Design of an Anomaly-based Threat Detection & Explication System

Robert Luh[1,2], Sebastian Schrittwieser[1], Stefan Marschalek[1] and Helge Janicke[2]
[1]*Josef Ressel Center TARGET, St. Pölten University of Applied Sciences, Matthias Corvinus St. 15, St. Pölten, Austria*
[2]*Faculty of Technology, De Montfort University, Leicester, U.K.*
{*first.last*}*@fhstp.ac.at*

Keywords: Intrusion Detection, Malware, Anomaly, Behavioral Analysis, Knowledge Generation, Graph.

Abstract: Current signature-based malware detection systems are heavily reliant on fixed patterns that struggle with unknown or evasive applications, while behavior-based solutions usually leave most of the interpretative work to a human analyst. In this paper, we propose a system able to explain anomalous behavior within a user session by considering anomalies identified through their deviation from a set of baseline process graphs. To minimize computational requirements we adapt star structures, a bipartite representation used to approximate the edit distance between two graphs. Baseline templates are generated automatically and adapt to the nature of the respective process. We prototypically implement smart anomaly explication through a number of competency questions derived and evaluated using the decision tree algorithm. The determined key factors are ultimately mapped to a dedicated APT attack stage ontology that considers actions, actors, as well as target assets.

1 INTRODUCTION

IT infrastructures are threatened by an ever-growing number of different cyber-attacks. With the emergence of Advanced Persistent Threats (APTs), the focus shifted from off-the-shelf malware to attacks tailored to one specific entity. While APTs use malware like most conventional attacks, the level of complexity and sophistication is usually much higher. This is problematic especially since defensive measures offered by security vendors often utilize the same detection approaches that have been used for many years – with mixed results. The major drawback of these primarily signature-based systems is that the binary patterns required for detection are unlikely to exist at the time of attack, as most APTs are tailored to one specific target. In addition, packers, meta- and polymorphic techniques are employed to throw off signature-based systems while the multi-stage nature of APTs makes it generally difficult to interpret findings out of context (Luh et al., 2016a). This increased complexity makes it necessary to explore novel techniques for threat intelligence and malicious activity detection on multiple layers.

Behavior-based approaches are a promising means to identify illegal actions. No matter the stealth techniques employed, the attacker will sooner or later execute his or her action on target – be it data theft or sabotage. Anomalies signifying a deviation from a known behavioral baseline can then be used to detect the threat. However, most existing systems do not disseminate the offending behavioral data and contribute little to aid in their interpretation. We argue that closing the resulting semantic gap is a vital next step in holistic IT system threat mitigation.

To achieve this goal, we introduce a system capable of dealing with a variety of targeted attack scenarios. Specifically, we contribute by presenting a holistic approach to collecting and correlating host and network events able to describe all APT attack stages coupled with a transparent anomaly detection and baseline graph extraction system based on star structures and edit distance computation. Anomalies are disseminated and interpreted using a semantic decision tree approach in combination with a comprehensive targeted attack ontology. This position paper presents an overview of this approach.

Related work – Anomaly-based malware or intrusion detection systems are found in many proposed solutions. However, it is rare to see it combined with a semantic component that is dedicated to the automated interpretation of the generated traces, logs or alerts. Noble and Cook (2003) explore graph-based anomaly detection through the detection of repetitive substructures within graphs as well as by determining which subgraph of interest consists of the highest number of unique substructures. The introduced system is also able to measure the regularity of a graph using con-

Luh, R., Schrittwieser, S., Marschalek, S. and Janicke, H.
Design of an Anomaly-based Threat Detection & Explication System.
DOI: 10.5220/0006205203970402
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 397-402
ISBN: 978-989-758-209-7

ditional entropy. Being mostly formal in nature, the approach does not consider attack semantics. Most other graph-based systems for intrusion detection scenarios discuss attack graphs, which put the focus on vulnerability analysis and the sequence of events leading to a state of compromise (Sheyner et al., 2002).

