TerminAPTor: Highlighting Advanced Persistent Threats through Information Flow Tracking

Guillaume Brogi, Valérie Viet Triem Tong

TerminAPTor: Highlighting Advanced Persistent Threats through Information Flow Tracking

Guillaume Brogi*†, Valérie Viet Triem Tong‡
* Akheros and † Conservatoire National des Arts et Métiers, Département IMATH, EA4629, Paris, France
guillaume.brogi@akheros.com
‡EPI CIDRE, CentraleSupelec, Inria, Université de Rennes 1, CNRS, IRISA, UMR 6074, Rennes, France
valerie.viettriemtong@centralesupelec.fr

***Abstract*—Long lived attack campaigns known as Advanced Persistent Threats (APTs) have emerged as a serious security risk. These attack campaigns are customised for their target and performed step by step during months on end. The major difficulty in detecting an APT is keeping track of the different steps logged over months of monitoring and linking them. In this article, we describe TerminAPTor, an APT detector which highlights links between the traces left by attackers in the monitored system during the different stages of an attack campaign. TerminAPTor tackles this challenge by resorting to Information Flow Tracking (IFT). Our main contribution is showing that IFT can be used to highlight APTs. Additionally, we describe a generic representation of APTs and validate our IFT-based APT detector.**

***Index Terms*—advanced persistent threat, chain of attacks, information flow tracking, intrusion detection**

I. Introduction

In recent years, Advanced Persistent Threats (APTs) have emerged as a new security risk for companies and governments. An APT is an attack campaign customised for its target and performed by highly skilled and highly motivated people. The attackers have precise goals and will do whatever it takes to achieve them, all the while avoiding detection. APTs have been recognised as a threat [1]–[3], but existing techniques are not able to detect them [4]. Research has started focusing on this problem [5], [6] but no perfect solution has emerged yet.

Even though each APT is tailored for its target and, thus potentially unique [7]–[10], their evolution usually follows a pattern well summarised in [7]. An APT starts with an initial reconnaissance step, followed by an initial compromise; once a foothold is established, the attacker will try to elevate their privileges and also move toward their target, which entails more internal reconnaissance and compromises; the final step, mission completion, is usually data exfiltration. We believe that this representation is flexible enough to represent every known APT and precise enough that it is the basis, in one form or another, of most of the research work on detecting APTs [11]–[14]. The hope is that by identifying the steps of an APT and by linking them together, then the APT itself can be detected. Most papers working on detecting APTs use the preceding observation of how an APT unfolds to base their approach on anomaly detection [15]–[18]. Since APTs are potentially unique, it is impossible to predict their exact behaviour and we feel that the phases described earlier are merely a guideline and want to avoid any set of hard rules based on their anticipated development.

This article focuses on the detection of long term attack campaigns. APTs are represented here as a succession of elementary attacks. These elementary attacks are detected by a standard intrusion detection system (IDS). We claim that detecting an APT is equivalent to highlighting the links between elementary attacks. For highlighting these links, we propose to track the information flows between the elements of the system used by the elementary attacks. To verify this claim, we have simulated two realistic APTs scenarios. All these experiments are realised using our product TerminAPTor.

The rest of the article is divided in five sections. In Sec. II, we present existing research pertaining to the detection of APTs and the use of information flow tracking for security purposes. In Sec. III, we present our characterisation of the monitored system and the attacker. In Sec. IV, we present TerminAPTor, the APT detector module, in details. In Sec. V, we present examples to show how, using information flow tracking, TerminAPTor can highlight APTs and to validate our approach. We also check that it can operate in near real-time. Lastly, Sec. VI concludes the paper.

II. Related works

In the following, we present how previous works have modelled the different stages of APTs. We also present the main characteristics of works about information flow tracking for security purposes.

