AGE-GNN: Adversarial Generator using Graph Neural Network for ML Evasion

Kunal Mukherjee
The University of Texas at Dallas

ABSTRACT

Cyber-criminals have been known to foil traditional defenses mechanisms such as signature-based and network-based defenses by using *zero-day* vulnerability or using *advanced persistent threat(APT)* attacks. The current state-of-the-art defences employ machine learning based detectors such as *LOF, AutoEncoders*, and *Graph Neural network*. The current ML based defenses focus on identifying abnormal patterns in the system to distinguish between normal and malicious processes in a system. But, recent attacks have been executed by using advanced stealthy techniques such as MITRE's ATT&CK Tactics, Techniques, and Practices(TTP) to bypass the ML based detection engine by impersonating and abusing trusted programs as well as erasing memory foot print.

To defend against these kind of advanced attacks researchers have developed powerful deep structures such as *Variational Auto Encoders*(VAE) and *Graph Neural Network*(GNN). These deep models are most well suited for capturing and learning system subject relationships, so that when any program deviate from the norm, they is easily identified. Therefore, advanced attacks such as APTs can be mitigated. With the use of these deep structures, they also have the risk of exposing the inside model that can be used by threat actors to devise adversarial examples that can go undetected by LOF, VAE and GNN.

In this paper, we present to you AGE-GNN, a graph neural network that is used to construct adversarial examples for a given LOF and VAE model. AGE-GNN can be used by researchers to evaluate current state of the art detection models. It can also be used to revamp the current deep structures to make it most suited for today's threat level as well as provide supplemental information to the security analyst and forensic analysts regarding what type of program behaviors were able to go undetected, so that they can be mitigated.

CCS CONCEPTS

• **Security and privacy** → Vulnerability management;

KEYWORDS

System Security, Variational AutoEncoders, Graph Neural Network, Adversarial Example

Conference'17, July 2017, Washington, DC, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Security landscape is changing everyday and threat actors are using previously unseen methods to attack secure infrastructure. Previously employed successful defenses such as signature-based [58] and network-based [52] fail miserable against stealthy attacks such as zero-day vulnerability [8], malware mutants [73] and advanced persistent threats (APTs) [13, 41, 61] attacks. It is natural that signature based defenses will fail against new attacks because these defenses rely heavily of structure and past information. Since, they have not seen the source code or program signature before their detection capability fails.

Network based defense fails because firewall and intrusion detection system do not guard well trusted programs [67] because of white listing as well as user specified exceptions. Recent studies and external industry researchers have shown that successful attack campaign have been possible due stealthy attacking vectors such as malicious programs impersonating or abusing well-trusted programs [27], where the malicious behavior is blended with benign behaviors of the targeted program. Therefore, the unknown characteristics of APTs make them most deadly as historical information regarding them are missing and most security triage happen by evaluating archived historical data and procedures.

The maturity of Machine Learning(ML) methods and applications have reached the security domain also. Security community has accepted ML based approaches as valid which is apparent by looking at recent publish papers [69] [31]. Initial papers, even though showed promising result used simple ML based approaches such as SVM [54], PCA [70], ensemble methods [31] such as bagging [25]/boosting [6] and local outlier factor(LOF) [23, 64]. But, recent stealthy attacks [65] as well as APTs [7, 55] have been able to mislead these ML based approaches.

Current and on-going work has been using two prominent ML technique called Variational Auto Encoders(VAE) [29] and Graph Neural Network(GNN) [62] based anomaly detection. While the specific details may vary, the high level idea is similar. They are first training a model with benign system activity and then uses it to classify system activities with the goal that system or user process that have been hijacked by malware(*e.g.,* attacker) will show abnormal system activity that can be identified. It is clear that for attackers to cause harm or do their bidding, they will have to interact with system entities like files, network sockets and processes. These interaction or behavior would be different than the benign program's normal behavior and this insights has motivated the latest ML based detection models.

These work has mentioned that there are two issues with the ML based threat detection approach [66], first, the training of the model takes considerable amount of time as well as resources. Second, the false positive rates is moderately high due to miss-classification of benign process as malicious. The miss-classification happen for valid reasons such as benign programs getting a feature after update, change in configuration, usage pattern and interface, and interaction with programs have changed. Therefore, the security forensic analyst who is in the loop for deployment pipeline has to continuously monitor and release benign processes that have been falsely identified as malicious. While, these issues are mentioned, it does not take away the fact that VAE [19, 30, 51] and GNN [71] detectors have shown superior performance among anomaly detectors.

Through a purely ML-based approach, AGE-GNN automatically mines the event history database of the target system to find and synthesize the attack vectors necessary for an APT-scenario. With an extended threat model that assumes a strong adversary with access to detection model internals and the target network, the adversary uses system data to replace the attack vectors with attack constructs which we name as *APT-gadget* (or in short *gadget*). In this sense, the APT-gadget defines a structure that carries malicious semantics when executed in reference to the APT-attack but is naturally deemed benign within the context of the target system. At a high level, AGE-GNN attempts to reverse the modeling operation of the target system. Given an attack vector, AGE-GNN will return a list of potential replacement candidate events along with a respective score, which we name as *regularity score*, used to measure the evasiveness of the *APT-gadget*. For the final solution to be realizable, AGE-GNN is built through a database of extensive data collection from real world networks with event traces collected from eighty-six hosts over eleven months.

We propose AGE-GNN, a systematic approach that re-examines the current practices of ML-based security by considering advanced active adversaries who focus on exploiting the weaknesses of ML-based security through evasive attacks. In particular, we assume an adversary who seeks to extend the APT-style attack campaign by integrating a series of advanced attack vectors [47] into a single attack scenario to avoid being detected by the ML-based security model.

We evaluated AGE-GNN with a realistic APT scenario [7, 55] to confirm our research insight. In our testing, AGE-GNN successfully found several replacements for all attack stages of our APT attack while still achieving an equivalent attack capability of the original. Although our work is built on a specific ML-security implementation [23, 64], the results empirically verify the feasibility of AGE-GNN.

In summary, the paper makes the following contributions.

- We developed a graph neural network(GNN) that generated adversarial examples for VAE and LOF based anomaly detector models.
- A purely data driven approach to contrive necessary attack vectors with minimal human oversight.
- Our study is verified from realistic dataset collected over eleven months duration from eighty-six hosts.
- We publicize our dataset and code for future research.

2 BACKGROUND

In this section, we give the necessary background on system events and program behavior. It also expands on the local outlier factor (LOF) [64], Variational Auto-Encoder(VAE) [29] and Graph Neural Network (GNN) [62] based detectors on which our AGE-GNN is based.

2.1 System Events

A system event [65] is made up of system entities(*e.g.,* process, file and socket) and its corresponding relation(*e.g.,* read, write and execute), which is interactions with other entities. A chain of system events is known as a causal path. Causal path can be used to detected if a program is benign or anomalous. After detection the causal path can be used to find the infection point. Most papers [20] due to resource constraint select only three types of system entities: processes, files, and socket(network connection). The system entity and relations they considered are:

- Process - Process: start, end
- Process - File: read, write and execute
- Process - Socket: read, write

2.2 Adversarial Example

Machine Learning(ML) models have been used for classification of different objects and security community has leveraged this to generate ML based malware detection models [23, 64, 69]. These models are able to predict if a causal path is malicious or not. An adversarial example [56] is an anomalous causal path with a small deviation in terms of detection such as new entity added or relationship formed between entities, so that the ML model makes an in-accurate prediction. Adversarial examples are extremely important as they show how an ML model is vulnerable to attacks and how much capability does an attacker needs to synthesize the attack.

2.3 Graph Neural Network

Graph Neural Network(GNN) [72] are neural network structures that are well-suited to learn graph based structured data. System entities and their relationship can be efficiently modeled as a directed graph. The entities can be modeled as vertex and relations can be modeled at edges. The causal paths would be a list of vertexes and their corresponding edges that start from the process of interest and end when the path reaches a node that has no out going edges.

GNN basically performs transformations on all the attributes of a graph that preserves graph symmetries. In general, GNN [72, 76] accepts a graph as input and iterative transforms the embedding without changing the connectivity of the graph. GNN [74] has a prevalent message passing framework that can be efficiently utilized to model system entity relationships, that's why many variational autoencoder are implemented as GNN [29]. Therefore, GNN are also well suited to create adversarial examples as it has information regarding node and its neighbouring edge connectivity. So, we can create a causal path that consists of entities which are benign in nature but has one or two outgoing node-edge relation that does the attacker's bidding.

2.4 ML detection models: Local Outlier Factor and Variational AutoEncoder

Two most prominent ML models used for malware detection are LOF [23, 64] and Variational AutoEncoder [9]. The ML workflow for LOF and autoencoder are shown in Figure 1. The provenance graph is first created for the target program which we want to analyze. The provenance graph is used to extract the causal paths using path

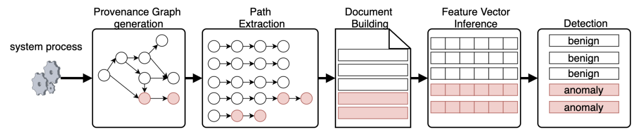

Figure 1: Detection workflow of ML based detector.

