pattern when someone is repeating previous steps to where he is and by treating effectively the timeout event.

The speed in computation depends on the the *update* function which performs two steps for each input log: 1) calculating the target transitions by selecting the transitions satisfying the input log and 2) updating the model concerning the target transitions. In worst case scenario, the target transitions are all the transitions in the detection system, so the complexity of the first step is $O(|T|)$. The second step is the update of the model including the update of state-model, transition-model and history. The time complexity is mainly related to the time required to access the specific tuple in the model. Since the state-model for a user is implemented as an array, the complexity is $O(|S|)$. The transition-model is implemented using a key-value pair and we can access the specific transition using the key in $O(1)$. Finally, to update the alert-history we add the new alert to the object, so the complexity for the alert-history is $O(1)$. Accessing a specific tuple in transition-history and timeout-history is $O(1)$, so the complexity of updating transition-history and timeout-history are same. Therefore, the update complexity for a log is $O(|T|)$, and the evaluation time for a dataset is $O(|T||D|)$, where $|D|$ is the number of logs in the dataset.

The design of the system interfaces considered scalability requirements. For the transition statistics, the records for transitions were grouped and represented by table of records rather than listed plainly, which significantly increased the readability of records. Also, the tool uses D3.js to implement the graphs and charts in the interface. It represents the graphs and charts by DOM manipulation. However, the performance of D3.js degrades significantly as the volume of data increases. In stress-testing datasets with large number of employees (more than 20000), the program ran out of memory. One solution for this is to render the graphs and charts in the back-end and present them in the browser. This could resolve the smoothness problem and the memory issue. However, the interactions would be disabled as the charts are static images.

5 Conclusions

The topic of insider threat has attracted the interested of the research community the last 20 years. A strand of literature has focused on proposing theoretical frameworks to conceptualise the problem whereas a different strand has emphasised on proposing anomaly detection systems. Organisations, in an attempt to mitigate risks from insider threats have developed security policies and rule-based systems to monitor for violations of these policies. In this paper we proposed a state machine system which complements all the aforementioned developments and combines data from detection systems and rule-based systems. Our system enables analysts to design and test attack patterns, incorporate data from anomaly detection or policy violation systems into these patterns and obtain a holistic understanding of the actions that employees perform. We have validated our approach on ten synthetic scenarios and by implementing the attack patterns provided in [3] we were able to detect the perpetrators in seven of these scenarios without generating false positive alerts. We have shown that our system is scalable in terms of computational time and memory usage, since the time complexity depends linearly on the size of the attack patterns and the memory complexity depends linearly on the product of the number of employees and the size of the attack patterns. Finally our system can provide real-time alerts for better situational awareness. Moving forward we intend to improve the performance of our system by adding a multi-threading module and deploy it it on a real organisation.

Appendix 1.A Dataset for every scenario

The detection system can have access to both raw data logs and alerts from anomaly detection systems. In our evaluation, these datasets contain both the organisation logs and the alerts generated by the CITD system which is in the same format with the other logs. Logs in each dataset are stored as cmss files, including

- cmss, which are the alerts generated by CITD system,
- cmss, which records the target addresses of emails sent by the users,
- cmss, which records the history of login and logout of the users,
- cmss, which contains the path of files accessed by the users,
- cmss, which records the URLs of websites accessed by the users,
- and cmss, containing the activity related to usb (inserted, removed).

This cmss data is composed by the attributes cmss, cmss, cmss and cmss which refer to the user's id, the time when this log is generated, the device's id and the behaviour of the user recorded by this log. An example of a row in the cmss is:

```
tellis1985,17/05/2013 14:49:11,PC025,http://sourceforge.net
```

Alert logs store the alert information in cmss and the severity of this alert in the extended attribute cmss which can be "Green", "Yellow" and "Red" as explained in [2]. An example of an alert data is:

```
{text}
    dricha1967,01/01/2013 06:39:19,PC060,Out of hours login,Red
```

Files for the same scenario are merged and sorted according to their cmss so we have a file for each scenario containing all the logs from oldest to newest. In addition to the logs, the information of employees and their occupation role duties is also provided for each dataset. This information can be further used in building novel attack patterns or refining current ones. For example, we may want to detect and add a step if any employee accesses sensitive files where admin is part of a path or a name of a file.

Appendix 1.B Attack patterns

Figure 8 shows the attack pattern which raised an alert for the scenario explained in detail in Section 4. The texts next to the transitions contain the ids of the original attack steps in [3] and brief descriptions of the implementation of the transitions. For example, the transition from S_0 to S_1 in Figure 8 refers to the attack step cmss that insiders login to the organisations' system using own credentials and this is implemented by capturing the logs in login system with value cmss.