In [19], Giura and Wang first present the main stages of an APT, which is basically the same model as in [12], [13], [20] or the one we present in this paper. Based on this model, they introduce the concept of attack pyramid, an extension of attack trees. An attack tree represents a threat, and is built recursively with the goal as the root of the tree and ways to reach that goal as children, who are then treated as new subgoals. A plane is the layer upon which a threat occurs, such as lockpicking in the physical plane or phishing on the user plane. The novelty of attack pyramids over attack trees is that paths can cross

planes; this better reflects what happens during APTs where attacks unfold over several planes. A set of rules is then used to correlate events and detect security alerts. When the risk and confidence factors of events and alerts rise above a given threshold, an APT alarm is triggered. This work is interesting because it specifically addresses the detection of APTs and presents attack pyramids to represent them, which takes into account the fact that an APT can operate on several planes. However, it requires that the potential goals of an APT, the paths leading there and the sets of rules be defined beforehand and updated frequently. This is specifically something that our approach wants to avoid, because it involves a lot of work to set up and maintain.

In [20], Sexton et al. decompose an APT into five phases, delivery, exploit, install, command and control (C&C), actions, which they call the attack chain. Contrary to the attack pyramid which starts from the goal and reconstructs possible attack paths, the attack chain is a model for APTs. This is more flexible in that there is no specified way to go from one phase to the next. Instead, each phase is associated with a number of event types indicative of this phase; events in different phases must then be combined before an APT can be detected. To combine them, each event is first assigned a score which is used to compute a score for each event type and then for each phase; an anomaly score is then computed for each host and each cluster of host where the same type of event is occurring. This work is interesting because it can highlight clusters being the target of an APTs. However, being based on anomaly detection, it requires that each possible event be listed and attributed to a phase of the APT model ahead of time.

Both of these approaches start from the same rough model of an APT in phases along its chain of attack and use it in different ways. This paper uses the same rough model to highlight APTs, but in a third way. However, both of these approaches require significant expert knowledge to set up and maintain. Our approach voluntarily distinguishes itself by eschewing the need for expert knowledge about the system and attacks. Instead, it requires tracking flows of information.

Information flow tracking refers to any method whose aim is to follow the evolution of pieces of information inside the observed system. For example, one application could be to follow sensitive information in the system in order to check that it isn't leaked on the network. To do this, an information flow tracking module has to be able to observe every flow of information and moreover to check if the flow involves a marked piece of information. This is usually done by adding a taint to the pieces of information to follow and then propagating that taint whenever a flow of information contains already tainted information. The main requirement of an information flow tracker is that every flow of information is indeed tracked.

There are two main kinds of approaches for information flow tracking. The first kind tracks the flows of information inside a single program while the second kind tracks all the flows inside a whole computer. Tracking the flows inside a single program is done either through binary instrumentation, like [21], or requires compilation of the source code with a special compiler, like [22]. Tracking flows inside a whole computer is usually done at the hardware level, thus requiring custom hardware, like [23], [24]. Deploying either approach is not very practical, requiring either custom hardware or modifying all the binaries on a system. An approach with the benefits of system-wide monitoring but without the cost associated with custom hardware is [25], where the authors modify qemu, a CPU emulator; this is an interesting compromise but cannot run on bare metal.

There is another option for system wide tracking without custom hardware or the need to modify every binary on the system: modify the OS to do the tracking. This is the approach adopted by [26], [27]. This way, the modification is software only and only needs to be done once. For example, in [27], the authors modify the Android OS in order to check that applications do not leak private data, and call it TaintDroid. They leverage the fact Android applications run on the Java Virtual Machine (JVM) and modify it to track information flows, even when the JVM interacts with the OS and during interprocess communications. Additionally, TaintDroid can track multiple sources of taints to better pinpoint the source of the data.

In this paper, we use a mix of kernel space and user space probes to track information flows. The flows are collected and then centralised. Only then are taints associated and propagated.

III. Characterisation

In this part we characterise both the monitored system and the attacker. We also list the requirements for the information flow tracking and enumerate the phases of an APT used in our model.

We propose here to protect enterprise networks from APTs. These networks are a mix of computer systems used as servers and others used as workstations. Each one of these computer systems is monitored by an agent. This agent has two roles; first it detects security violations and raises alerts, which can be done by classical intrusion detection systems; and second, it records information flows in the system and associates them with security alerts when appropriate. TerminAPTor then uses the output of the agent to check if separate security alerts are related and part of the same attack campaign. In the rest of the paper, we assume that the agent does detect all security alerts and only security alerts.

