Volt Typhoon: Addressing the Threat to National Security

Nathan Wishne
Cyber Risk-Management
Drury University
Springfield, United States
nwishne@drury.edu

Shannon McMurtrey
Associate Professor of Cyber Risk-Management
Drury University
Springfield, United States
smcmurtrey@drury.edu

Abstract— **People's Republic of China's state-sponsored hacking group known as Volt Typhoon has used Living off the Land (LotL) techniques to preposition themselves in the Information Technology (IT) systems of US Critical Infrastructure organizations. The advanced persistent threat has a goal of disrupting the United States in the event of a future conflict. This project aims to analyze the vulnerabilities being exploited, investigate the underlying motives behind their attacks, and evaluate the possible role of machine learning in detecting and mitigating the risks presented by the group.**

Keywords—Living off the Land (LotL), Machine Learning (ML), Anti-Virus (AV), Information Technology (IT)

I. INTRODUCTION

Cyber-warfare has been a hot topic since its inception, drawing attention from the public eye for roughly two decades now. The fact that conflicts can now be waged and fought behind computer screens has added a whole new front for statecraft. In this time, state-sponsored hacking has grown and been adopted as a strategy by nearly all the leading countries in the world. Developing groups capable of sophisticated attacks typically related to espionage and destabilization [1]. Whenever tensions or conflicts arise, there are now often cyberattacks that accompany them. The state-sponsored groups are constantly evolving and adopting new techniques to improve the abilities of their operations.

This project is a look into the Tactics, Techniques, and Procedures (TTP) of the Chinese Advanced Persistent Threat (APT) Volt Typhoon. The threat has mainly targeted the United States and its territories in industries that prop up the nation's vital functions. Specifically targeting the energy, communications, water and wastewater, and transportation sectors. Their actions are aimed at compromising systems and maintaining persistence using Living off the Land (LotL) techniques that hide malicious actions behind what looks like normal system activity. A strategy that has lately become far more common in attacks by highly-sophisticated actors including Volt Typhoon [13]. Throughout this project we aim to find answers to a few key questions.

Who is Volt Typhoon?

What makes the APT especially difficult to detect?

What methods can we use to improve our ability to detect their techniques?

To answer the proposed questions this project covers information provided about Volt Typhoon by the joint agencies in their advisories [6][7]. Explaining the groups usage of LotL techniques as well as the other sophisticated tactics for various purposes throughout the cyber kill chain[11]. We will gauge based on previous sophisticated hacks what Volt Typhoon might be capable of in the event that they decide to switch from their current efforts to a destructive attack approach. Considering LotL strategies proves very difficult to detect with many of the current endpoint detection and response (EDR) solutions that are in place today. This project attempts to address the lack of viable detection solutions by testing two machine learning models with different approaches. The models are from Elastic [15] and Adobe [5], and are tested to see how effective their approaches are at detecting Volt Typhoon's various indicators of compromise (IOC).

To further understand the importance of addressing Volt Typhoon, it is important to mention the growing tensions between the United States and China. While working on this project, a trade war has commenced between the two nations. This is yet another demonstration of the distrust and competition that characterizes this relationship. This more recent dispute coupled with the constantly debated status of Taiwan and its goal of independence indicate no slowdown in cyberattacks any time soon.

II. BACKGROUND

Volt Typhoon was initially disclosed to the public in 2023 with the release of the first joint advisory [6]. However, evidence strongly suggests that the group has had access to the IT environments of critical infrastructure as early as 2019 [7]. The threat is ongoing, and they are still actively compromising systems. Unlike typical cyber espionage of state backed hacking campaigns, Volt Typhoon has clearly been acting with a motive of prepositioning themselves to be destructive in the

event of a conflict between the United States and China. While their actions have been targeted at IT environments, the larger goal of the APT is to move laterally into Operational Technology (OT) [7]. OT describes systems that remotely operate and interact directly with industrial control systems (ICS). This means a compromise of OT systems would also be a severe compromise of United States critical infrastructure [10]. Most organizations use segmentation to segment their IT and OT networks, but sophisticated actors have proven able to get around this measure on occasion.