Table 1: APT TTPs for cyber-killchain stages

Cyber-killchain Stages	Techniques (ATTCK TTP)	Scenarios
Gain Access	Exploitation for Client Execution (T1203)	Attackers modifies a benign looking executable, but once the user opens the application it can be used by the attacker for arbitrary code execution
	File and Directory Permissions Modification (T1222)	Attacker modifies objects in the system, so that it can be copied by lower priviledge users that the attacker have hijacked
Establish a Foothold	Data from Local System (T1005)	Attacker moves around the file system, finding files that contain valuable information
	Exfiltration Over C2 Channel (T1041)	Attacker downloads valuable files into local directory
Deepen Access	Create and Modify system process (T1543)	Attacker creates system process that can run in the background and do reconnaissance or mine information
	Service Stop (T1489)	Attacker stops firewall or external IDS services, so that they cannot detect the APT
Move Laterally	Process injection (T1055)	Attacker injects vulnerable process such as trojan into a benign application, so that IDS cannot differentiate.
Look, Learn and Remain	System Information Discovery (T1082)	Attacker discover system hardware information so that they can craft better exploits or exploit hardware vulnerability
	Network Service Scanning (T1046)	Attackers scans network services to find services they can use as backup or use as a secondary mode of connections.
	Network Sniffing (T1040)	Attackers sniffs the network to find insecure SSL connection or any other connection to extract valuable information

selection algorithm mentioned in [23] and they are embedded into feature vectors (*e.g.,* doc2vec [33]). Finally, the feature vectors are used for training, validation and testing(*e.g.,* detection).

Malware detector VAE is a deep generative model whose architecture consists of three parts components: an encoder, a decoder and a loss function. The encoder and decoder network are deep neural network, convolution layers. The prime functionality of encoder is to compresses the feature vectors of the causal path to a low-dimensional vector, v, while decoder reconstruct the original feature vectors from v to match the input feature vector. The loss function tries to minimize the difference between original feature vector and the reconstructed feature vector.

2.5 APT Attack Scenario

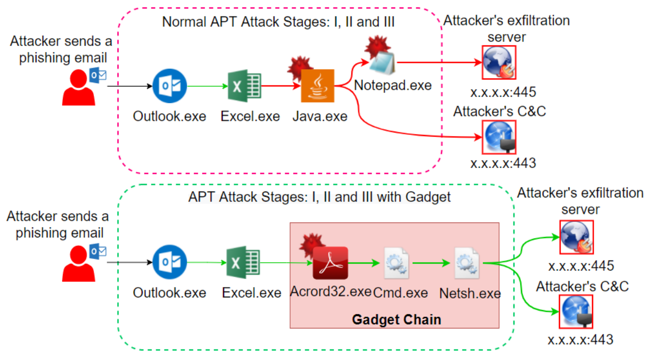
Normal APT Attack Stages: I, II and III
Attacker sends a phishing email
Outlook.exe
Excel.exe
Java.exe
Notepad.exe
Attacker's exfiltration server
x.x.x.x:445
Attacker's C&C
x.x.x.x:443
APT Attack Stages: I, II and III with Gadget
Attacker sends a phishing email
Outlook.exe
Excel.exe
Acrord32.exe
Cmd.exe
Netsh.exe
Gadget Chain
Attacker's exfiltration server
x.x.x.x:445
Attacker's C&C
x.x.x.x:443

Figure 2: Provenance graph for APT attack vs APT with gadget.

Advanced attack campaigns using APT-like attacks follow set stages. While details may vary, the typical APT scenario in an advanced attack campaign can be modeled into a systematic framework for analysis. The Cyber-killchain framework [1] divides an APT campaign into multiple sub-stages listed in MITRE's ATT&CK framework [47] and shown in Table 1. ATT&CK, a public knowledge framework for attacker Tactics, Techniques, and Practices(TTP) accumulates prominent attack vectors needed in different APT stages.

In a typical APT attack scenario, a potential adversary initiates the campaign by sending an email to the target victim with an embedded malicious macro. Figure 2 introduces a typical leakage attack where the target network is managed by a domain controller host configured with the necessary components needed to set up a typical enterprise network (*e.g.,* firewall, network proxy, mail server *etc.* .) As the email is opened, the embedded macro writes a payload to the filesystem which then impersonates itself as a benign program(*e.g.,* java.exe). To minimize its signature footprint, the payload deletes itself after execution of different malicious process(*e.g.,* notepad.exe) and operates from address space afterwards.

Once a foothold is established, the program moves laterally to gather the information necessary to further exploit the system. In particular, vulnerabilities allowing for privilege escalation (❷) are sought after to gain local root access allowing the attacker to monitor other processes on the host or steal additional credentials. After penetrating the domain controller and obtaining control over the entire domain (❸) the final step of the attacker is to ex-filtrate the desired assets from the database (❹) to a remote host(❺) accessible by the attacker.

2.6 Threat Model

Our threat model assumes that the attacker has in-depth knowledge of the victim network. We assume an adversary who has given sustained efforts to obtain the measures and information needed to launch an APT-style attack. Specifically, we assume that the attacker has gain detailed knowledge regarding 1 ML-based security solutions and modeling approaches, 2 the status of the established model, 3 datasets which have contributed to the model, and 4 schematics of target network.

The attacker intricately knows the normal behaviors of the victim network and possesses sufficient resources to construct his own anomaly detection models based on such knowledge. The specific ML-based detection approach and internal state of the victim's established detection model is also known to the attacker and is used by the attacker to construct a stealthy attack in hopes of evading the ML-based detector.

3 ATTACK SYNTHESIS

In this section, we discuss gadget creation and usage along with the gadget synthesis methodology. First, we discuss gadget limitations and the validity of attacks generated through gadget chaining. Next, we discuss how gadgets are mined from provenance graph data. Finally, we will discuss the metrics used to calculate the anomalous score of gadgets along with synthesizing an attack using mined gadgets.

In an APT attack, the main challenge of the attacker is structuring his attack in such a way that the effects of the attack go unnoticed by the defender even with an advanced review of the target system. This task is made difficult by the use of provenance-based tools such as ProvDetector [64], and NoDoze [23] that allow defenders full access to all events in the target system. Even with the use of stealthy malware techniques to hijack benign system programs or deny defenders access to malware binaries, defenders are still able to identify attackers based on behavior alone.

3.1 Gadget Chaining

Regardless of the type of APT attack, the attacker must at some point interact with the system to deliver the payload of the attack and perform some type of action to execute the payload. We define the Intro Gadget to be the first event an attacker executes to start their attack. Because the attacker must make some conscious decision regarding entry into the system and the following execution of his payload, the source of the Intro Gadget is predetermined by the attacker. Specifically, the *src* is fixed by the attacker in the system event *e = (src,dst,rel)*. Intro Gadgets then serve as the boundary between user and attacker-controlled events, and mark the earliest point at which defenders can identify the attacker. Within a chain of gadgets, Intro Gadgets are placed at the start of the chain.

Thus, to hide the effects and prevent the detection of the attack, the attacker may use a Payload Gadget to mask his behavior. We define the Payload Gadget to be the final system event in the action path of the attacker, ending with the process that performs the desired malicious actions of the attacker. Because the payload of the attacker must be executed to conclude the attack, the Payload Gadget must end with the payload package. Specifically, in the system event $e = (src, dst, rel)$, the dst of the Payload Gadget is fixed to the attacker's payload. Through the use of the Payload Gadget, the attacker can ensure that the final method used to execute the payload is benign in appearance.

The Gadget Chain contains a minimum of one single gadget to a series of gadgets that are chained together in such a way that the individual connections between the system events in the gadgets appear benign. Within the Gadget Chain, the Intro Gadget encapsulates the start of the attack and the Payload Gadget represents the system event responsible for the final malicious effect. Connector Gadgets form the middle remaining gadgets in the Gadget Chain. Overlap may occur between the gadgets (e.g. there can be a chain with a single gadget if the behavior of the Intro Gadget and Payload Gadget are the same).

In AGE-GNN, Gadget Chains represent the sum of actions of the attacker: from the point that they gain access to the system to the final execution of the malicious payload. Using the method from [64], the anomaly score of the entire chain can be calculated as a product of the gadgets within the chain. Since the anomaly score is propagated through the entire gadget chain, the strength of the attack chain is measured by the weakest gadget in the chain.

3.2 Realizable Attack Vectors

Gadgets provide a way for anomalous connections to hide behind *benign masks* by replacing their anomalous connections with historically benign connections. While many gadget chains can be built from available gadgets, there still exists a real world limitation in that the final attack path of the resulting *gadget chain* must be realizable. For any attack utilizing gadgets, there must exist a continuous path of execution through every gadget in the *gadget chain* to ensure that the final payload can be activated. Additionally, for the *gadget chain* to succeed and escape detection, the sum of all gadget behavior must match the surrounding system environment well enough to escape detection.

Of these limitations, the ability of gadgets to achieve the desired goal while still passing under the radar poses the largest problem for the attacker to solve. In the threat model, it is already assumed that the attacker is capable of carrying out an advanced APT-like attack upon the target system and thus it can be assumed that the attacker is already capable of achieving his desired malicious goals. The utility of gadgets exists solely to provide the attacker a way to escape detection. However, since the use of gadgets limit the attacker's actions to involve only processes that are common to the system, sufficient gadgets must be found that contain the necessary paths of execution needed to exploit the system.