The attacker has a very precise goal and follows the steps of an APT: reconnaissance, compromise, establish presence, any number of lateral movements composed of the previous three steps but on the internal network and then mission completion. Except for the reconnaissance part, we suppose here that the attacker has to exploit an existing tool or deploy their own in order to progress from one step to the next. For example, in the compromise part, the attacker will send a spearphishing email with a document containing a payload. When the document is

opened, the payload installs a Remote Access Tool (RAT) which is then used in the establish presence phase. In any case, if TerminAPTor can show the links between the different steps all the way to the mission completion step at the end, then it will have highlighted the chain of attack used in this APT.

Thus, we assume that the attacker is capable of infiltrating any system they want to. To do so, they exploit existing tools or insert their own. Through the exploitation of these tools, the attacker goes from one step to the next; and these exploitations create flows of information that we track. One caveat is that the attacker does not want to be detected, so, for example, even though the attacker could become administrator and disable the monitoring agent, doing so would alert to their presence, thus, the attacker will not disable the monitoring agent or take any other actions leading to their being discovered.

In an attack campaign, each attack is part of a series of coordinated attacks. Thus, we define a *chain of attacks* as the list of attacks in an attack campaign, and we do not make a difference between an attack campaign and an APT. Each step in an attack campaign is an attack, and for a given attack, the attacks in the attack campaign which were used to set it up are called the direct ancestors and they are placed before the attack in the chain. More generally, ancestors of an attack are the attacks which precede it in the chain. Similarly, the direct descendants of an attack are the attacks which it sets up, and they are placed after it in the chain. We also generalise the concept of descendant to all the attacks after it in the chain.

IV. TerminAPTor, the APT detector

TerminAPTor acts over a supervision system composed of classical IDS which also record information flows. The IDS outputs data which are composed of *events* in chronological order, order which is assumed correct. The IDS also raises alerts whenever an *attack* is found. Each alert includes the list of events which triggered it. Thus, an attack is a list of events. Some event types can also be interpreted as *flows of information*. Every event interpreted as a flow of information is listed in Tab. I; each line in the table indicates that information is passed from the input objects to the output objects. For example, for the "Read file" event type, there is a flow of information from the file to the process reading it. As such, events are considered a tuple with four values: the event type, the timestamp, a list of references of input objects, and a list of references of output objects. If the lists of objects are empty, then the event cannot be interpreted as a flow of information. *Objects* are elements of the monitored system which either contain data (files and processes) or are communication channels (sockets). These flows are then processed in chronological order to perform information flow tracking.

Taking the list of events, whether they are part of an attack, and their associated information flows as our only known data, we aim to highlight chains of attacks. Our work is built on the intuition that if two attacks are consecutive steps in a chain of attacks, then there must be an information flow from the outputs of the events part of the first one to the inputs of events part of the second one. Since the first attack sets up the next, there should be an information flow from the first attack to the tools it uses and from these tools to the second attack. Let's take as example two attacks: the first attack installs a RAT, and the second uses that RAT to exfiltrate sensitive data. In this example, there is an information flow from the first attack to the RAT and then from the RAT to the second attack. Thus, if we want to link attacks in attack chains, we need to be able to differentiate the information flows depending on which attack they went through; the solution is to create a new *tag* to represent each newly detected attack. Tags are immutable and represent one and only one attack. They are attached to object through propagation. Whenever an object becomes tagged with a given tag, then it means that this object is part of an information flow coming from the attack represented by this tag.

TABLE I
List of events and their information flows

Type of event	Input objects	Output objects
Create process	current process	new process
Execute file	filename	current process
Change permissions	current process	filename
Change owner	current process	filename
Create file	current process	filename
Read file	filename	current process
Write file	current process	filename
Receive packet	remote socket	local socket current process
Send packet	local socket current process	remote socket

The tags representing each attack are propagated through the monitored system by flows of information, and the attack chains are highlighted at the same time. Each flow of information is processed in chronological order. The propagation is done differently whether the inputs are tagged and whether the event is part of an attack:

1) The inputs are not tagged and the event is not part of an attack: the event is ignored;
2) The inputs are tagged and the event is not part of an attack: the tags on the inputs are propagated to the outputs (Propagate);
3) The inputs are not tagged and the event is part of an attack: a new chain of attacks containing only this attack is added to the list of chains of attacks (CreateChain) and the tag representing the attack is propagated to the outputs (Propagate);
4) The inputs are tagged and the event is part of an attack: the attack is appended to the appropriate chains of attacks (AppendAndMergeChains) and the tag representing the attack is propagated to the outputs (Propagate).

