A Tripwire Grammar for Insider Threat Detection

Ioannis Agrafiotis
Department of Computer Science
University of Oxford
ioannis.agrafiotis@cs.ox.ac.uk

Arnau Erola
Department of Computer Science
University of Oxford
arnau.erola@cs.ox.ac.uk

Michael Goldsmith
Department of Computer Science
University of Oxford
michael.goldsmith@cs.ox.ac.uk

Sadie Creese
Department of Computer Science
University of Oxford
sadie.creese@cs.ox.ac.uk

ABSTRACT

The threat from insiders is an ever-growing concern for organisations, and in recent years the harm that insiders pose has been widely demonstrated. This paper describes our recent work into how we might support insider threat detection when actions are taken which can be immediately determined as of concern because they fall into one of two categories: they violate a policy which is specifically crafted to describe behaviours that are highly likely to be of concern if they are exhibited, or they exhibit behaviours which follow a pattern of a known insider threat attack. In particular, we view these concerning actions as something that we can design and implement *tripwires* within a system to detect. We then orchestrate these tripwires in conjunction with an anomaly detection system and present an approach to formalising *tripwires* of both categories. Our intention being that by having a single framework for describing them, alongside a library of existing tripwires in use, we can provide the community of practitioners and researchers with the basis to document and evolve this common understanding of tripwires.

Keywords

Insider threat; Tripwire; Security policies; Attack-pattern; Grammar

1. INTRODUCTION

There is a growing concern that insider threats pose one of the most significant risks to organisations today. Recent reports concur that there is an increase in cases of insider activity [10], assuming that one can keep the threat out of an organisation is simply not a practical stance to adopt.

Addressing the risk from insider threat is not a new concept particularly for large organisations, who have for some time concerned themselves with personnel security. However, the practices that have evolved commonly are based around trained investigators determining when an employee may have become a threat. Such investigations need to be triggered; an event must be detected to raise an alarm and in most cases suspicious behaviour is reported either by other employees or by customers and partners.

Research suggests that transparent and well formed policies provide the foundation for a strong security culture [5]. It is not surprising that one of the first attempts organisations employed to mitigate the insider threat problem was the design of ad-hoc policies based on knowledge from insider incidents [2]. The dynamic legal and regulatory environment, the cultural differences in ethical policies, as well as the volatile threat landscape of insider activity requires these policies to be constantly under scrutiny and refinement, rendering the design of an unambiguous method for capturing and implementing these policies of paramount importance.

This paper describes how we intend to support insider threat detection when actions are taken which can be immediately determined as of concern because they fall into one of two categories: they violate a policy which is specifically crafted to describe behaviours that are highly likely to be of concern if they are exhibited, or they exhibit behaviours which follow a pattern of a known insider threat attack. In particular, we view these concerning actions as something that we can design and implement and define them as *tripwires* within a system to detect. We define insiders to be people with legitimate access to any system of the organisation.

Whilst there is already much sharing of experience and insight within the expert community, we believe that this can be substantially scaled by providing a framework within which this knowledge might be unambiguously captured, shared and used to directly generate tripwires or equivalent detection rules in users' technology of choice.

We present here the foundations for our capture of that understanding - an approach to formalising *tripwires* of both categories. Our intention being that by having a single framework for describing them, alongside a library of existing tripwires in use, we can provide the basis to document and evolve this common understanding of tripwires; a language in which we can develop a knowledge base that is unambiguous and can therefore be used to map experi-

MIST'16, October 28 2016, Vienna, Austria
© 2016 ACM. ISBN 978-1-4503-4571-2/16/10. . . $15.00
DOI: http://dx.doi.org/10.1145/2995959.2995971

ences regardless of the heterogeneity of the security tools and practices deployed.

In what follows, Section 2 reflects on methods focusing on formalising the insider threat problem and presents common policies identified for both types of tripwires which are used to validate our formal approach presented in Section 4. Finally, Section 5 concludes the paper and provides insights for future work.

2. FORMALISING THE INSIDER THREAT

Kammüller et al. [6] present one of the first attempts to formalise organisational policies for insider threats. They provide the language to formalise structural information and they try to establish a series of actions that would result in a system state where the negated formalised version of the policy is true.

In [3], researchers adopt a two-fold approach formalising a taxonomy of insider threats presented by Nurse et al [9] and estimating the likelihood of an employee conducting an insider attack. The approach is able to formalise concepts such as personal characteristics of an attacker and skills required to execute an attack.

