APT Attack Detection Method Based on Traceability Graph

Yihan Yin, Xiangjie He, Yiwei Liao
Institute of Computer Science and Information Engineering, Harbin Normal University, Harbin, China

Abstract: This article proposes an Advanced Persistent Threat (APT) attack detection method based on traceability graphs, aimed at addressing the complexity and concealment of APT attacks. This method describes system behavior by constructing a traceability graph, optimizing it to reduce redundant information, converting the traceability graph sequence into a feature vector sequence, and using an encoder decoder model to train the GRU (Gate Recurrent Unit) model to extract long-term features of the sequence. Finally, a normal behavior model is established through clustering to detect APT attacks.

Keywords: Traceability Graph; APT Attack; Attack Detection; Sequence Signature

1. Introduction

With the rapid development of information technology, Advanced Persistent Threat attacks have become a major challenge in the field of network security. APT attacks are usually characterized by strong targeting, tight organization, long duration, covertness, and indirect attacks[1].Attackers make use of social engineering and advanced attacking techniques to bypass the defenses of traditional security technologies, such as randomization of the address space, in order to steal sensitive information , destroying systems, etc. for the purpose of bringing serious economic losses and security threats to enterprises and society.

Currently, many enterprises rely on active defense techniques to cope with common attacks, but APT attacks are complex and hidden, making these tools to detect and prevent APT attacks have limitations. Suspected attacks in system operation, security detection system frequent alarms, a large number of alarms need to be identified and processed by security analysts, which not only increases the cost, but also due to false alarms caused by important attacks are easy to be omitted from the detection. Therefore, in order to effectively deal with APT attacks, it is necessary to study new detection and analysis methods, summarize the cause and effect relationship of attacks, and help security analysts to determine the major invasions, the attacker's means of evasion and the impact of the attack. The traceability graph can trace the cause and effect relationship, and can be used for a variety of network security tasks after processing and analysis[2]. In recent years, the use of traceability graph to detect APT attacks has received widespread attention, but the existing methods have shortcomings and need to be improved. In conclusion, APT attack detection based on traceability graph is of great significance and key to improving network security defense capability.

2. Related Concepts

2.1 APT Attack

According to several APT attack studies[2-3], the APT attack process usually covers the following phases:

(1) Implantation phase:

The attacker will first analyze the attack target, including collecting target information, clarifying the target's defense mechanism and determining whether the target has attack value, and also infiltrate with the help of technical means to obtain intelligence, and ultimately determine the attack means used. Then, based on the determined attack means and the characteristics of the attack target, develop the corresponding Trojan virus and malicious code, and use 0day vulnerability, spear phishing, mobile storage devices and other ways to implant malicious programs into the system.

(2) Confirmation stage:

The attacker works to implant the malicious code built in the previous phase into the system.

(3) Extension stage:

In order to be able to further access and control the system, attackers will establish attack strongholds, install backdoor programs and other

controls in the system, create software, replace or hijack legitimate code, or add startup code. In addition, to explore the network more deeply, an attacker may exploit bugs or vulnerabilities in the system configuration, modify notes using forged tokens, elevate his or her attack privileges through the registry, and so on.

(4) Mobile phase:

To further expand the scope of the attack, the attacker observes the network and system, steals accounts and passwords, and uses these legitimate credentials to access the system and create more accounts to help achieve their goal.

(5) Maintenance phase:

The attacker will endeavor to lurk in the system to avoid detection, thus maintaining his presence within the system for an extended period of time. Thereafter, the attacker will carry out a long period of continuous network infiltration and gradually obtain internal network privileges in order to hide in the internal network for a long period of time and continue to collect various types of information until important intelligence is stolen.

2.2 Traceability Graph

A traceability graph is a data structure extracted from audit logs that traces the causal relationships between entities and objects retrospectively by presenting the dependencies between the processes. A traceability graph consists of nodes that represent the entities and the objects as well as the edges that represent the actions between them, and it is a directed graph with edges pointing from the entities to the objects[3].