Dolgikh et al. (2012) conduct dynamic behavioral analysis of applications capable of automatically creating application profiles for both malicious and benign samples. Their system considers recorded API calls that are subsequently transformed into a labeled graph representing a stream of system calls. Graphs are compressed using a genetic data processing algorithm in order to extract behavioral profiles.

On the network traffic side, Münz and Carle (2007) present TOPAS, a traffic flow and packet analysis system compatible with NetFlow and IPFIX. The system's detection algorithm uses threshold-based detection via pre-defined values as well as outlier detection through the comparison of a sample to previously learned, normal behavior. While this offers a good foundation for traffic anomaly detection, the link to local processes and applications is not investigated.

The shift of focus towards semantic awareness is visible in several, more general works. For example, Anagnostopoulos et al. (2005) present a system for the application of semantics to general intrusion scenarios. The authors seek to classify and predict attacker intentions using a Bayesian classifier and a probabilistic inference algorithm. Their semantic model includes both legitimate and illegitimate actors, activities in the form of sequential events, concrete commands issued, and an overall state of attack.

As above works exemplify, none of the solutions quite manage to bridge the gap between system events (be they function calls or traffic flows) and a truly meaningful representation of an attack in its entirety. Closing this semantic gap is one of the main goals of the system presented in this paper.

2 REQUIREMENTS

There are several formal, semantic, and strategic requirements that have to be met before the technical implementation can be approached. The design of the system is based on the roadmap for a conceptual APT defense system introduced in a survey by Luh et al. (2016a). In order to fulfill the requirements for the comprehensive detection and analysis of targeted attacks we followed the authors' system capabilities checklist and extended the design with the ability to explain detected anomalies in behavioral data.

2.1 Threat Definition and Modeling

For threat definition, we decided to use the cyber kill chain model by Hutchins et al. (2011). In order to combine and interpret data sources and the events they generate, we constructed a top-down view on a potential targeted attack by combining a streamlined version of the model with the construction of an APT ontology first introduced in Luh et al. (2016b).

Ontologies are a promising way of approaching the challenge of formalizing threats and threat responses in a semantics-aware manner. Originally a discipline of philosophy, ontologies in information science have become an accepted formal approach to describing data types, properties, and interrelationships of entities within a specific domain. Their reasoning capabilities and data formats set them apart from semantics-unaware relational databases. Depending on general requirements and desired granularity, system designers can choose from numerous languages or systems. Several semantics-based works (see survey by Luh et al. (2016a)) successfully utilize ontologies as part of their threat mitigation approach.

In preparation for the technical system design introduced below, we used Protégé to create aforementioned attack ontology. For concrete threat actions (including assets and actors), we used a goal modeling approach based on GRL. Attack specifics were derived and expanded from a formal definition of malware behavior introduced by Dornhackl et al. (2014) as well as from the aforementioned kill chain model.

2.2 Data Provider Selection

Following the definition of the ATP ontology and its inherent rules, we linked each attack stage and activity to specific system events or, as discussed below, to specific behavioral anomalies. In order to retrieve event data that depicts a wide range of possible attacker actions, it is prudent to choose data providers capable of collecting both host-based as well as network-based information. To be in line with this requirement, our system design is based upon the use of kernel event traces and network flow information. The information is processed to gather intelligence on the attack, correlate the two classes of data providers, detect an attack through an anomaly-based approach, and analyze the outcome to extract semantically relevant information. In combination, the system is able to counter the threats of malware, host intrusions in general, as well as several classes of network attacks. Mapped onto the kill chain model, our proposed design is capable of processing reconnaissance, delivery, exploitation, installation, command &