Crampton et al. [4] adapt role-based access-control languages to the insider threat problem. The authors suggest the formalisation of access requests in the form of tuples, which contain contextual information (such as whether the user is accessing the system remotely). Policies either accept or deny these tuples.

Finally, Magklaras et al. make use of Domain Specific Language to construct an Insider Threat Prediction Specification Language [8]. The aim is to capture the misuses of IT policy violations, indicative of insider activity.

Reflecting on the relevant literature, all the formalisations are expressive enough to either formalise organisational policies in terms of access control or describe situations indicative of insider threat. There is not, however, a clear understanding of what data these formalisations require and how they may be used to provide a triggering event. It is this gap that this paper attempts to fill by providing a tripwire closely related to the detection system we have experimented with over the last two years, and applicable to all other systems capable of implementing tripwires.

3. TRIPWIRES IDENTIFIED FOR FORMALISATION

In this paper we are developing support for two types of tripwires: *policy violation* and *attack-pattern* tripwires. These two types of tripwires do not require anomaly detection techniques in order to fire; both are simply looking for specific behaviours and patterns directly presented in the data. This also means that they will not suffer from false-positive alerts, as they will only fire when it is certain that a behaviour of interest has been exhibited somewhere on the system. However, the degree to which the behaviour or events are indicate a significant threat will be open to interpretation by organisations. Since the nature of insider threats vary amongst different organisations (some organisations are prone to IP theft others to sabotage), so will the relevance and usefulness of the tripwires. Of course, as the threat landscape is constantly changing, so too will the usefulness of tripwires.

The policy violations capture tripwires which we have identified in publicly available documents and are widely applied in organisations. The attack-patterns capture tripwires on events of interest based on known insider attacks collected in [1].

The common policies we will use in validating the utility of our formalisation are:

- Physical impossibility (e.g. multiple VPN connections from geographically diverse IP addresses)
- Exceed threshold of emails containing attached files of significant size

Policies are of great interest as they can easily be flagged in near real-time. The output format of the L1 tripwire alerts in our system will be tuples in the form:

> ⟨User, Name of policy, Severity of alert[1], Number of violations in the current day⟩

Regarding tripwires based on attack-patterns which capture sequential actions of interest, we will use the most commonly identified patterns in [1]. The output format of the sequential actions in our system is:

> ⟨User, Name of attack-pattern tripwire, Severity of alert[1], Number of violations in the current day⟩

The attack pattern we have identified [1] and will use as an example is:

- Physical impossibility → Logons outside a set timeframe → Exceed threshold on deleted files

4. FORMALISING TRIPWIRES

Our formal language provides functions which transform data from different types of logs, features of the detection system and alert outputs in the form of tuples. These tuples provide information which is critical for the implementation of L1 alerts.

We define a set, *Raw_Data*, of functions for transforming raw data into tuples. More specifically, $Raw\_Data = \{f_{\mathit{email}},\ f_{\mathit{web}},\ f_{\mathit{file}},\ f_{\mathit{VPN}},\ f_{\mathit{log\text{-}on}},\ f_{\mathit{log\text{-}off}}, \ldots\}$. Each f in *Raw_Data* maps an individual log entry of the appropriate type into a valid *raw data* tuple.

In a similar vein, we define a set of functions named, $Features = \{f_{\mathit{1}},\ f_{\mathit{2}}, \ldots,\ f_{\mathit{n}}\}$, containing a function for each feature extracted by a detection system. Each f_n maps the value of the counter to an instance of the n^{th} class of features into a valid *feature* tuple.

Finally, we define a set of functions named, $F_{Alert} = \{f_{\mathsf{New\_email}}, f_{\mathsf{Email}}, f_{\mathsf{New\_file}}, f_{\mathsf{File}}, f_{\mathsf{New\_log\text{-}on}}, f_{\mathsf{Log\text{-}on}}, f_{\mathsf{User}}, f_{\mathsf{New\_Hourly}}, f_{\mathsf{Hourly}}, f_{\mathsf{New\_web}}, f_{\mathsf{Web}}, f_{\mathsf{New\_log\text{-}off}}, f_{\mathsf{Log\text{-}off}}, f_{\mathsf{Role}}, \ldots\}$. Each function f_X in F_{Alert} maps the data attached to an X-alert into a valid *alert tuple.*