(1) Entity

Entities in a causal graph are objects in the system such as processes, files, network ports, etc., Entities are unique system subjects or objects extracted from the causal graph and represented as nodes. Entities can be processes, files, and network connections.

(2) Neighborhood graph

Given a traceability graph, if 2 nodes u and v are connected through an edge, the 2 nodes are said to be neighboring nodes. The neighbor graph of N_1 is a subgraph consisting of the node N_1 and its neighboring nodes and edges, and the neighbor graph of a group of nodes $\{N_1, N_2, \cdots, N_n\}$ is a subgraph including the node $\{N_1, N_2, \cdots, N_n\}$ and its neighboring nodes and edges.

(3) Event

An event ε is a quaternion consisting of a source (src), an action, a destination (dest), and a timestamp (t). Where source and target are 2 entities connected by an action, and the timestamp indicates when the event occurs. Given an entity e, its events can be extracted from the adjacency graph of that includes all the actions associated with the neighboring nodes of that.

(4) Sequence

Given an entity e, a sequence S can be extracted from the causal graph. The sequence S contains all the events in the neighborhood graph of the entity e in chronological order, denoted as $S\{e\} = \{\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_n\}$. Similarly, if a set of entities is given, a sequence including all events in their neighborhood graph can be extracted.

2.3 Optimization Techniques for Traceability Graph

(1) Edge reduction techniques

Including Causality Preserving Reduction, Process-centered Causality Approximation Reduction and Domain knowledge-based reduction. PCAR retains causality and removes repeated read/write operations unrelated to the target file; DOM mainly removes temporary files.

(2) Node Reduction Technique

This technique focuses on the life cycle of objects. For example, NodeMerge, which uses fixed libraries and read-only resource sets to reduce system event data.

3. APT Attack Detection Method Based on Traceability Graph

3.1 Data Preprocessing

(1) Deletion of edge nodes

Some nodes in the traceability graph have no operations with other nodes, and such nodes will not be used when learning with the model, so they are all removed.

(2) Delete duplicate edges

There are a large number of repetitive operations in the system, resulting in many duplicate edges between nodes, deleting these redundant edges will not cause too much loss to the overall semantics, and can effectively optimize the traceability graph, so all edges between 2

entities in the traceability graph are deleted except for the first operation, i.e., only the first event of the two is retained.

(3) Merging identical nodes and edges

If the structure of certain combinations of nodes and edges in the traceability graph is identical, it means that they represent the same events that occurred, and these nodes can be merged into a single node, and then the earliest operations in these nodes are retained as edges of the new node.

3.2 Traceability Graph Feature Extraction

The way of generating the sequence of traceability graph features in UNICORN[4] is adopted, which is divided into the following 3 steps:

(1) Collecting system call logs, extracting subjects and objects as traceability graph nodes from the logs, generating edges of the traceability graph based on the call relationships between the nodes, and generating the traceability graph using Cam Flow[5].

(2) Using the information of traceability graph nodes as well as their neighboring nodes, node labels are generated and the statistical information of traceability graph node labels is recorded in the form of histograms.

(3) Since step (2) may generate node labels that have not appeared, the number of transverse coordinates of the histogram is uncertain, and the general algorithm requires a fixed-length feature vector as input. The HistoSketch[6] algorithm can effectively solve this problem by transforming the histogram generated in step (2) into a fixed-length feature vector. Denote this sequence of feature vectors as $S = \{f_1, f_2, \cdots, f_n\}$, where $f_i \in R^d$, its dimension is d-dimensional and n denotes the length of the sequence.

3.3 Sequence Feature Extration

(1) Data Dimensionality Reduction

In the sequence S of input features, the feature f_i in the sequence describe the state of the system at the time i, so when two system states are similar, such as neighboring system states, the corresponding two feature vectors are also similar. This makes the features extracted directly from the traceability graph have high dimensionality and contain a lot of redundant information. Therefore, in part 1 of this module, a multi layer perceptron is first used to reduce the dimensionality of all features in the sequence to generate tighter feature vectors in a lower dimensional space. By this way both the redundant information can be removed, the computational amount can be reduced, and the efficiency of attack detection can be improved, and at the same time, the proportion of valid information in the features inputted to the subsequent module can be improved, and the feature sequence after dimensionality reduction can be memorized as $X = \{x_1, x_2, \cdots, x_n\}$.