3.3 Gadget Mining

Gadgets are unique to the specific behavioral profile of a given system and directly depend on the actions of the native processes within. To maximize the probability of success, gadgets should be mined from system data using the same evaluation metrics of the defenders to circumvent both human and automated anomaly detection tests. For this purpose, common programs with a high frequency in the target system along with common benign profiles make good candidates for gadgets. The graph neural network(GNN) is trained using benign system events that has the same frequency distribution, therefore the gadget the GNN would recommend will have a high frequency.

Once an attacker is familiar with the normal operations of the target system, gadget mining can be done in a straightforward manner.ProvDetector [64] calculates the anomalous score of a given system event by the frequency of its appearance and connections in the overall system. One way of circumventing this score calculation is to find and use only gadgets with a high frequency of appearance in the target system which AGE-GNN does. A simple ordering of system events by number of appearances can be used to create a list of gadgets that have the highest likelihood of passing detection.

3.4 Anomaly Score Calculation and Propagation

To avoid detection, an attacker using gadgets hopes to identify and replace the rare causal paths in the attack created as part of the APT campaign. The rare paths are calculated by finding the *regularity score* of the causal paths λ, much alike previous work [64] and [23]. The regularity score for a causal path $\lambda = \{e_1, e_2, \ldots, e_n\}$ is defined as $R(\lambda) = \prod_{i=1}^{n} R(e_i)$, where $R(e_i)$ is the regularity score of an event e. The regularity score of an event e is defined as $R(e) = \frac{|Freq(e)|}{|Freq_{src\_rel}(e)|}$. Here, $Freq(e)$ is equivalent to how many times the system event e has occurred in the historic window with all 3-tuples (src, dst, rel) of e being the same and $Freq_{src\_rel}(e)$ is equivalent to the frequency of system event e where only src and rel from the 3-tuples are the same.

The regularity score of a system event shows the probability of a specific system event occurring. If event e has never occurred in the system, then the value of $R(e)$ is 0 indicating that e is a rare system event. If e is the only event between the *src* and any other entity with *rel*, then $R(e)$ is 1 indicating that e is a popular system event. Therefore, the regularity score of a causal path λ is dependent on the regulatory score of its constituent system events, $\{e_1, e_2, ..., e_n\}$. Thus, a causal path composed of many rare system event would have a low regularity score, resulting in a higher probability of detection.

3.5 Gadget Finding Objective Function

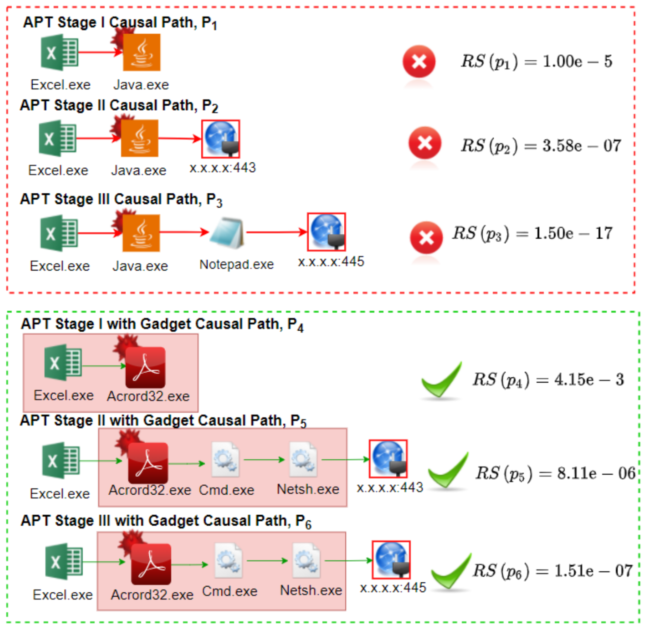

Figure 3: Regularity Score calculation for causal paths for APT attack vs APT attack with gadgets.

The purpose of AGE-GNN is to find system events with high regularity scores that can be used as gadgets to replace events in an attack with a lower regularity score. In this way, the overall regularity score of the entire attack path will increase, making it harder to detect as shown in Table 2. The regularity score can be used as a justification regarding the evasiveness of the gadget or in other words we can provide a justification of how a causal path was able to evade ML based detection. We formalize our problem statement as *finding the top K popular system events with high regularity scores that can be used to replace rare paths in an attack.*

Figure 3 shows the regularity score for the causal paths that were extracted from Figure **??**. Using Table 2 and focusing on the *events in APT Stages using Gadget Chain 1* column, we can see that the regularity score of gadgeted events are larger than the events of APT attacks without gadgets. From Figure 3 causal path p_1, we can notice that the APT attack Initial Access has the events excel.exe $\longrightarrow$ java.exe and we can find the regularity score of the event from Table 2 using the *events in APT Stages* column to be 1.00e-05. Therefore, using *events in APT Stages using Gadget Chain 1,2,3* columns in Table 2, we try to find the best gadget for the system event such that the gadget is much more frequent in the system such as excel.exe $\longrightarrow$ acrord32.exe which has regularity score of 4.15e-03 or excel.exe $\longrightarrow$ chrome.exe with regularity score of 1.81e-04 and should form a chain such that all the APT attacks can be realized. Table 2 is created in such a way that the gadgets from the top row connects to the gadget below it. So, that the entire APT attack chain can be realised by following each of the gadgets, they ranked in such a way that Gadget Chain 1 had the best regularity score and Gadget Chain 3 has a good regularity score.

An interesting point to note is that in the APT attack initial access, excel.exe $\longrightarrow$ chrome.exe has a higher regularity score than excel.exe $\longrightarrow$ acrord32.exe, but in the long run(*i.e.,* all the APT attack stages combined) the gadget chain starting with excel.exe $\longrightarrow$ acrord32.exe has a much higher regularity score of 4.56e-17 compared to others (*e.g.,* 2.96e-19 and 4.94e-27) as seen by the last row in Table 2, which is the product of all the event's regularity scores in the particular column. As mentioned in the earlier sections, when we are trying to find gadgets we are trying to balance two criteria: *(1)* finding relatively popular system events, *(2)* ensuring the system events can be chained together to create all the APT attack stages with a higher regulatory score compared to their original apt attack stage counterpart.

This can be seen for other APT attack stages such as Privilege Escalation, where we have three potential distinctive gadgets as seen in fourth row of Table 2. Gadget one, netsh.exe $\longrightarrow$ x.x.x.x:445, is a very frequent system event as netsh.exe [40] is a system application used for modifying network configuration and connects to domain controller frequently. Gadget two chrome.exe $\longrightarrow$ x.x.x.x:445 is a very frequent system event as chrome.exe [17] is a user application used to connect to domain controller. Gadget three is interesting because we were able to find system applications from the frequency database that are indeed common and do form a chain *e.g.,* chrome.exe $\longrightarrow$ svchost.exe $\longrightarrow$ powershell.exe $\longrightarrow$ x.x.x.x:445, but it can be noticed that forming a big chain has its disadvantages by looking at its regularity score in the last column and also, intuitively, a larger causal path has higher probability of being noticed.

From Table 2, it can be seen that occasionally gadgets such as those in Exfiltration are used where the attacker needs to run cscript.exe. There are few system events that can be used for this purpose but incidentally, the events are not as rare as other APT stages and thus can be kept the same. cscript.exe is windows script host that is commonly used by different database applications to connect to osql.exe and sqlservr.exe [39].

3.6 Gadget Finding Algorithm

To increase the regularity score of a path with rare connections, select gadgets need to be found that can replace rare connections and increase the regularity score as shown in Figure 3. As mentioned, a path consists of a collection of system events $p_{rare} = \{e_1, e_2, ..., e_n\}$ with the overall regularity score equal to the product of all individual scores in the path. Because rare system events have a regularity score much lower than that of common system events and scores propagate to the overall score of the path, replacing the events with gadgets significantly increases the regularity score of the path.

We present a robust gadget finding algorithm such that a path with a low score, like one containing an APT attack, can be made

Table 2: APT attack stages using gadget chain and their regularity score.