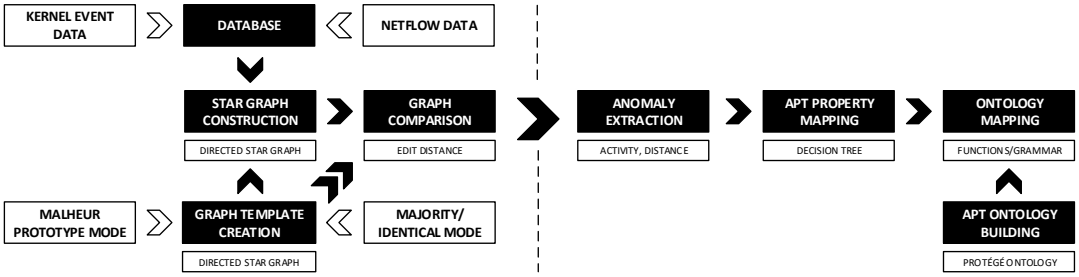

Figure 1: System design overview.

control, as well as action on target activity. We also formally checked our technical design against Luh et al.'s system design checklist (Luh et al., 2016a) in order to confirm aforementioned capabilities.

3 SYSTEM DESIGN

The proposed system is composed of several components enabling the underlying anomaly detection and knowledge explication process. The initial tasks encompass the acquisition of data from a number of monitored devices as well as the transmission and translation of kernel events to a clean database format. Afterwards, we link events by their contextual parent and construct traces in the form of star structures, simple graphs that describe the operations conducted by each process within a specific time range. From a baseline of benign system behavior we then extract one or several process-unique templates that are subsequently used to check new activity for anomalies by measuring the edit distance between the graphs.

Our approach not only calculates deviations but also returns a list of actions that constitute the identified anomaly. Combined with an APT property extraction routine based on decision trees, we are ultimately able to map each behavioral pattern to aforementioned ontology.

3.1 Data Collection

The proposed system is based on two data types collected on both the hosts as well as on a network device connecting these endpoints:

Event traces are typically defined as descriptions of OS kernel behavior invoked by applications and, by extension, a legitimate or illegitimate user. More often than not, these events are abstractions of raw system and API calls (e.g. `CreateProcess`) that yield information about the general behavior of a sample (Wagner et al., 2015). In the context of our system, event data is collected directly from the Windows kernel. This gives us unimpeded access to events depicting operations related to process and thread control, image loads, file management, registry modification, network socket interaction, and more.

Network traffic analysis, on the other hand, comes in two distinct flavors (Sperotto et al., 2010): Packet inspection, where the payload of certain (or all) packets is analyzed, and flow-based detection systems. The latter focus on communication patterns instead of individual packets. Such patterns typically include source and destination IP addresses, port numbers, transmission times, as well as the amount of data and number of packets sent. Our system correlates network flows captured at a central switch or border gateway with local network events captured at each monitored host. This way we are able to link specific processes to conversations over the network – be it a host-to-host exchange or the interaction with remote Internet resources.

The relative ease of monitoring as well as the semantic expressiveness of kernel events and network flows make them ideal for dynamic smalware analysis and application classification. The design introduced in this paper uses this rich repository of behavioral data to compile a graph-like star structure of event sequences that can describe not only a single application, but also a system session as a whole. This process is detailed in the following:

Event linking: All previously collected events are subsequently linked through their parent process in order to establish a semantic connection between action and cause. This is realized through 3 attributes that are present in all the data collected by the host monitoring agent: Creation time, and the PID that forms a unique identifier for each process. Threads work in a similar fashion. Like PIDs, thread IDs (TIDs) are logged by other event types (e.g. registry events) and can therefore also be used for event linking. It becomes possible to reconstruct the entire event flow (tree) or determine specific dependencies

between processes or general events. Concatenated into a full system graph, the sequence of events of the monitored session can be assembled. In a follow-up step, these graphs are reduced to star structures centered around a process triggering a specific event.

Star construction: In subgraph search, *star structures* are a means to reduce the complexity of a known NP problem to polynomial complexity (Hu et al., 2009). Instead of searching entire system session graphs for matching patterns, the star structure approach breaks down the computation into a triplet of vertices connected by a labeled edge, denoted as $G = (U, V, E)$, where U and V are nodes and E is the respective edge. The edge label is used as basis for minimal cost calculation of same-size star structures. Specifically, it utilizes bipartite graph matching based on the Hungarian algorithm (Kuhn, 1955), where every star is processed as a matrix. Graph edit distance calculation determines the minimal costs of relabeling the nodes and edges of a graph G to match a graph H. The edit path $P_{G,H}$ can be understood as a sequence of transformation operations σ. The final graph edit distance is determined by the cheapest of all edit paths between G and H.