A *raw-data tuple* is defined to contain five fields; ⟨*user*, *device*, *activity*, ⟨*key*: *metadata*⟩⟩; where:

- *user* $\in \boldsymbol{U}$, all the employees in an organisation
- *device* $\in \boldsymbol{D}$, all devices organisations might create logs for

[1] Yellow, orange, red based on thresholds defined by organisations

- *activity* $\in \boldsymbol{A}$ {file, log-on, log-off, VPN, email, web, insert, remove}

- $\langle$*key_data* $\in \boldsymbol{K}$: *metadata* $\in \boldsymbol{M}\rangle$, a sequence of attributes and their values

Attributes are drawn from a set of tags $\boldsymbol{G}$={Filename, URL, Email_to, Email_from, Source_IP,...} and values are drawn from a space $\boldsymbol{V}$=*Time* $\uplus$ *Size_of_file* $\uplus$ *Duration* $\uplus$ *IP* $\uplus$ *Size_of_attachement* $\uplus$ *Email_to* $\uplus$ *Email_from* $\uplus$ $Category\_for\_websites$ $\uplus$ ... In this set we include all the available types of data that might be tagged with an attribute.

It is evident that some tuples are not valid because the metadata values are not relevant for the activities they capture (i.e. we cannot have a file activity and an attribute Email_to attached to it). We therefore define a set of valid tuples $\boldsymbol{J}$.

For example, consider the raw VPN logs from the initial trial partner case study. These are of the form [VPN_name, date_utc, messsage_id, source_ip, userid, duration, bytes_sent, bytes_received, internal_ip, reason_for_disconnect]. Let us apply the f_{VPN} function to a data entry with the following value:

raw_vpn_1 = [sngidc-vpn-cluster-1, 2014-10-10 03:32:35, ASA-4-113019, 69.89.31.226, lbegum1962,
25:02:01, 7426402, 4758369, 192.168.1.50, Idle Timeout].

$f_{VPN}(raw\_vpn\_1) = \langle lbegum1962, \perp, VPN, \langle \mathsf{Source\_IP} : 69 : 89 : 31 : 226, \mathsf{Duration} :$
25:02:01, Date:2014-10-10 03:32:35$\rangle\rangle$, which is a valid tuple in $\boldsymbol{J}$.

Here, the device field is missing from the data logs. We obtain a set comprising tuples; we assign to the user named lbegum1962, a VPN activity which originated from a device with 69.89.31.226 as an IP address; the first tuple indicates that the activity lasted for 25 (hours):02 (minutes):01 (seconds) and the second tuple that it started at 2014-10-10 03:32:35.

Missing metadata fields are replaced with a $\perp$ symbol, representing *undefined*. In cases where missing values are crucial for an implementation of a tripwire, further triangulation of logs of different types will be required.

An anomaly detection system comprises of features fed to the machine learning approach [7]. Therefore, the *features* tuple is defined to contain three fields, namely the name of the user, the feature we are interested in and the value attached to the n^{th} feature. More formally, $\langle$*user*, n, value of feature $n\rangle$.

Finally, attack-pattern tripwires require access to alerts of detections systems to capture intermediate steps of unusual activity. We therefore define an alert tuple which contains three fields, namely the name of the user, the name of the alert being raised, the severity of the alert. More formally, *alert_tuple* = $\langle$*user*, *alert*, *severity*$\rangle$.

When we recognise that a policy violation occurred we raise the alert to the main detection engine, which increments its count. Write **Flag** $\langle$*user*, *alert*, *severity*$\rangle$, where *alert* is the name of the policy we are raising a flag for. Since the engine is keeping count of the alerts, when we observe a tuple, a count field will be included $\langle$*user*, *alert*, *severity*, *count*$\rangle$, where *alert* is the name of the tripwire we are raising a flag for and *count* is the number of **Flag** tuples the system has observed so far in the current day for the same tripwire.

4.1 Policy violation tripwires

This section provides the formalisations of the policies described in Section 3.