MITRE ATT&CK TTP	events in APT Stages	events in APT Stages using Gadget Chain 1	events in APT Stages using Gadget Chain 2	events in APT Stages using Gadget Chain 3
Initial Access	`excel.exe` → `java.exe` (1.00e−05)	`excel.exe` → `acrord32.exe` (4.15e−03)	`excel.exe` → `chrome.exe` (1.81e−04)	`excel.exe` → `chrome.exe` (1.81e−04)

Establish a Foothold	`java.exe` → `x.x.x.x:443`
	(3.58e−01)
	`java.exe` → `notepad.exe`
	(1.00e−05)
		`acrord32.exe.exe` → `cmd.exe`
		(3.86e−02)
		`cmd.exe` → `netsh.exe`
		(2.31e−01)
		`netsh.exe` → `x.x.x.x:443`
		(2.19e−02)
			`chrome.exe` → `x.x.x.x:443`
			(9.11e−01)
				`chrome.exe` → `x.x.x.x:443`
				(9.11e−01)

Privilege Escalation	`notepad.exe` → `x.x.x.x:445`
	(1.50e−06)
		`netsh.exe` → `x.x.x.x:445`
		(4.08e−03)
			`chrome.exe` → `x.x.x.x:445`
			(1.16e−05)
				`chrome.exe` → `svchost.exe`
				(1.16e−05)
				`svchost.exe` → `powershell.exe`
				(2.23e−03)
				`powershell.exe` → `x.x.x.x:445`
				(5.75e−03)

Deepen Access	`notepad.exe` → `cmd.exe`
	(1.00e−05)
	`cmd.exe` → `cscript.exe`
	(4.23e−04)
		`netsh.exe` → `cmd.exe`
		(2.77e−02)
		`cmd.exe` → `cscript.exe`
		(4.23e−04)
			`chrome.exe` → `cmd.exe`
			(3.10e−04)
			`cmd.exe` → `cscript.exe`
			(4.23e−04)
				`chrome.exe` → `msiexec.exe`
				(2.31e−04)
				`msiexec.exe` → `cmd.exe`
				(1.76e−03)
				`cmd.exe` → `cscript.exe`
				(4.23e−04)

Exfiltration	`cscript.exe` → `osql.exe`
	(1.76e−03)
	`cscript.exe` → `sqlservr.exe`
	(6.67e−01)
		`cscript.exe` → `osql.exe`
		(1.76e−03)
		`cscript.exe` → `sqlservr.exe`
		(6.67e−01)
			`cscript.exe` → `osql.exe`
			(1.76e−03)
			`cscript.exe` → `sqlservr.exe`
			(6.67e−01)
				`cscript.exe` → `osql.exe`
				(1.76e−03)
				`cscript.exe` → `sqlservr.exe`
				(6.67e−01)

Regularity Score	2.67e−28	4.56e−17 **(best)**	2.95e−19 **(better)**	4.94e−27 **(good)**

popular to avoid detection. For our algorithm, we assume the scenario of an attack campaign where the attacker launches an APT attack against a target system. Because of the rarity of the attack however, the casual path containing the attack will have a low regularity score relative to the average paths throughout the system and will be easily detected by ML-based detectors.

To bypass detection, we assume that the behavior of the overall system is known to the attacker as stated in the threat model and that the attacker is able to obtain the provenance graph structures that can be generated from the system. We also assume that the attacker is able to obtain the causal path data of the system using a similar methodology as [64] and is able to create a system event frequency database or frequency database (*i.e.,* FreqDB) similar to [23]. Finally, we assume that the attacker is able to use the rare path detection algorithm defined in [64] to select the top k rare paths based on regularity score. From this, the attacker can calculate the exact regularity score and rarity of his attack.

Thus, to increase the low regularity score of their attack, the attacker can now replace the rare system events in the path with more popular system events thereby increasing the regularity score of the entire attack causal path until the path score is above the threshold used by the defenders to detect anomalous behavior. This process is formalized in algorithm 1 where the attacker first finds the rare events in the causal path with the help of function GetRareEventsFromPath, algorithm 2. The algorithm 2 finds rare edges in a given causal path by iterating though causal events in the causal path and calculating the regularity score for each causal event. If the regularity score is below a certain threshold, *MIN_PROB_THRES_EVENT*, the event is marked as rare. The threshold, *MIN_PROB_THRES_EVENT*, is set empirically through the examination of event frequencies in the frequency database.

Once the rare edges lowering the score of the overall path have been identified, algorithm 1 proceeds to replace the rare edges with suitable gadgets using function ReplaceRareEventsWithGadget, algorithm 3. The algorithm 3 takes the rare edges that the attacker wants to replace and finds suitable gadgets such that the regularity score of the gadget replaced causal path is above the threshold *MIN_PROB_THRES_PATH*. We set the *MIN_PROB_THRES_PATH* empirically based on the benign system data such that only gadgets which have a higher regularity score than the original system event and have the capability of subverting [64] LOF as well as VAE detection algorithm are selected.

4 IMPLEMENTATION

AGE-GNN can operate on both Linux and Windows systems but we evaluate mainly Windows as industrial Windows servers are the target of most APT campaigns. System level data is stored in a PostgresSQL database using the Windows ETW framework [3] to collect a subset of system calls relevant to our interested system entities (*i.e.,* file, process and network socket), which include system calls for *(1)* file operations (*e.g.,* `read()`, `write()`, `unlink()`), *(2)*

```
Algorithm 1: GadgetReplacedPaths
  Input: Causal Path $\lambda_{rare} = [e_1, e_2, \ldots, e_n]$; Frequency
         Database, $FreqDB$
  Output: List of gadget replaced causal paths
          $[\lambda_{gadget,1}, \lambda_{gadget,2}, \cdots]$
1 $path \leftarrow [e_1, e_2, \ldots, e_n]$
2 RareEvents = GetRareEventsFromPath($path, FreqDB$)
3 GadgetReplacedPaths ⟵ []
4 for $e_{rare}$ in RareEvents do
5     GadgetPaths = GadgetPaths ∪
      ReplaceRareEventsWithGadget($path$, $FreqDB$,
      $e_{rare}$)
6 return GadgetReplacedPaths
```

```
Algorithm 2: GetRareEventsFromPath
  Input: Causal Path $\lambda_{rare}$=[$e_1$,$e_2$,...,$e_n$]; Frequency Database,
         $FreqDB$
  Output: list of events $[e_{rare,1}, e_{rare,2}, ...]$
1 $path \leftarrow [e_1, e_2, \ldots, e_n]$
2 RareEvents ⟵ []
3 for $e_i$ in path do
4     $score = getRegularityScore(e_i, FreqDB)$
5     if $score > MIN\_PROB\_THRES\_EVENT$ then
6         RareEvents = RareEvents ∪ $e_i$
7 return RareEvents
```

```
Algorithm 3: ReplaceRareEventsWithGadget
  Input: Causal Path, $\lambda_{rare}$=[$e_1$,$e_2$,...,$e_n$]; Frequency Database,
         $FreqDB$; Rare Event, $e_{rare}$
  Output: List of gadget replaced causal paths for a rare
          event[$\lambda_{gadget,1,e_{rare}}$, $\lambda_{gadget,2,e_{rare}}$,...]
1 $path \leftarrow [e_1, e_2, \ldots, e_n]$
2 GadgetReplacedPaths ⟵ []
3 $gadgets = FindGadgets(e_{rare}, FreqDB)$
4 for $gadget$ in $gadgets$ do
5     $new\_path$ = ReplaceRareEdgewithGadget($path$,
       $gadget$)
       $score = getRegularityScore(new\_path, FreqDB)$
6     if $score > MIN\_PROB\_THRES\_PATH$ then
7         GadgetReplacedPaths = GadgetReplacedPaths
          ∪ $new\_path$
8 return GadgetReplacedPaths
```

network socket operations (*e.g.,* `connect()`, `accept()`), *(3)* process operations (*e.g.,* `create()`, `exec()`, `exit()`). Upon receiving system-call sequences from the kernel, the data collection module reconstructs system information by probing the OS and maintaining a modeling state.

The provenance graph creation and path extraction is done in 15k lines of Java code. The event frequency database is implemented in 4k lines of Java code. The event frequency database is populated by host events over a period of 24 hours (*e.g.,* chosen empirically) across several days. During runtime, AGE-GNN queries this database to calculate event frequencies for rare system events and to find suitable replacement gadgets. GadgetReplacedPaths algorithm 1, GetRareEventsFromPath algorithm 2 and ReplaceRareEventsWithGadget algorithm 3 are implemented using Python.

Certain *provenance data preprocessing* were implemented such as path abstraction and socket connection abstraction seen in related works [64] [23] [35] [22]. We trained a *Doc2Vec* model to generate 50 dimensional feature vectors from the causal paths. The feature vectors were used to train ML based detection models such as Local Outlier Factor (LOF) [5] and convolutional autoencoder [4].

The gadget is found using GNN as mentioned in ReplaceRareEventsWithGadget algorithm 3 *line 3.* The GNN backend is written using Tensorflow GNN(TF-GNN) library [59]. We use the Doc2Vec model creation function implemented in the Gensim Library [2] to implement the document embedding model to convert causal paths to feature vectors.

The scikit-learn's sklearn library was used to implement the Local Outlier Factor (LOF) [5]. The Keras library with Tensorflow backend was used to implement the convolutional autoencoder model [4]. AGE-GNN's autoencoder uses four fully connected layers with 50, 10, 10, and 50 neurons respectively. The first two layers are used for encoding and the last two are used for decoding. 100 epochs and L1 regularization was used to prevent overfitting. To tackle the important issue of local minimization of autoencoder and LOF, we trained the autoencoder multiple times and chose the best model.

5 EVALUATION

We seek answers to the following research questions:

RQ1: Detection Evasion. How effective is AGE-GNN in hiding different APT attack stages against ML based behavioral analysis (LOF and autoencoder)? subsection 5.6

RQ2: Ease of Detection. How do different APT attack scenarios compare against each other in terms of ease of detection? subsection 5.7

RQ3: Performance Overhead. How much overhead is incurred by the attacker to find the appriopiate gadget for a particular APT attack stage? subsection 5.8

5.1 Experiment Protocol

The benign program profile was built from low-level system events generated from 86 Windows devices. The Windows devices were a mixture of server and general-purpose hosts to collect different program profiles for multiple use cases. The Windows hosts consisted of students, researchers, developers, and administrators enabling a variety of program usage data to be collected. The benign data collection ran for eleven months from January to November, 2021 and consisted of over 14 TB of system events. The benign profile dataset is larger in magnitude compared to previous works [64]. ProvDetector [64] which collected only 3 months of system events data and did not have a high entropy enviroment as they only focused on enterprise environment.