(2) Sequence Feature Extraction

In this paper, the encoder-decoder based model is used to train the GRU model to extract sequence features. And in the testing phase, only the model and parameters of the encoder part are kept.The role of the encoder is to encode the sequence of features into a feature vector which contains the information of the whole sequence, so the encoder module adopts the GRU model which can efficiently extract the features of the long term sequence. The formalization of the encoder module is described as $Output, feature = Encoder(X)$. The feature sequence X after dimensionality reduction is input into the encoder module composed of the GRU model, and the output includes the feature sequence $Output$ and the feature vector $feature$, where $Output \in R^{n \times d}$ is a sequence composed of feature vectors whose feature dimension is equal to the dimension of the feature vector in the input sequence, and the length of the sequence is equal to the length of the input sequence. During each round of iteration, the GRU model outputs a feature vector which contains the local feature information of the input feature sequence. These output feature vectors are put together in order to form the output feature sequence $Output$, which is the optimization of the input feature sequence. At the same time, the output after the last iteration is used as the feature vector $feature$, which incorporates the features of the whole sequence, contains the long-term history information of the sequence, and can reflect the characteristics of the whole sequence. In the attack detection stage, the direct output is used as the input of the subsequent clustering module.The function of the decoder module is to reconstruct the input feature sequence X according to the sequence features $feature$

generated by the encoder.

3.4 Attack Behavior Modeling

Since only the normal behavioral data of the system is available during the training process, this paper uses clustering to gather all the feature vectors extracted from the training set into K classes. Where the K value is a hyperparameter that needs to be set in advance. Through the parameter experiment, it can be seen that K has a better performance in the range around 7. In this paper, the number of clusters is set to 7. These feature vectors contain the system behavior characteristics, and the system behaviors with similar behavioral properties will be closer in the feature space[6]. In the attack detection phase, the distance between the extracted feature vectors and the clustering center is calculated, and the behaviors exceeding the threshold are judged as attacks.

4. Conclusion

In this paper, we propose an APT attack detection method on traceability graph. The method uses a sequence of traceability graphs to describe the changes in system behavior and transforms the sequence of traceability graphs into a sequence of features, after which a GRU model is used to capture the long-term change characteristics of the sequence. In the model training phase, a model based on encoder-decoder architecture is used to train the GRU network model; in the attack detection phase, the trained GRU model is used directly to extract sequence features. Finally, a clustering algorithm is used on the training data to model the normal behavior of the system, and in the attack detection phase, sequences far from the center of the clusters are identified as attacks.

Acknowledgment

This present research work was supported by Harbin Normal University Higher Education Teaching Reform Research Project(No. XJGZ202409).

References

[1] Anjum M M, Iqbal S, Hamelin B. ANUBIS: a provenance graph-based framework for advanced persistent threat detection[C]// Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. New York: ACM Press, 2022: 1684-1693.

[2] Liu J X, Shen Y, Simsek M, et al. A new realistic benchmark for advanced persistent threats in network traffic[J]. IEEE Networking Letters, 2022, 4(3): 162-166.

[3] Corallo A, Lazoi M, Lezzi M, et al. Cybersecurity awareness in the context of the industrial Internet of things: a systematic literature review[J]. Computers in Industry, 2022, 137: 103614.

[4] Han X Y, Pasquier T F J M, Bates A, et al. UNICORN: runtime provenance-based detector for advanced persistent threats. In: Proceedings of the 27th Annual Network and Distributed System Security Symposium, San Diego, 2020.

[5] Pasquier T F J M, Han X Y, Goldstein M, et al. Practical whole-system provenance capture. In: Proceedings of Symposium on Cloud Computing, Santa Clara, 2017. 405-418.

[6] Yang D Q, Li B, Rettig L, et al. Histosketch: fast similarity-preserving sketching of streaming histograms with concept drift. In: Proceedings of IEEE International Conference on Data Mining, New Orleans, 2017. 545-554.