Compared to full graph matching, this approach is typically a faster, but less precise approximation, as it only matches the immediate neighborhood of one node at a time. In our system, we use an adaptation of Hu et al. (2009) approach that combines n bipartite graphs into one star representing a single process. This makes the effect on result accuracy far less pronounced: With a focus on individual processes, our input data can already be reduced to star structures without significantly compromising trace semantics. This is due to the fact that we anchor every event to a trigger (parent) process (see Section 3.1) that actively invokes respective actions, making this process the natural center vertex of a star-shaped graph. In our system, elemental operations for determining the minimal cost graph edit distance between individual elements are not limited to relabeling nodes, but consider edges as well. *Vertex edit operations* encompass single vertex relabeling σ_{RV} operations as well as both an insert vertex operation σ_{IV} and a delete vertex operation σ_{DV}. Semantically speaking, each vertex is akin to a system event (event type plus parameter) as introduced in Section 3.1. *Edge edit operations*, on the other hand, primarily consider the edge relabeling cost σ_{RE}. We opted to dynamically assign individual relabel costs based on the type of event considered, making the approach fully capable of assessing event similarities. Specifically, σ_{*V} will drastically increase in cost when converting a semantically expensive process event to a relatively low-impact registry event. At the same time, the type of operation (numerical representations of create, modify, delete, start, and stop operations) considered by σ_{RE} determines the final cost of edge relabeling.

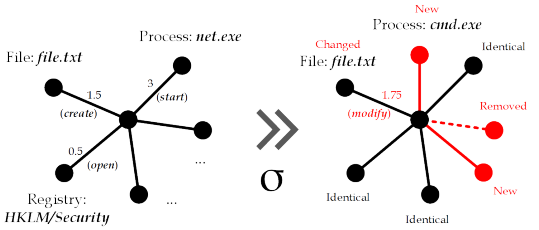

Figure 2: Example transformation of baseline (left) to target graph (right) for process `svchost.exe`.

Figure 2 shows a simplified example. In the depicted case, the base graph consists of various vertices representing events such as the creation of a file, the start of a process and an open/read operation conducted in the `HKLM/Security` hive of the Windows registry. When comparing the graph to another, the introduced Hungarian graph edit distance approach will use σ to determine the minimal cost of transforming G to H. In case of the exemplary file event `file.txt`, this edit distance is a mere 0.25, since a single σ_{RE} operation is sufficient to transform the biparte graph $G(svchost.exe, 1.5, file.txt)$ to $H(svchost.exe, 1.75, file.txt)$.

This approach of determining the minimal edit distance between two star-shaped graphs is used as the foundation for context-aware anomaly detection utilizing supervised learning on a per-process basis.

3.2 Anomaly Detection & Explication

The required transformation operations and, by extension, the minimal graph edit distance between to star structures can be used to determine the event-level deviation between two instances of the same process. In order to automatically determine unique thresholds for each observed process, we first need to create a template from a benign environment. Only then can we match base to target graphs and disseminate differences.

We have implemented the generation of templates in 3 different ways. In each case we take a set of benign process graphs and extract an optimal representative: The *perfect match* approach extracts identical events found in each iteration of a process and assembles an entirely new graph. This creates a sleek template that enables the analyst to primarily focus on hitherto unobserved events. However, there is an performance-for-accuracy trade-off that results in higher mean edit distance values. More accurate than

'perfect match', the *majority* approach picks the most common base graph from the input set and converts it to a template without altering its contents. However, this approach struggles with processes that show greater deviations between multiple benign instances due to their multifaceted nature. Lastly, we implemented the *prototype extraction* approach that is especially useful for diverse processes. It uses the Malheur algorithm (Rieck et al., 2011) to extract not one, but several prototypes representative of the various aspects of a single process. While template generation and distance calculation is most performance-heavy, it also promises the best accuracy by far.