Table 3: Number of vertex and edges used to create benign profile for system programs.

Applications	Avg # of causal paths	Avg # of total vertices and edges	Avg # of forward vertices and edges	Avg # of backward vertices and edges
System Programs				
acrord32.exe	1957.08	39.58 / 46.51	6.28 / 5.28	33.3 / 41.23
certutil.exe	28162.32	35.09 / 74.54	3.37 / 2.37	31.72 / 72.17
cmd.exe	2462.71	21.99 / 26.71	5.57 / 4.57	16.42 / 22.14
code.exe	10579.16	67.06 / 92.53	16.53 / 15.53	50.53 / 77.0
conhost.exe	4418.39	33.92 / 35.51	2.01 / 1.01	31.91 / 34.5
cscript.exe	6949.2	52.8 / 65.2	2.0 / 1.0	50.8 / 64.2
cvtres.exe	24.5	11.5 / 10.0	2.0 / 1.0	9.5 / 9.0
msiexec.exe	11473.0	74.0 / 96.0	2.0 / 1.0	72.0 / 95.0
netsh.exe	4181.39	34.18 / 44.14	2.31 / 1.31	31.87 / 42.83
powershell.exe	1429.78	33.28 / 38.69	5.06 / 4.06	28.22 / 34.63
sc.exe	270.05	10.06 / 9.31	2.89 / 1.89	7.17 / 7.42
svchost.exe	4.54	5.62 / 3.62	3.31 / 2.31	2.31 / 1.31
tasklist.exe	123.0	14.33 / 19.67	2.0 / 1.0	12.33 / 18.67
taskmgr.exe	3621.88	42.83 / 50.33	2.0 / 1.0	40.83 / 49.33
userinit.exe	77.0	89.34 / 87.34	86.67 / 85.67	2.67 / 1.67
winlogon.exe	30.75	34.5 / 32.5	31.0 / 30.0	3.5 / 2.5
Average	**4735.30**	**37.51** / **45.78**	**10.94** / **9.93**	**26.57** / **35.85**
User Programs				
acrobat.exe	92.08	11.35 / 14.32	2.46 / 1.46	8.89 / 12.86
chrome.exe	3028.15	50.17 / 58.69	2.01 / 1.01	48.16 / 57.68
discord.exe	2228.39	61.42 / 76.29	26.03 / 25.03	35.39 / 51.26
excel.exe	33113.53	131.54 / 158.53	35.67 / 34.60	95.87 / 123.93
explorer.exe	9119.99	327.03 / 371.69	315.41 / 355.87	11.62 / 15.82
firefox.exe	9792.64	78.66 / 91.53	21.15 / 20.44	57.51 / 71.09
javaw.exe	22500.40	71.82 / 123.45	10.58 / 20.76	61.24 / 102.69
notepad.exe	34141.24	92.07 / 144.66	2.22 / 1.19	89.85 / 143.47
osql.exe	415.29	26.15 / 32.29	3.29 / 2.29	22.86 / 30.0
outlook.exe	42796.90	219.80 / 267.90	90.00 / 88.90	129.80 / 179.00
pycharm64.exe	850.38	28.51 / 31.13	7.02 / 6.33	21.49 / 24.8
python.exe	248.67	10.93 / 11.17	3.0 / 2.0	7.93 / 9.17
slack.exe	3242.43	96.71 / 119.57	30.57 / 29.57	66.14 / 90.0
word.exe	3341.0	58.88 / 72.0	26.38 / 25.38	32.5 / 46.62
Average	**11779.36**	**90.36** / **112.38**	**41.13** / **43.92**	**49.23** / **68.46**

Table 4: Number of vertex and edges used to create APT attack with and without gadget profiles.

Applications	Avg # of causal paths	Avg # of total vertices and edges	Avg # of forward vertices and edges	Avg # of backward vertices and edges
APT Kill Chain Scenario				
Initial Access	182.61	54.2 / 55.89	11.03 / 10.08	43.17 / 45.81
Establish a Foothold	221.65	72.3 / 74.4	27.56 / 26.63	44.74 / 47.77
Privilege Escalation	743.36	117.89 / 122.0	70.96 / 70.25	46.93 / 51.75
Deepen Access	850.33	102.05 / 106.14	57.1 / 56.43	44.95 / 49.71
Exfiltration	471.67	127.45 / 129.03	29.21 / 27.33	98.24 / 101.70
Average	**493.92**	**94.78** / **97.48**	**39.17** / **38.14**	**55.61** / **59.35**
APT Kill Chain Scenario with Gadget				
Initial Access	32.50	29.1 / 27.5	23.8 / 22.8	5.3 / 4.7
Establish a Foothold	54.51	40.22 / 38.37	31.71 / 30.71	8.51 / 7.66
Privilege Escalation	613.23	38.45 / 39.66	21.97 / 20.97	16.48 / 18.69
Deepen Access	88.86	20.77 / 19.12	5.52 / 4.52	15.25 / 14.6
Exfiltration	90.55	23.41 / 22.83	6.43 / 5.43	16.98 / 17.4
Average	**175.93**	**30.39** / **29.50**	**17.89** / **16.89**	**12.50** / **12.61**

To evaluate AGE-GNN against ML-based anomaly detection such as [37, 64], we implemented the Local Outlier Factor (LOF) algorithm using the same configurations as [64]. Likewise, to evaluate the capability of AGE-GNN in adversarial example generation, we implemented an advanced ML-based anomaly detection model, namely a convolutional autoencoder [11, 63, 75]. Our autoencoder has similar shape, optimizer, and training epochs as [63]. We chose [63]'s autoencoder configuration as the baseline because their evaluation was performed recently and in a similar diverse environment to us, plus they had good results.

The frequency database and event database were constructed and stored on a Linux server running PostgreSQL. Provenance graph construction and path extraction was done with the assistance of SQL queries issued by the causality tracker. Five servers running in parallel were used to generate the provenance graphs as provenance graph generation can be be done in parallel. We trained AGE-GNN's behavior models (*e.g.,* LOF and autoencoder) on a Windows host machine with an Intel Core i7-11375 Quad-Core Processor (4.8 GHz) and 16 GB RAM running Windows 11 OS equipped with a 6 GB RTX 3060 graphics card. The detection was also performed on the same machine.

5.2 Benign Profile Creation

We selected 30 system programs from our event database that are commonly used in APT campaigns from previous studies [35, 64]. The list can be found in Table 3. The benign dataset consists of two kinds of programs: system programs used by the OS for system functionalities and user programs used in everyday general workloads.

The provenance graphs generated from the benign system programs consisted of 4735.30 causal paths, 37.51 vertices and 45.78 edges on average (Table 3). The provenance graph generated from the benign user application consisted of 11779.36 causal paths, 90.36 vertices and 112.38 edges on average (Table 3). For interested readers who are looking for more details regarding the benign dataset statistics please refer to Table 3.

For each program, we generated a provenance graph based on the relevant system events and extracted the causal paths. We combined the causal paths generated by day and selected the top 10k causal paths for the benign dataset. Finally, we combined all program specific benign datasets to create the complete benign dataset. The complete benign dataset contained over 5 million causal paths. These causal paths serve two purposes. Firstly, they are used to create the frequency database used in gadget mining. Secondly, they are used to train the ML-based detection models with 500k causal paths per application and 10k causal paths used for testing input.

5.3 Malicious Profile Creation: APT and APT with gadget

The anomaly dataset contains two datasets: APT attack campaign and APT attack campaign with gadget. The APT attack campaign with gadget consists of the APT attack that was conducted using the gadget mentioned in column *events in APT Stages using Gadget Chain 1* Table 2. The anomaly dataset was created in a similar fashion to the benign dataset, except a key difference, *e.g.,* the top 15% were selected from the anomalous paths using the path selection algorithm mentioned in [64]. The value of 15 was chosen as the entire provenance graph is not an accurate representation of the APT attack campaign and the majority of the anomalous graphs are still benign; this is a reasonable assumption as mentioned in previous works [64].

We used a malicious testbed to collect labeled datasets necessary for prediction tasks by running five different kinds of APT attack stages with and without gadgets. The anomaly dataset consists of causal paths that were extracted after we conducted a representative attack, *i.e.,* email phishing proposed in prior works [36, 68]. We coordinated the attack vectors to comprise the end-to-end attack campaign (TTPs) referred by the MITRE ATT&CK framework [48].

We conducted our experiment for each of the APT attack stages(*e.g., Initial Access*, *Establish a Foothold*, *Privilege Escalation*, *Deepen Access* and *Exfiltration*), with and without gadgets, five times each. In total, we ran 25 experiments for APT attack and APT attack with gadget with 50 total experiments. The provenance graphs for *APT Kill Chain Scenario* contains an average of 493.92 causal paths, 94.78 vertices and 97.48 edges. The provenance graphs for *APT Kill Chain Scenario with Gadget* has an average of 175.93 causal paths, 30.39 vertices and 29.50 edges. Interested readers can refer to Table 4 for further details. The provenance graphs for the *APT Kill Chain Scenario* with gadgets are smaller because the attacks focus on a particular APT stage and thus create less noise.