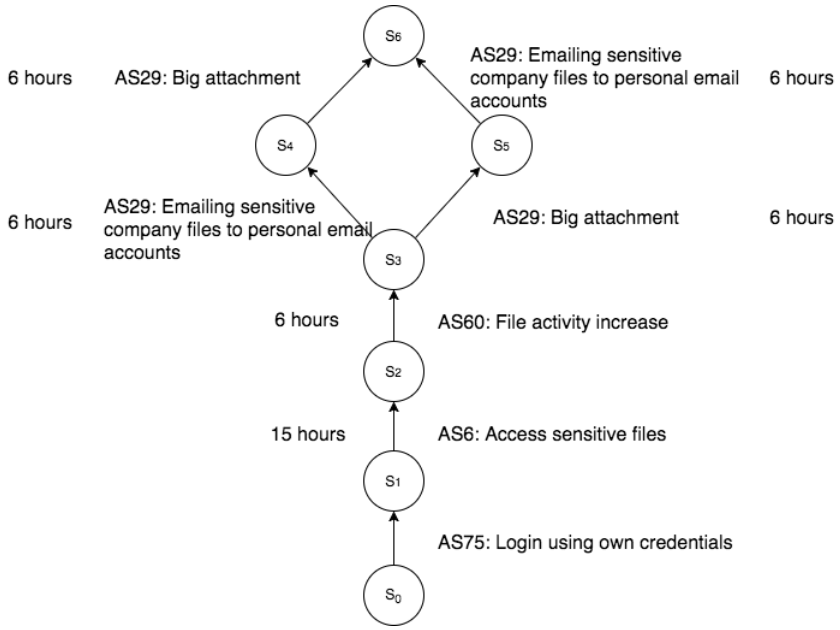


Fig. 8: The attack pattern which generated an alert in our scenario

Figure 9 illustrates the trace followed by the insider (which is highlighted) and the thickness of the arrows in the figure represents the frequency of the transitions.

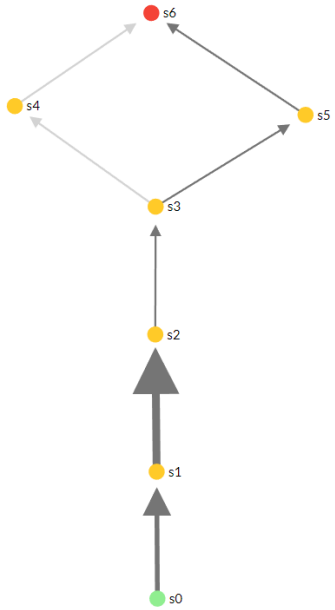


Fig. 9: The attack path which the insider followed in Scenario 1

Appendix 1.C Pattern Editor

Figure 10 presents the pattern editor interface and illustrates how analysts can straightforwardly design novel attack patterns without the need to change the code of the tool.