Before any anomaly detection can be implemented, we need to set alert thresholds. In our case, they are determined by comparing the generated template(s) to the remainder of the benign input graphs. This yields a minimum, mean, and maximum lower-bound edit distance for each process. Depending on the level of scrutiny, both mean and maximum distance can be used as anomaly threshold. Armed with one or several templates for each process, we can now check unknown graphs against the predetermined thresholds.

As our system focuses on the interpretation of anomalies and not entire unclassified system traces, the amount of data processed in the explication stage is drastically reduced. Specifically, we explain why the anomaly detection routine has identified a star structure as significantly deviating from the template, thereby disseminating the in-depth knowledge gained in the process. Only afterwards can we commence with anomaly interpretation:

One of the advantages of our anomaly detection approach lies in the fact that the star depiction of of a graph allows for the comprehensive dissemination of semantic information that details each and every anomaly in a simple fashion. The analyst is presented a report detailing the events that constitute each deviation. Below snippet shows an example process being checked against one of its prototype templates:

```
==> svchost.exe [Deviation (threshold):
    300.5 (123.3) --> ANOMALY]
svchost.exe spawns 13 additional threads
svchost.exe loads 54 additional images
=> atl.dll
=> bcrypt.dll
   (...)
svchost.exe sets 6 additional registry entries
=> /HKLM/System
   (...)
svchost.exe modifies/deletes 6 additional files
=> 21253908f3cb05d51b1c2da8b681a785
```

The system allows for several levels of granularity. Registry paths can either be normalized to hive

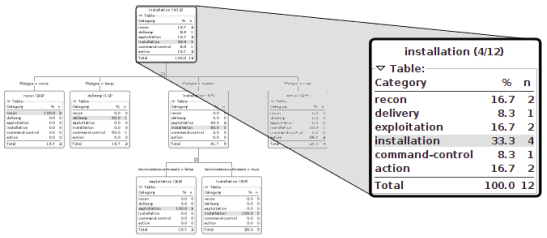

Figure 3: Stage affinity determined by the decision tree.

names as seen above or be processed in their entirety. Abstraction of ID numbers, memory addresses, user IDs is implemented as well – as is anonymization of private file names, IP addresses, and other personally identifiable information. This knowledge dissemination offers interesting information to the analyst but does not yet help with its interpretation. One of the key components of our proposed system is to explicate certain combinations of anomalous events and map them to the APT attack stages contained in the ontology. To this end, we explore event combinations via a decision tree in combination with the intelligent tagging of over 1,700 known Windows modules. In the first step, a number of competency questions that are expected to aid in the decision of whether a factor contributes to a malicious action were identified. These questions include simple Boolean queries into the presence of events over another event (e.g. if the number of thread terminations exceed the number of thread spawns) as well as decisions based on the presence of certain activity tags describing the base functionality of a module (e.g. networking, security, user interface, kernel, etc.). In order to support the development of expressive competency queries we manually defined a number of questions and continuously test them against the decision tree.

The decision tree is rooted in the six aforementioned APT categories. By answering all competency questions, we get a probability describing the graph's affinity towards a certain APT stage (see Figure 3). The key factors leading to the decision are identified as well, providing opportunities for future prediction and enabling aforementioned self-improvement of the decision system. Based on the response to the competency questions, we infer rules that describe the detected malicious actions. There are two stages to this process: In the *learning stage*, use genetic algorithms (Papagelis and Kalles, 2000) to directly evolve the decision tree. We then extract and feed the identified high-impact rules into our ontology, which links them to pre-specified APT stages, actors, assets, and concrete attack scenarios. During the *matching stage*, the decision tree becomes the means to determine and map newly discovered anomalies to the already populated ontology.