5.4 Evaluation Methodology

The evaluation is split into two sections: *APT Kill Chain Scenario* and *APT Kill Chain Scenario with Gadget. APT Kill Chain Scenario* considers an APT attack conducted with programs whereas in the *APT Kill Chain Scenario with Gadget*, the attacker uses curated gadgets to bypass ML-based detection. To evaluate our adversarial framework, we use two ML-based anomaly detection models: Local Outlier Factor (LOF) and autoencoder model.

After generating the benign and malicious datasets, we extract the feature vectors using a *Doc2Vec* model. The *Doc2Vec* model used was trained with the benign deployment dataset but it is used for both benign and anomalous feature vector inference. The extracted feature vectors are then used for training both the LOF and autoencoder model. The malicious dataset along with the benign testing dataset was used for model evaluation. Similar to previous works such as [37, 64], we selected precision, recall and F1 score as our metrics to measure the detection efficacy of the ML models.

We trained the autoencoder using benign program profile data to determine the detection threshold t for the benign dataset. Empirically, we chose the 99th percentile of the reconstruction error to maximize the detection accuracy and subsequent F1 score. The threshold value is a hyper tuned parameter using the benign validation dataset. We believe our unsupervised modeling is a reasonable design choice for security applications where concrete assumptions can't be made about attack behaviors.

5.5 Attack Case: Phishing Email

The phishing email attack can be classified according to MITRE ATT&CK framework into five major TTPs: Initial Access (TA0001) [46], Establishing a Foothold [43], Privilege Escalation (TA0004) [49], Deepen Access [44], and Exfiltration (TA0010) [42]. For our experiment, we were able to conduct the five TTPs using the well-known penetration testing framework [38]. The attack involves an attacker crafting a malicious macro (*e.g.,* malware named java.exe) embedded attachment (*e.g.,* Excel document) which is sent to a victim through email. The first TTP, Initial Access, is realized when the victim downloads and opens the email attachment.

The malicious macro starts a new malware process called java.exe which opens an initial connection with the attacker's C&C. The attacker runs specialized software [45] to get the password hashes and LSA secrets. The second TTP, Establishing a Foothold, is realized here. The attacker use the password hashes and username to attack the domain controller host running on port 445 as well as try to exploit it. Once the Windows active directory (*e.g.,* domain controller) is exploited, the attacker can easily open a privileged command prompt (*e.g.,* cmd.exe) using the credentials thus completing the third TTP of Privilege Escalation.

After launching a privileged shell, the attacker conducts a network scan to find and connect to the SQL server. The fourth TTP, Deepen Access, is realized here as the attacker tries to penetrate the enterprise and infect more victims. Once the SQL server is located, the attacker executes a malicious visual basic script file using cscript.exe to creates another malware instance. This malware process executes SQL commands in the privileged shell using osql.exe and sqlservr.exe. Finally, the attacker downloads the database dumps generated by the command completing the fifth TTP of exfiltration.

5.6 ML detection: LOF and autoencoder

The detection accuracy of the APT attack stages by different ML-based behavioral analysis is measured in precision, recall, and F1 score. The detection results are shown in Table 5. The precision of both LOF and autoencoder ranges from 0.90 to 1 under *APT Kill Chain Scenario* meaning that both models correctly detect the anomalous causal paths. Under *APT Kill Chain Scenario with Gadget*, the precision for LOF is 1.0 for all APT attack stages meaning that the LOF model correctly detects the anomalous causal paths. The precision for autoencoder ranges from 0.80 to 1 meaning that some paths were marked as false positives when gadgets were used. This is a side effect of the gadget's noise. That is, although it appears to be benign, chaining the benign behaviors together creates a less common causal path. Consequently, using gadgets does not always complement the benign profile completely. For example, acrord32.exe can sometimes called adobe_updater.exe which is rare if this event is chained with excel.exe opening acrord32.exe. Thus, some malicious causal paths will still be detected even with the use of gadgets. We will discuss later why using gadgets cannot hide the attack completely.

The autoencoder's recall and F1 scores under *APT Kill Chain Scenario* range from 0.99 to 1.0 and 0.95 to 1.0, respectively. This means that the autoencoder is extremely proficient in detecting anomalous causal paths. The LOF model's recall and F1 scores under *APT Kill Chain Scenario* range from 0.75 to 0.82 and 0.85 to 0.90 respectively. The relatively low recall and F1 scores compared to the autoencoder means that LOF cannot easily detect anomalous causal paths. However, this is expected as the APT attack is a stealthy attack vector and the LOF model is a relatively weak ML-based behavior model compared to autoencoder. It should be noted that [64], which uses a LOF model, needs only t causal paths

Table 5: ML-based behavior models' (LOF and autoencoder) detection results for APT attack and APT attack with gadget.

MITRE ATT&CK TTP	Precision	Recall	F1
APT Kill Chain Scenario			
LOF			
Initial Access	1.0	0.75	0.85
Establish a Foothold	1.0	0.80	0.89
Privilege Escalation	1.0	0.76	0.86
Deepen Access	1.0	0.77	0.87
Exfiltration	1.0	0.82	0.90
Autoencoder			
Initial Access	1.0	0.99	1.0
Establish a Foothold	0.95	0.99	0.97
Privilege Escalation	1.0	0.99	0.99
Deepen Access	0.90	0.99	0.95
Exfiltration	0.99	1.0	0.99
APT Kill Chain Scenario with Gadget			
LOF			
Initial Access	1.0	0.02	0.05
Establish a Foothold	1.0	0.08	0.15
Privilege Escalation	1.0	0.04	0.08
Deepen Access	1.0	0.07	0.13
Data Exfiltration	1.0	0.20	0.34
Autoencoder			
Initial Access	0.88	0.09	0.16
Establish a Foothold	0.98	0.20	0.33
Privilege Escalation	1.0	0.09	0.17
Deepen Access	0.80	0.24	0.36
Exfiltration	1.0	0.24	0.39

to be marked as anomalous for the entire provenance graph to be considered anomalous. LOF's F1 score ranges from 0.85 to 0.90 and autoencoder's F1 ranges from 0.95 to 1, suggesting that these *APT Attack Kill Chain* graphs would be marked as malicious and that the APT attack would be detected by both LOF and autoencoder models.

The autoencoder's recall and F1 scores range from 0.09 to 0.24 and 0.16 to 0.39 respectively against the *APT Kill Chain with Gadget* suggesting that the detection capability of the autoencoder model was significantly decreased. LOF's recall and F1 scores range from 0.02 to 0.20 and 0.05 to 0.34, respectively which also suggest that the anomaly detection capability of LOF against *APT Kill Chain with Gadget* was decreased to a great extent. The low F1 scores of LOF and autoencoder for the *APT Kill Chain with Gadget* scenario suggests that there are less than *t* causal paths marked as anomalous in the APT campaign thus allowing the APT attack campaign to remain undetected.

5.7 Relative Ranking of APT attack stages

We considered the research question **RQ2** so attackers can know which APT attack stages have the greatest probability of detection (*e.g.,* Exfiltration). Additionally, we pose the same question to allow enterprise security analysts to identify the attack stages that are easier to detect (*e.g.,* Initial Access and Privilege Escalation).

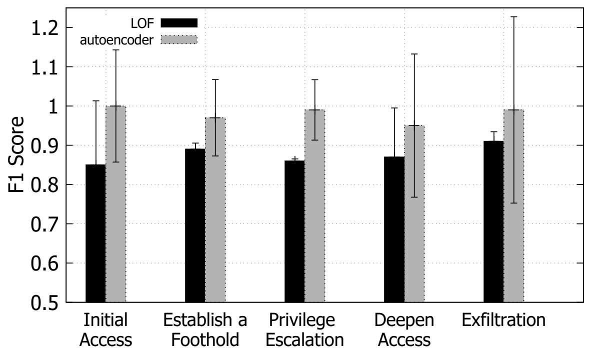

Figure 4: F1 score for APT attack stage without gadget detection.

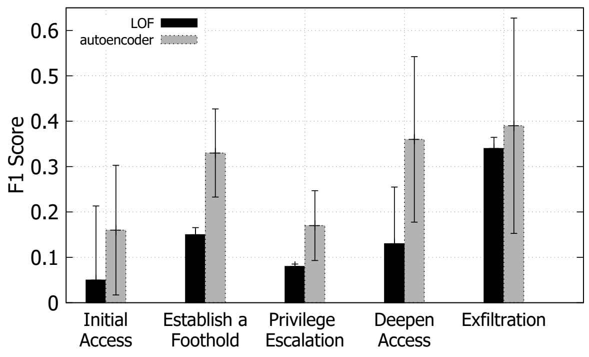

Figure 5: F1 score for APT attack stage with gadget detection.

Using Figure 4 and Figure 5, we can see that the F1 score of Exfiltration is regularly higher than all other APT attack cases for both ML models across both scenarios. Therefore, using the F1 scores in Figure 4 and Figure 5, we can list the APT attack stages from easiest to hardest to detect: exfiltration, deepen access, establish a foothold, privilege escalation, and initial access.