The physical impossibility

The first example of tripwire captures physical impossible VPN accesses (multiple VPN connections from different regional IP addresses). These types of violations may be indicative of stolen VPN credentials and elevation of access privileges. This formalisation is based on the initial trial partner datalogs for VPN. Suppose $\boldsymbol{K}$ contains $Source\_IP$, *Duration*, *Date*. Let $u \in \boldsymbol{U}$ and ip_1, $ip_2 \in \boldsymbol{V}$ and are values of the $Source\_IP$, $date_1$, $date_2 \in \boldsymbol{V}$ and are values of *Date* and $duration_1$, $duration_2 \in \boldsymbol{V}$ and are values of *Duration*. If we observe $\langle u, \perp, VPN, \langle$Source_IP:$ip_1$, Duration:$duration_1$, Date:$date_1\rangle\rangle$ and $\langle u, \perp, VPN, \langle$Source_IP: ip_2, Duration:$duration_2$, Date:$date_2\rangle\rangle$, such that $ip_1 \neq ip_2$ AND $date_1 \leq date_2$ AND $date_1 + duration_1 \geq date_2$ then **Flag** $\langle u, Physical\_impossibility, Red\rangle$ AND the system will increase this specific alert count. Notice that we do not assume that we have information about the name of the device, hence the $\perp$ wildcard symbol is used. If the VPN session has not been terminated, the duration value is ∞.

Emails with large attachment files

This example restricts the size of attachment files sent over email communication. Data exfiltration over email is a popular method followed by insiders. Our related tripwire formalisation below is based on the logs obtained from the initial trial partner.

Suppose $\boldsymbol{K}$ includes *Email_to*, *Attachment_size*, *Attachment_name* and *Date*. For $u \in \boldsymbol{U}$, *email_to*, *size*, *name* and *date* $\in \boldsymbol{V}$, if we observe $\langle u, \perp$, *email*, $\langle$Email_to:*email_to*, Attachment_size:*size*, Attachment_name:*name*, Date:*date*$\rangle\rangle$ and *size* $\geq 2GB$ then **Flag** $\langle u$, *Email_attachment*, *Red*$\rangle$.

4.2 Attack-pattern tripwires

Attack pattern-tripwires extend the grammar presented in Section 4.1 to allow us to maintain states of interesting behaviour for employees. We define a set of abstract states Σ, which reflects the state of progress of each attack pattern for a specific user. We also define a universal initial state S_0 as the starting point of all the attack patterns and a set of final states named Σ_f comprising of states (S_{fn}) denoting the end of an attack pattern. The initial and final states are part of Σ. We also define a set Σ_i to capture intermediate states, denoting situations where there are more than one activities required to reach the next state, the sequence of which is irrelevant.

Consider the example where an attack pattern requires 3 steps to completion with a single point as an outcome; there is an initial state S_0, a final state S_f, two states S_1 and S_2, and $\Sigma = \{S_0,\ S_1,\ S_2,\ S_f\}$. When we reach the final state we flag the alert with the appropriate data. This is captured as **Flag** $\langle User\ N, Name\ of\ pattern, Severity\rangle$.

We define a set Φ to denote valid transitions between states. A transition is a triple of the form $(S_n,\ t,\ S_{n+1})$, where $S_n \in \Sigma - \Sigma_f$, $S_{n+1} \in \Sigma$ and t is the triggering event which belongs to a set T of triggering events. The triggering event is a valid tuple in the $\boldsymbol{J}$ set or a *timeout* event. A timeout event is created when a specified time between two triggering events has passed, however the required event leading to the next state has not been observed. Time constraints between state-transitions will allow us to create windows of opportunity for an attack pattern to be executed and emphasise on situations of interest. Regarding transitions, each one comprises two attributes:

- *trigger* captures events required to commence a transition. In practice, this will be expressed by a pattern the tuple should match, which may contain values observed in previous tuples.
- *time* defines the time interval within which triggering events must be observed.

The following notation is used for the transitions:

$$\phi : S_n \xrightarrow{(trigger,\ time)} S_{n+1}$$

Next, we formalise one example of an attack pattern we have identified in our work [1]. As for the policy based tripwires, this set of examples is exhaustive is far from exhaustive and merely representative for the purpose of this paper.