4 CONCLUSION

The current prototype of the system has been implemented in a test-bed environment consisting of 10 physical Windows machines actively used by developers and office personnel. The deployed kernel monitoring agent logs all the event types described in section 3.1 to a central listener that in turn writes the events to a database server. SQL is used to query the database and to construct the star structures that are the basis for all further processing. Our approach is able to selectively retrieve entire system sessions or pick out individual processes, whereby any temporal range can be specified. For example, we can process only the first *n* seconds after an application's launch or extract data from a specific point within its lifetime. The resulting set of CSV-formatted graphs is converted into matrices that are the foundation of Hungarian distance calculations implemented in R. The correlation of network flow events and process information is handled by a Python-based framework capable of grouping destination IP addresses by domain owner. For prototype-based template generation, we utilize a local Malheur (Rieck et al., 2011) installation configured to accept non-MIST input data. Decision trees are computed in GAtree (Papagelis and Kalles, 2000). Initial evaluation puts the computational requirements of the anomaly detection routines in the span of seconds to minutes, depending on the size of the graphs. Preliminary tests using the Windows generic host process against 18 automatically generated prototype templates have yielded correct anomaly detection results for a total of 81 out of 83 system sessions infected by over 15 classes of malware. The remainder was deemed inconclusive due to a lack of activity. A detailed evaluation of the system's anomaly detection accuracy and its reasoning capabilities will be discussed in future works. Further research will also be conducted into the improvement of the decision tree as well as the automation of the ontology mapping process. Ultimately, the introduced anomaly detection and explication system will offer invaluable aid to malware analysts and security operators alike.

ACKNOWLEDGMENTS

The financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged.

REFERENCES

Anagnostopoulos, T., Anagnostopoulos, C., and Hadjiefthymiades, S. (2005). Enabling attack behavior prediction in ubiquitous environments. In *Int. Conference on Pervasive Services*, pages 425–428. IEEE.

Dolgikh, A., Nykodym, T., Skormin, V., and Birnbaum, Z. (2012). Using behavioral modeling and customized normalcy profiles as protection against targeted cyber-attacks. In *Computer Network Security*, pages 191–202. Springer.

Dornhackl, H., Kadletz, K., Luh, R., and Tavolato, P. (2014). Malicious behavior patterns. In *2014 IEEE 8th Intl. Symposium on Service Oriented System Engineering (SOSE)*, pages 384–389. IEEE.

Hu, X., Chiueh, T.-c., and Shin, K. G. (2009). Large-scale malware indexing using function-call graphs. In *16th conference on Computer and communications security*, pages 611–620. ACM.

Hutchins, E. M., Cloppert, M. J., and Amin, R. M. (2011). Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Luh, R., Marschalek, S., Kaiser, M., Janicke, H., and Schrittwieser, S. (2016a). Semantics-aware detection of targeted attacks: a survey. *Journal of Computer Virology and Hacking Techniques*, pages 1–39.

Luh, R., Schrittwieser, S., and Marschalek, S. (2016b). TAON: An ontology-based approach to mitigating targeted attacks. In *iiWAS 2016*. ACM.

Münz, G. and Carle, G. (2007). Real-time analysis of flow data for network attack detection. In *10th IFIP/IEEE Int. Symposium on Integrated Network Management*, pages 100–108. IEEE.

Noble, C. C. and Cook, D. J. (2003). Graph-based anomaly detection. In *9th Intl. conference on knowledge discovery and data mining*, pages 631–636. ACM.

Papagelis, A. and Kalles, D. (2000). GA Tree: genetically evolved decision trees. In *12th Intl. Conference on Tools with Artificial Intelligence*, page 203.

Rieck, K., Trinius, P., Willems, C., and Holz, T. (2011). *Automatic analysis of malware behavior using machine learning*. Journal of Computer Security.

Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. M. (2002). Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and privacy*, pages 273–284. IEEE.

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., and Stiller, B. (2010). An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356.

Wagner, M., Fischer, F., Luh, R., Haberson, A., Rind, A., Keim, D., Aigner, W., Borgo, R., Ganovelli, F., and Viola, I. (2015). A Survey of Visualization Systems for Malware Analysis. In *Eurographics Conference on Visualization*, pages 105–125. EuroGraphics.