The Propagate function merges the tags already present on the output with all the tags present on the inputs as

in (1); let **I** be the set of inputs and **O** the set of outputs for a given event:

$$\forall O \in \mathbf{O},\ \text{Tags}(O_{t_n}) = \text{Tags}(O_{t_{n-1}}) \cup \bigcup_{I \in \mathbf{I}} \text{Tags}(I_{t_{n-1}}) \quad (1)$$

When a new attack is detected, its links to other attacks can show that previously unrelated attacks are in fact related. The AppendAndMergeChains function must take this into account. Let L_{CA} be the list of chains of attacks (2). First, all the tags of the inputs of the new attack are put in a set, T (3). Then, each chain in L_{CA} finishing with a tag in T is moved to L_{new}, a new list (4). Then, each chain in L_{CA} containing at least one tag of T is copied; each copy is truncated after the last tag of T in it; the truncated copies are put in L_{new} (5). Then, all the chains in L_{new} are merged by creating a new chain with all the attacks in chronological order (6); the new attack is appended to this chain (7); This new chain is added to L_{CA} (8).

$$L_{CA} = \{A_1 - A_4, A_2 - A_5, A_3\} \quad (2)$$

$$\text{new attack: } A_6,\ T = \{2, 4\} \quad (3)$$

$$\begin{aligned} L_{CA} &= \{A_2 - A_5, A_3\} \\ L_{new} &= \{A_1 - A_4\} \end{aligned} \quad (4)$$

$$\begin{aligned} L_{CA} &= \{A_2 - A_5, A_3\} \\ L_{new} &= \{A_1 - A_4, A_2\} \end{aligned} \quad (5)$$

$$\begin{aligned} L_{CA} &= \{A_2 - A_5, A_3\} \\ L_{new} &= \{A_1 - A_2 - A_4\} \end{aligned} \quad (6)$$

$$\begin{aligned} L_{CA} &= \{A_2 - A_5, A_3\} \\ L_{new} &= \{A_1 - A_2 - A_4 - A_6\} \end{aligned} \quad (7)$$

$$L_{CA} = \{A_1 - A_2 - A_4 - A_6, A_2 - A_5, A_3\} \quad (8)$$

V. Examples of APT detections

The aim of these examples is to show that TerminAPTor is capable of highlighting whole attack campaigns. All the tests used here are reproducible and available in [28].

As a first example, we detail an APT targeted at a web server with a database. The attack campaign is composed of four attacks. Simultaneously, two other independent attacks also take place. For the duration of the scenario, standard web server usage is also simulated. In the attack campaign, the attacker starts by exploiting the shellshock vulnerability to install a Remote Access Tool (RAT) ($Attack_1$). Using the RAT, the attacker exploits a vulnerable application and starts a reverse shell with administration level rights ($Attack_2$). The attacker uses this shell to install a scheduled task which starts a second RAT also with administrator level rights ($Attack_3$). This second RAT is used to exfiltrate the website's database ($Attack_4$). While this is going on, two other attackers are active. The first one acts between $Attack_1$ and $Attack_2$, using an SQL injection to exfiltrate the database ($Attack_5$). The last attacker acts between $Attack_3$ and $Attack_4$, and uses the shellshock vulnerability to exfiltrate the list of users and passwords on the server ($Attack_6$).