In this attack pattern the insider logs on at an unusual hour and deletes an unusually large number of sensitive files. It is likely that the insider will try to obtain VPN credentials from a colleague to cover their tracks. Here, we provide an example that may have more than one variation, meaning that the insider may omit some steps. More specifically, we have the initial state S_0; transition to the final state S_f requires the user to exceed a threshold on deleting files; transition to the first step S_1 requires the observation of a violation of the physical impossibility alert; to reach the second state S_2, this pattern requires the observation of a logon activity outside a set timeframe. The transitions in the Φ set are built from:

- $\phi_0 : S_0 \xrightarrow{(\langle User\ N,\ Physical\ Impossibility\rangle,\ \perp)} S_1$
- $\phi_1 : S_0 \xrightarrow{(\langle User\ N,\ Logons\ outside\ timeframe\ \geq\ Orange\rangle,\ 48h)} S_2$
- $\phi_2 : S_1 \xrightarrow{(\langle User\ N,\ Logons\ outside\ timeframe\ \geq\ Orange\rangle,\ 24h)} S_2$
- $\phi_3 : S_1 \xrightarrow{(\langle User\ N,\ exceed\ deletion\ threshold\ \geq\ Orange\rangle,\ 24h)} S_f$
- $\phi_4 : \forall\ S_n\ \in\ \Sigma - S_0,\ S_n \xrightarrow{timeout} S_0$

Finally, $\Phi = \phi_0 \cup \phi_1 \cup \phi_2 \cup \phi_3 \cup \phi_4$. Note that there are two possible transitions from the initial state. One which leads to step one and an alternative which leads directly to step two. There is a 24 hour window to observe the VPN policy violation, a 24 hour window to observe the logon outside a timeframe activity, provided there was a VPN violation, and a 48 hour window to observe the same event from the initial state. Once the logon activity is observed there is a 24 hour window to detect an activity exceeding the threshold on deletion of files. The ϕ_4 transition ensures that, if there is a *timeout* event at any point of the attack pattern, we will return to the initial state.

5. CONCLUSIONS AND FUTURE WORK

This paper detailed our current research into how we support an anomaly detection system for insider threats with tripwires triggered when either a violation of a policy occurs or evidence of a known attack pattern is found. We provided a grammar able to capture policies which organisations have adopted in recent years as well as patterns of known insider threat behaviour. These tripwire formalisations are the basis for automatic code generation for tripwire alerts in our detection system. We believe that constructing such a grammar provides a framework for unambiguously collecting the body of knowledge and practice around insider threats, and it could support detection efforts in many operational contexts - not just in our prototype detection tool.

6. ACKNOWLEDGMENTS

This research was conducted for the Corporate Insider Threat Detection project, sponsored by the UK National Cyber Security Programme in conjunction with the Centre for the Protection of National Infrastructure, whose support is gratefully acknowledged.

7. REFERENCES

[1] I. Agrafiotis, J. R. Nurse, O. Buckley, P. Legg, S. Creese, and M. Goldsmith. Identifying attack patterns for insider threat detection. *Computer Fraud & Security*, 2015(7):9–17, 2015.

[2] D. M. Cappelli, A. P. Moore, and R. F. Trzeciak. *The CERT guide to insider threats: how to prevent, detect, and respond to information technology crimes (Theft, Sabotage, Fraud).* Addison-Wesley, 2012.

[3] T. Chen, F. Kammüller, I. Nemli, and C. W. Probst. A probabilistic analysis framework for malicious insider threats. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 178–189. Springer, 2015.

[4] J. Crampton and M. Huth. Detecting and countering insider threats: Can policy-based access control help. In *Proceedings of 5th International Workshop on Security and Trust Management*, 2009.

[5] C. for the Protection of National Infrastructure. Holistic management of employee risk. http://www.cpni.gov.uk/Documents/Publications/2012/2012021-homer.pdf, 2012. Last accessed: 2016-07-20.

[6] F. Kammüller and C. W. Probst. Invalidating policies using structural information. In *Security and Privacy Workshops (SPW), 2013 IEEE*, pages 76–81. IEEE, 2013.

[7] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese. Automated insider threat detection system using user and role-based profile assessment. 2015.

[8] G. Magklaras and S. Furnell. Insider threat specification as a threat mitigation technique. In *Insider Threats in Cyber Security*, pages 219–244. Springer, 2010.

[9] J. R. Nurse, O. Buckley, P. A. Legg, M. Goldsmith, S. Creese, G. R. Wright, and M. Whitty. Understanding insider threat: A framework for characterising attacks. In *Security and Privacy Workshops (SPW), 2014 IEEE*, pages 214–228. IEEE, 2014.

[10] H. Poll and A. Kellett. Vormetric insider threat report. https://www.sans.org/reading-room/whitepapers/monitoring/insider-threat-mitigation-guidance-36307, 2015. Last accessed: 2016-07-20.