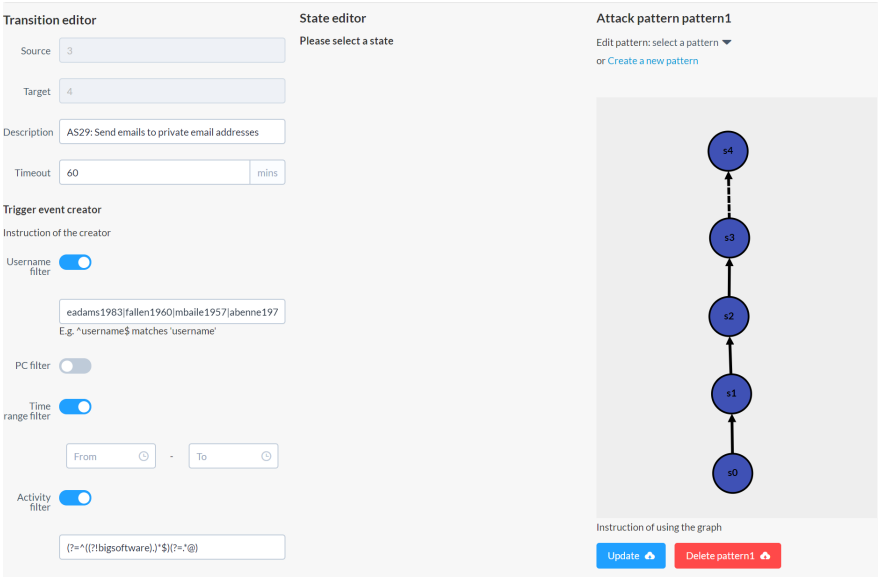


Fig. 10: The interface of the pattern editor

References

1. Agrafiotis, I., Erola, A., Goldsmith, M., Creese, S.: Formalising policies for insider-threat detection: A tripwire grammar. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA) 8(1), 26–43 (2017)
2. Agrafiotis, I., Erola, A., Happa, J., Goldsmith, M., Creese, S.: Validating an insider threat detection system: A real scenario perspective. In: Security and Privacy Workshops (SPW), 2016 IEEE. pp. 286–295. IEEE (2016)
3. Agrafiotis, I., Nurse, J.R., Buckley, O., Legg, P., Creese, S., Goldsmith, M.: Identifying attack patterns for insider threat detection. Computer Fraud & Security 2015(7), 9–17 (2015)
4. Arulampalam, M.S., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. IEEE Transactions on signal processing **50**(2), 174–188 (2002)
5. Bishop, M., Conboy, H.M., Phan, H., Simidchieva, B.I., Avrunin, G.S., Clarke, L.A., Osterweil, L.J., Peisert, S.: Insider threat identification by process analysis. In: Security and Privacy Workshops (SPW), 2014 IEEE. pp. 251–264. IEEE (2014)
6. Bostock, M.: D3. js. Data Driven Documents 492, 701 (2012)
7. Brdiczka, O., Liu, J., Price, B., Shen, J., Patil, A., Chow, R., Bart, E., Ducheneaut, N.: Proactive insider threat detection through graph learning and psychological context. In: Security and Privacy Workshops (SPW), 2012 IEEE Symposium on. pp. 142–149. IEEE (2012)
8. Breach, G.: In: level index—data breach database & risk assessment calculator (2016), http://www.breachlevelindex.com/
9. Buford, J.F., Lewis, L., Jakobson, G.: Insider threat detection using situation-aware mas. In: Information Fusion, 2008 11th International Conference on. pp. 1–8. IEEE (2008)

10. Cappelli, D.M., Moore, A.P., Trzeciak, R.F.: The CERT guide to insider threats: how to prevent, detect, and respond to information technology crimes (Theft, Sabotage, Fraud). Addison-Wesley (2012)
11. Chen, Y., Malin, B.: Detection of anomalous insiders in collaborative environments via relational analysis of access logs. In: Proceedings of the first ACM conference on Data and application security and privacy. pp. 63–74. ACM (2011)
12. Eberle, W., Graves, J., Holder, L.: Insider threat detection using a graph-based approach. Journal of Applied Security Research 6(1), 32–81 (2010)
13. Fedosejev, A.: React. js Essentials. Packt Publishing Ltd (2015)
14. Kingdom, H.P.E.U.: In: Ponemon cyber crime report: It, computer and internet security (2015), [Online]. Available: http://www8.hp.com/ uk/en/software-solutions/ponemon-cyber-security-report/
15. Magklaras, G., Furnell, S.: Insider threat prediction tool: Evaluating the probability of it misuse. Computers & Security 21(1), 62–73 (2001)
16. Moore, A.P., Cappelli, D., Caron, T.C., Shaw, E.D., Spooner, D., Trzeciak, R.F.: A preliminary model of insider theft of intellectual property (2011)
17. Moore, A.P., Cappelli, D.M., Trzeciak, R.F.: The big picture of insider it sabotage across us critical infrastructures. In: Insider Attack and Cyber Security, pp. 17–52. Springer (2008)
18. Myers, J., Grimaila, M.R., Mills, R.F.: Towards insider threat detection using web server logs. In: Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies. p. 54. ACM (2009)
19. Nurse, J.R., Buckley, O., Legg, P.A., Goldsmith, M., Creese, S., Wright, G.R., Whitty, M.: Understanding insider threat: A framework for characterising attacks. In: Security and Privacy Workshops (SPW), 2014 IEEE. pp. 214–228. IEEE (2014)
20. Nurse, J.R., Legg, P.A., Buckley, O., Agrafiotis, I., Wright, G., Whitty, M., Upton, D., Goldsmith, M., Creese, S.: A critical reflection on the threat from human insiders–its nature, industry perceptions, and detection approaches. In: International Conference on Human Aspects of Information Security, Privacy, and Trust. pp. 270–281. Springer (2014)
21. Parveen, P., Thuraisingham, B.: Unsupervised incremental sequence learning for insider threat detection. In: Intelligence and Security Informatics (ISI), 2012 IEEE International Conference on. pp. 141–143. IEEE (2012)
22. Rashid, T., Agrafiotis, I., Nurse, J.R.: A new take on detecting insider threats: Exploring the use of hidden markov models. In: Proceedings of the 2016 International Workshop on Managing Insider Security Threats. pp. 47–56. ACM (2016)
23. RSAConference, I..: In: State of cybersecurity: Implications for 2015 (2015), [Online]. Available: http://www.isaca.org/cyber/Documents/ State-of-Cybersecurity Res Eng 0415.pdf
24. Sarkar, K.R.: Assessing insider threats to information security using technical, behavioural and organisational measures. information security technical report 15(3), 112–133 (2010)
25. Tilkov, S., Vinoski, S.: Node. js: Using javascript to build high-performance network programs. IEEE Internet Computing 14(6), 80–83 (2010)
26. Upton, D.M., Creese, S.: The danger from within. Harvard business review 92(9), 94–101 (2014)
27. Young, W.T., Memory, A., Goldberg, H.G., Senator, T.E.: Detecting unknown insider threat scenarios. In: 2014 IEEE Security and Privacy Workshops. pp. 277–288 (May 2014). https://doi.org/10.1109/SPW.2014.42