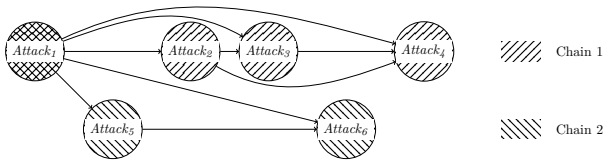

Fig. 1. Results of the information flow tracking on the server

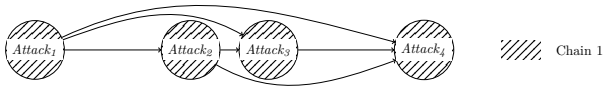

Fig. 2. Results of the information flow tracking on the workstation

The logs produced by the IDS are collected at the end of the scenario. First, all the attacks are detected and the events that triggered their detection identified. Then, TerminAPTor is launched to highlight links between attacks and extract the chains. The results are shown in Fig. 1. The first chain of attack accurately links the attack campaign we wanted to detect. This shows that TerminAPTor is capable of properly linking attacks part of the same attack campaign. This is very encouraging since some steps are executed only indirectly by the previous attack, e.g. $Attack_2$ starts $Attack_3$ through a scheduled task, but they are still properly linked. The second highlighted chain of attack is a false positive. TerminAPTor detects an attack campaign because all three attacks use the same entry point even though they are unrelated. This is because our flow tracking cannot follow information flows inside processes. Thus, the module has to taint every output of a process with the taints of all the inputs, which causes false positives.

The second example showcases an attack campaign targeted at a workstation. The attack campaign is composed of four attacks. The monitored system is a single machine used as a workstation. The attack campaign starts when the attacker sends a phishing email to the user of the workstation. The email contains an infected pdf file which is not detected by the various antivirus software protecting the workstation. When the user opens the file, a simple RAT is launched ($Attack_1$). The attacker uses this RAT to install a larger, more feature complete RAT ($Attack_2$). The attacker uses the second RAT to inspect the system and find a mean to elevate their privilege ($Attack_3$). With elevated privileges, the attacker installs a third RAT and enables persistence ($Attack_4$), so that the third RAT restarts with the workstation.

Like in the first scenario, the logs are collected at the end of the scenario This time TerminAPTor finds one chain of attack as shown in Fig. 2. This scenario is simpler since each step directly launches the next and, as expected, TerminAPTor properly follows the flows of information and links the four attacks in a single attack campaign.

While the results are encouraging, we believe that they could be improved upon by a second phase to filter out

false positives. In this regard, we are currently working on using our representation of APTs to create Markov models, based in part on the ideas in [29], [30]. Such models would be used to assess the quality of each link.

Additionally, our validation would benefit a lot from a public corpus of APT scenarios. Not only would this make validation more robust, but it enables comparing the results of different solutions. For example, it would have been very interesting to compare our results to those of the solutions presented in Sec.II.

Lastly, since the processing can be done on a separate machine, it has no impact on the monitored system. The only requirement is that it should be done in near real time. In order to test TerminAPTor on month long logs, we took the data from the APT against a server scenario and augmented it by mixing and mutating the original log as well as by inserting attacks. This produced a 67 day long log. In all, 3.2 billion events are processed, containing 500 attacks, over 67 days. This represents an average of 48 million events and 7.4 attacks per day. Processing the log on a single core took a bit less than 3 days and never exceeded $35MB$ in RAM usage. The aim of processing events in near real time is certainly met, especially since the code could certainly be optimised.

VI. Conclusion

This article proposes a new approach for detecting long lived attack campaign. The intuition leading to this work is that when an attacker constructs an APT step by step, each of these steps is a single attack. We claim that every single attack can be detected by a traditional intrusion detection system. Our main challenge was thus to be able to establish a link between these different attacks. In this article, we propose to retrace these links by observing information flows between traces left by the attacker during every single attack. We describe TerminAPTor, the tool being born from this idea. We present also some experiments as close as possible to real life APTs, using actual APT samples. These tests showed that our intuition is correct insofar that TerminAPTor accurately highlighted the chain of attacks targeting the monitored system.

References

[1] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, 2011.

[2] F. Li, A. Lai, and D. Ddl, "Evidence of advanced persistent threat: A case study of malware for political espionage," in *Malicious and Unwanted Software (MALWARE)*, 2011.

[3] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network security*, 2011.

[4] G. Ács Kurucz, Z. Balázs, B. Bencsáth, L. Buttyán, R. Kamarás, G. Molnár, and G. Vaspöri, "An independent test of apt attack detection appliances," https://blog.mrg-effitas.com/wp-content/uploads/2014/11/Crysys_MRG_APT_detection_test_2014.pdf, 2014.