A typical APT attack scenario for initial access involves the user downloading a malicious file attachment and opening it. Then, unbeknown to the user the malicious attachment creates a new malicious process in the system. The new process then connects to the attacker's C&C. This scenario is very similar to the benign behavior of opening an email, downloading an attachment, and then creating a new process for printing the attachment through a shared network printer. Therefore, both ML models fail to classify the initial access cases with high accuracy as the model finds it hard to distinguish between normal and malicious behavior. Similarly, privilege escalation use attack vectors which are similar to administrative work performed by system or network admins in an enterprise. Using the same enterprise scenario, it can be explained why establish a foothold and deepen access is easier to detect as it is unusual for enterprise users to perform network scans, port scans, or create database connections. Even though it is possible for those actions to happen in the normal use of the system, for example, in the case of network administrators and data engineers, but they are still rare compared to the rest of the enterprise environment's workload.

The probability of detecting the Exfiltration TTP is highest even with gadget use as seen in the high F1 score for both LOF and autoencoder in the *APT Kill Chain Scenario with Gadget* and LOF in *APT Kill Chain Scenario*. This is because lots of temporary files are created and a visual basic script is executed with a high number of socket connections in Exfiltration TTP. These actions create a lot of noise in the provenance graph, as many events in the system are captured by the provenance graph and detected.

5.8 AGE-GNN Performance Overhead

The runtime overhead of AGE-GNN was measured to answer **RQ3**. The overhead for AGE-GNN can be separated into three stages: data mining, gadget finding and detection. The first stage contained the highest overhead of three hours used to build the provenance graphs of the programs (Table 3) and APT attack stages (Table 4) as well as extract the relevant paths. For each event, the graph creation and extraction process was around one minute. The large volume of the benign paragraphs(*e.g.,* eleven months of data) contributed most to the overhead for this stage. We parallelized the provenance graph creation process across five servers.

The second highest overhead comes from gadget finding stage. The gadget finding stage has three sub-stages: frequency database creation, transition matrix creation, and gadget finding. The transition matrix creation stage took the longest at one hours fifteen minutes to search through 1200 relevant programs in the event database. Frequency database creation and gadget finding are relatively quicker taking around seven minutes and two minutes respectively. The large overhead from stage two is proportional to the dataset meaning a smaller overhead for a shorter timeframe of data.

The last stage, detection, incurs the smallest overhead at around two and five minutes for both training and detection stages in the LOF and autoencoder models. An additional common overhead of three minutes exists to convert the causal paths to feature vectors through the *Doc2Vec* model. However, note that our experiments were conducted in a desktop with a lower-end graphics card installed as well as the program being single threaded for stage two and three. The total workload can be further reduced with additional parallelization for gadget finding by AGE-GNN.

6 RELATED WORKS

6.0.1 Data Generation using Generative models. Recent works have used LOF [23, 64], Variational Autoencoder (VAE) [29, 30, 51] and Generative Adversarial Network (GAN) [62] to generate artificial training examples to prevent data shortage during training. Therefore, in-case there is not much data available for a new system or device configuration, the model cannot be trained correctly.

Some recent works have generated grey scale image from a malware elf from VAE model [34] [26] [12]. The work [26] first disassemble malware binaries and train the VAE model using the execution sequences. Then they used the decoder to generate an disassembled binaries that can be used as an adversarial example. These works have not impacted the security domain in creating live malware samples or valid programs behavior in the system that would make them indistinguishable from a real world example.

6.0.2 Adversarial Example generation using Generative models. The prominence rise of generative models such as VAEs [34] has established the field of adversarial example generation [18]. Image classification and object detection [16] has both benefited from adversarial examples as these works showed how the models could be attacked. Therefore, the researcher were more well suited to mitigate the risk or retain the model with more regularization or changing the activation function function such that the models becomes secure against stronger adversaries.

6.0.3 Adversarial Example generation using deep structures. Many real life scenarios can be modelled as graph such as traffic pattern, molecules and drone path. Many VAE [29] are using GNN [24] as the encoder and decoder's deep structure as they are the most suited for digesting the graph structure. The works have mostly focused on generating data that has the potential of excavating new relationships between graph structures and formulating sub-graphs that by themselves are adversarial examples.

The research on adversarial machine learning has gained huge momentum since Kurakin et al. [32] first proposed the attack on an established model for image recognition problem. Since then, numerous works [19, 50, 51] have followed that propose a variety of approaches to deceive the ML models built for different domains, using different modeling approaches. Coinciding with the recent proliferation of research on graph learning, the community has also come to look at the adversarial graph learning problem [76]. For example, et al. [60] has proposed adversarial training to increase the robustness of ML models.

7 DISCUSSION

7.0.1 Generalization of Adversarial ML security. Our target security defense is built on provenance analysis [28] and its graph representation. Although our evaluation focuses on Windows server system events, AGE-GNN is applicable to both Linux and Windows operating systems requiring only system event logs to build the necessary provenance graphs. Our current implementation and evaluation mainly consider Windows-based devices. However, our approach is general and applicable to devices with other operating systems. For example, the Linux operating system also has an auditing system to log system events [53]. Along the same, in the case where Windows audit is not available, we can use other data collection channels. For example, Sysdig [57] and Linux DTrace [14]. However, these alternatives have code maturity or coverage issues.

7.0.2 Site and Execution History Dependencies. The benign program profile is dependent on the site and execution history of the hosts. In our work, the hosts (e.g. students, researchers, developers, and administrators) had a variety of settings (e.g. home, school, enterprise). Thus, our results depended upon the host's role and setting as the host affects the volume and nature of collected system events. This in turn affects the activities that are classified as benign (i.e. the most frequent activities performed by the hosts will be classified as benign).

7.0.3 Other ML-embedding Approaches. Our work only implements a convolutional autoencoder and a LOF, but other ML models exist that can detect anomalous behavior [10, 15, 21]. Chaudhary et al. [10] utilize a Graph Neural Network (GNN) to detect anomalies in social networks. In their GNN, they classify anomalous node behavior with two criteria: *(1)* the node is connected to very few other nodes, and *(2)* the node is connected to a majority of the network [10]. Hasan et al. [21] results show that the random forest model is effective at classifying attacks on IoT systems. Erfani et al. [15] proposed a hybrid anomaly detection pipeline that utilizes a deep belief network (DBN) to extract important features from data which is then used as training input for a one-class support vector machines (SVM). With this approach, they are able to overcome the limitations of the SVM (e.g. memory and time intensive to train) and gain its advantages (e.g. insensitive to noise in the training data) [15].

7.0.4 Explainable Security Model. ML-based models are known to be unexplainable and posed important lines of research problems. This is also the case for ML-based security solutions as many ML-based security solutions are unable to provide explainable reasons for their detection results. In contrast, AGE-GNN is able to specifically identify the causal paths that were classified as anomalous through analysis of the frequency database and provide regularity score (*e.g.,* a measure of rarity). Additionally, the data and code of our work will be public for future research.

8 FUTURE WORK

The detection pipeline of AGE-GNN consists of autoencoder and LOF. We plan on implementing graph neural network based detection. So, that we can show that state of the art GNN implementation such as Chaudhary et al. [10] is also susceptible to AGE-GNN's adversarial examples.

AGE-GNN utilizes provenance graphs built from kernel-level events. As such, they provide course-grained information about the system's behavior (e.g. does not specify what parts of a file are read, only that it was read). A finer approach to system event data collection will result in better models. Furthermore, even with course-grained information, provenance graphs still explode in size (i.e. dependence explosion). Post-processing of graphs to reduce size while still maintaining the causal dependencies in the graph can reduce the overhead of storage space for the data. Utilizing more precise system behaviors may also produce better results.

9 CONCLUSION

We presented a novel adversarial attack generation framework that autonomously synthesizes stealthy attack vectors to evade ML-based security solutions. Designed as a purely data-driven framework, AGE-GNN successfully demonstrates its ability to formulate advanced attacks and bypass the intrusion detection system of a secure network. Our analysis draws important insights of ML-based detection metrics with regards to APT attacks showing the evasiveness of each step in the attack. This insight can be used to strengthen security defenses. We are confident that AGE-GNN will make a substantial contribution to the future of ML-based security systems as an innovation to pioneer future adversarial security detection research.

REFERENCES

[1] Cyber Kill Chain® | Lockheed Martin. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html. (????). (Accessed on 07/20/2021).

[2] 2019. gensim: Topic modelling for humans. https://radimrehurek.com/gensim/index.html. (2019).

[3] 2021. Event Tracing. https://docs.microsoft.com/en-us/windows/desktop/ETW/event-tracing-portal. (2021).

[4] 2021. Keras: the Python deep learning API. https://keras.io/. (2021).

[5] 2021. scikit. https://scikit-learn.org/stable/l. (2021).

[6] Mnahi Alqahtani, Hassan Mathkour, and Mohamed Maher Ben Ismail. 2020. IoT botnet attack detection based on optimized extreme gradient boosting and feature selection. *Sensors* 20, 21 (2020), 6336.