[5] "Assessing outbound traffic to uncover advanced persistent threat," https://www.sans.edu/student-files/projects/JWP-Binde-McRee-OConnor.pdf, 2011.

[6] K. D. Bowers, M. Van Dijk, R. Griffin, A. Juels, A. Oprea, R. L. Rivest, and N. Triandopoulos, "Defending against the unknown enemy: Applying flipit to system security," in *Decision and Game Theory for Security*, 2012.

[7] "Apt1: Exposing one of china's cyber espionage units," http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf, 2013.

[8] "Apt30 and the mechanics of a long-running cyber espionage operation," https://www2.fireeye.com/rs/fireye/images/rpt-apt30.pdf, 2015.

[9] "Operation red october," http://ver007.com/tools/APTnotes/2013/McAfee_Labs_Threat_Advisory_Exploit_Operation_Red_Oct.pdf, 2013.

[10] "Safe: a target threat," http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-safe-a-targeted-threat.pdf, 2013.

[11] P. Bhatt, E. Toshiro Yano, and P. M. Gustavsson, "Towards a framework to detect multi-stage advanced persistent threats attacks," in *Service Oriented System Engineering (SOSE)*, 2014.

[12] J. De Vries, H. Hoogstraaten, J. van den Berg, and S. Daskapan, "Systems for detecting advanced persistent threats: A development roadmap using intelligent data analysis," in *Cyber Security (CyberSecurity)*, 2012.

[13] A. K. Sood and R. J. Enbody, "Targeted cyberattacks: a superset of advanced persistent threats," *Security & Privacy*, 2013.

[14] N. Virvilis, D. Gritzalis, and T. Apostolopoulos, "Trusted computing vs. advanced persistent threats: Can a defender win this game?" in *Ubiquitous Intelligence and Computing*, 2013.

[15] S. Chandran, P. Hrudya, and P. Poornachandran, "An efficient classification model for detecting advanced persistent threat," in *Advances in Computing, Communications and Informatics*, 2015.

[16] I. Friedberg, F. Skopik, G. Settanni, and R. Fiedler, "Combating advanced persistent threats: from network event correlation to incident detection," *Computers & Security*, 2015.

[17] A. Vance, "Flow based analysis of advanced persistent threats detecting targeted attacks in cloud computing," in *Infocommunications Science and Technology*, 2014.

[18] Y. Wang, Y. Wang, J. Liu, and Z. Huang, "A network gene-based framework for detecting advanced persistent threats," in *P2P, Parallel, Grid, Cloud and Internet Computing*, 2014.

[19] P. Giura and W. Wang, "A context-based detection framework for advanced persistent threats," in *Cyber Security*, 2012.

[20] J. Sexton, C. Storlie, and J. Neil, "Attack chain detection," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2015.

[21] F. Qin, C. Wang, Z. Li, H.-s. Kim, Y. Zhou, and Y. Wu, "Lift: A low-overhead practical information flow tracking system for detecting security attacks," in *Microarchitecture*, 2006.

[22] L. C. Lam and T.-c. Chiueh, "A general dynamic information flow tracking framework for security applications," in *Computer Security Applications Conference*, 2006.

[23] M. Dalton, H. Kannan, and C. Kozyrakis, "Raksha: a flexible information flow architecture for software security," in *ACM SIGARCH Computer Architecture News*, 2007.

[24] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure program execution via dynamic information flow tracking," in *ACM Sigplan Notices*, 2004.

[25] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *ACM conference on Computer and Communications Security*, 2007.

[26] R. Andriatsimandefitra and V. Viet Triem Tong, "Capturing android malware behaviour using system flow graph," in *Network and System Security*, 2014.

[27] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems*, 2014.

[28] G. Brogi, "moirai," https://github.com/akheros/moirai.

[29] G. Ioannou, P. Louvieris, N. Clewley, and G. Powell, "A markov multi-phase transferable belief model: an application for predicting data exfiltration apts," in *Information Fusion*, 2013.

[30] N. Ye, Y. Zhang, and C. M. Borror, "Robustness of the markov-chain model for cyber-attack detection," *Reliability, IEEE Transactions on*, 2004.