[7] Tal Be'ery. Target's Data Breach: The Commercialization of APT. https://www.securityweek.com/targets-data-breach-commercialization-apt. (????). (Accessed on 11/09/2021).

[8] Leyla Bilge and Tudor Dumitraş. 2012. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security.* 833–844.

[9] Roland Burks, Kazi Aminul Islam, Yan Lu, and Jiang Li. 2019. Data augmentation with generative models for improved malware detection: A comparative study. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON).* IEEE, 0660–0665.

[10] Anshika Chaudhary, Himangi Mittal, and Anuja Arora. 2019. Anomaly Detection using Graph Neural Networks. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon).* 346–350. https://doi.org/10.1109/COMITCon.2019.8862186

[11] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. 2018. Autoencoder-based network anomaly detection. In *2018 Wireless Telecommunications Symposium (WTS).* 1–5. https://doi.org/10.1109/WTS.2018.8363930

[12] Zhihua Cui, Fei Xue, Xingjuan Cai, Yang Cao, Gai-ge Wang, and Jinjun Chen. 2018. Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics* 14, 7 (2018), 3187–3196.

[13] Michael K Daly. 2009. Advanced persistent threat. *Usenix, Nov* 4, 4 (2009), 2013–2016.

[14] DTrace 2018. DTrace. https://en.wikipedia.org/wiki/DTrace. (2018).

[15] Sarah M Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. 2016. High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. *Pattern Recognition* 58 (2016), 121–134.

[16] Volker Fischer, Mummadi Chaithanya Kumar, Jan Hendrik Metzen, and Thomas Brox. 2017. Adversarial examples for semantic image segmentation. *arXiv preprint arXiv:1703.01101* (2017).

[17] google. Netsh Command Syntax, Contexts, and Formatting. https://www.google.com/chrome/. (????). (Accessed on 11/29/2021).

[18] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *European symposium on research in computer security.* Springer, 62–79.

[19] Shixiang Gu and Luca Rigazio. 2014. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068* (2014).

[20] You Joung Ham, Daeyeol Moon, Hyung-Woo Lee, Jae Deok Lim, and Jeong Nyeo Kim. 2014. Android mobile application system call event pattern analysis for determination of malicious attack. *International Journal of Security and Its Applications* 8, 1 (2014), 231–246.

[21] Mahmudul Hasan, Md Milon Islam, Md Ishrak Islam Zarif, and MMA Hashem. 2019. Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. *Internet of Things* 7 (2019), 100059.

[22] Wajih Ul Hassan, Lemay Aguse, Nuraini Aguse, Adam Bates, and Thomas Moyer. 2018. Towards scalable cluster auditing through grammatical inference over provenance graphs. In *Network and Distributed Systems Security Symposium.*

[23] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NoDoze: Combatting threat alert fatigue with automated provenance triage.. In *NDSS.*

[24] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 1857–1867.

[25] Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. 2020. Intrinsic certified robustness of bagging against data poisoning attacks. *arXiv preprint arXiv:2008.04495* (2020).

[26] Kesav Kancherla and Srinivas Mukkamala. 2013. Image visualization based malware detection. In *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS).* IEEE, 40–44.

[27] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. 2017. Certified malware: Measuring breaches of trust in the windows code-signing pki. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* 1435–1448.

[28] Samuel T King and Peter M Chen. 2003. Backtracking intrusions *(USENIX Symposium on Operating Systems Design and Implementation (OSDI)).* https://doi.org/10.1145/945445.945467

[29] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[30] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. 2016. Deep learning for classification of malware system call sequences. In *Australasian joint conference on artificial intelligence.* Springer, 137–149.

[31] P Arun Raj Kumar and S Selvakumar. 2011. Distributed denial of service attack detection using an ensemble of neural classifier. *Computer Communications* 34, 11 (2011), 1328–1341.

[32] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236* (2016).

[33] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. 1188–1196.

[34] Xinbo Liu, Jiliang Zhang, Yaping Lin, and He Li. 2019. ATMPA: attacking machine learning-based malware visualization detection methods via adversarial examples. In *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[35] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. 2018. Towards a Timely Causality Analysis for Enterprise Security.. In *NDSS*.

[36] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. 2016. Protracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting.. In *NDSS*.

[37] Wataru Matsuda, Mariko Fujimoto, and Takuho Mitsunaga. 2018. Detecting apt attacks against active directory using machine leaning. In *2018 IEEE Conference on Application, Information and Network Security (AINS)*. IEEE, 60–65.

[38] metasploit. metasploit. https://www.metasploit.com/. (????). (Accessed on 11/29/2021).

[39] microsoft. Execute SQL Statements. https://docs.microsoft.com/en-us/windows/win32/msi/execute-sql-statements. (????). (Accessed on 11/29/2021).

[40] microsoft. Netsh Command Syntax, Contexts, and Formatting. https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts. (????). (Accessed on 11/29/2021).

[41] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. 2019. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1137–1152.

[42] MITRE. Exfiltration. https://attack.mitre.org/tactics/TA0010/. (????). (Accessed on 11/29/2021).

[43] MITRE. gsecdump. https://attack.mitre.org/software/S0008/. (????). (Accessed on 11/29/2021).

[44] MITRE. gsecdump. https://attack.mitre.org/techniques/T1534/. (????). (Accessed on 11/29/2021).

[45] MITRE. gsecdump. https://attack.mitre.org/software/S0008/. (????). (Accessed on 11/29/2021).

[46] MITRE. Initial Access. https://attack.mitre.org/tactics/TA0001/. (????). (Accessed on 11/29/2021).

[47] MITRE. MITRE ATT&CK®. https://attack.mitre.org/. (????). (Accessed on 07/23/2021).

[48] MITRE. MITRE ATT&CK®. https://attack.mitre.org/. (????). (Accessed on 07/23/2021).

[49] MITRE. Privilege Escalation. https://attack.mitre.org/tactics/TA0004/. (????). (Accessed on 11/29/2021).

[50] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2574–2582.

[51] Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 427–436.

[52] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. 2007. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys (CSUR)* 39, 1 (2007), 3–es.

[53] Redhat. 2017. The Linux audit framework. (2017). https://github.com/linux-audit/.

[54] Kshira Sagar Sahoo, Bata Krishna Tripathy, Kshirasagar Naik, Somula Ramasubbareddy, Balamurugan Balusamy, Manju Khari, and Daniel Burgos. 2020. An evolutionary SVM model for DDOS attack detection in software defined networks. *IEEE Access* 8 (2020), 132502–132513.

[55] David E Sanger. U.S. Said to Find North Korea Ordered Cyberattack on Sony. *The New York Times* (????). (Accessed on 11/09/2021).

[56] Lewis Smith and Yarin Gal. 2018. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533* (2018).

[57] sysdig 2018. Sysdig. https://sysdig.com/. (2018).

[58] Yong Tang and Shigang Chen. 2007. An automated signature-based approach against polymorphic internet worms. *IEEE Transactions on Parallel and Distributed Systems* 18, 7 (2007), 879–892.

[59] tfgnn 2018. Introducing TensorFlow Graph Neural Networks. https://blog.tensorflow.org/2021/11/introducing-tensorflow-gnn.html. (2018).

[60] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204* (2017).

[61] Nikos Virvilis and Dimitris Gritzalis. 2013. The big four-what we did wrong in advanced persistent threat detection?. In *2013 international conference on availability, reliability and security*. IEEE, 248–254.

[62] Binghui Wang and Neil Zhenqiang Gong. 2019. Attacking graph-based classification via manipulating the graph structure. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2023–2040.

[63] Chao Wang, Bailing Wang, Hongri Liu, and Haikuo Qu. 2020. Anomaly detection for industrial control system based on autoencoder neural network. *Wireless Communications and Mobile Computing* 2020 (2020).

[64] Q. Wang, Wajih Ul Hassan, D. Li, K. Jee, X. Yu, Kexuan Zou, J. Rhee, Z. Chen, Wei Cheng, C. Gunter, and H. Chen. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In *NDSS*.

[65] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A Gunter, et al. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis.. In *NDSS*.

[66] Xin Wang and Siu Ming Yiu. 2016. A multi-task learning model for malware classification with useful file access pattern from API call sequence. *arXiv preprint arXiv:1610.05945* (2016).

[67] Marc Welz and Andrew Hutchison. 2001. Interfacing trusted applications with intrusion detection systems. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 37–53.

[68] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. 2016. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 504–516.

[69] Jin Ye, Xiangyang Cheng, Jian Zhu, Luting Feng, and Ling Song. 2018. A DDoS attack detection method based on SVM in software defined network. *Security and Communication Networks* 2018 (2018).

[70] Zong-Han Yu and Wen-Long Chin. 2015. Blind false data injection attack using PCA approximation method in smart grid. *IEEE Transactions on Smart Grid* 6, 3 (2015), 1219–1226.

[71] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 371–372.

[72] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.

[73] Qinghua Zhang and Douglas S Reeves. 2007. Metaaware: Identifying metamorphic malware. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, 411–420.

[74] Zhiqiang Zhong, Cheng-Te Li, and Jun Pang. 2020. Hierarchical Message-Passing Graph Neural Networks. *arXiv preprint arXiv:2009.03717* (2020).

[75] Chong Zhou and Randy C Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 665–674.

[76] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2847–2856.