While this is a current threat, this is not a new idea. OT systems have been compromised and disrupted in many different instances by cyberattacks. Two past instances that are relevant to our current situation are the Russian cyberattacks on Ukraine in 2015 and 2016. Russia's attacks compromised Supervisory Control and Data Acquisition (SCADA) systems. SCADA is a common type of OT interface used to interact with ICS [2]. These SCADA systems were used by Ukraine to monitor and manage their power grid. With the SCADA systems compromised, Russia was able to perform actions such as opening breakers, disabling protective relays, and more to interfere with Ukraine's power grid [1]. These actions lead to blackouts that left millions of people without power [1]. Most importantly, they displayed the disastrous capabilities of sophisticated actors with access to OT. This knowledge of previous attacks, of which there are many, highlights the need for focus on detecting and preventing Volt Typhoon before they gain access to OT.

Accessing OT is the end goal for Volt Typhoon, but there is a lot that goes into compromising and maintaining persistence in IT systems prior to this point. Volt Typhoon uses many of the same processes and tactics observable in nearly any well-done hacking effort [7]. Many of the observed tactics can be categorized into different steps. Such as recon, initial access, lateral movement, and potential impact. These steps each include the use of many different techniques for various purposes.

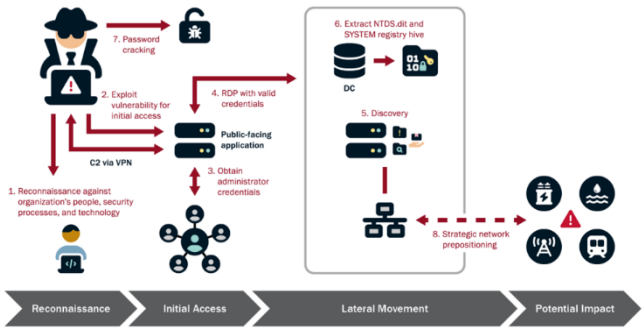

Fig. 1. Volt Typhoon Typical Activity. Source: [7]

As seen in figure 1, Volt Typhoon begins with extensive reconnaissance to find out as much information on their targets as possible. The group uses various open-source intelligence (OSINT) tools such as FOFA, Shodan, and Censys to locate exposed infrastructure. With this information they use a series of multiple different proxy computers chained together to route traffic through to the exposed systems on the perimeter of the infrastructure organizations network [7]. This series of proxies is comprised of a botnet. A botnet is a large group of devices that are connected to the internet and have been infected by malware that lets the attacker control the device remotely. Botnets are a common tool used by hackers to hide identifiable information to stay anonymous. Volt Typhoon uses their botnet, which is made up of mostly devices like residential routers and Internet of things (IOT) devices for this purpose. Making the traffic transmitted in their attacks look as if it originates from a system with an IP address, MAC address, and other identifiable information apart from Volt Typhoon's Command and Control servers (C2).

Using this masked connection, the group leverages both known vulnerabilities and zero-days to exploit the perimeter networking devices and gain initial access [7]. They establish persistence and further anonymity in these exploited devices by encrypting traffic with VPN connections to their C2 servers. [7]

With access to systems on the perimeter of networks, Volt Typhoon then limits their use of malware and adopts the use of LotL techniques [6]. LotL includes the malicious use of native system processes or tools. These native binaries that can be used for both good and bad are referred to as LOLbins [12]. LotL is especially effective when using LOLbins for gaining credential access and discovery [7]. Some of the most common tools and commands observed in use by Volt Typhoon for LotL include the following in figure 2.

cmd	net user	ping	tasklist
certutil	netsh	PowerShell	wevtutil
dnscmd	nltest	quser	whoami
ldifde	netstat	reg query	wmic
makecab	ntdsutil	sysinfo	xcopy

Fig. 2. Commonly used LotL commands and tools by Volt Typhoon

An example of one main way Volt Typhoon uses these commands is to extract credentials of users and group information from a domain controller. To do this they use a combination of the items listed in figure 2. Windows Management Instrument Command-Line (WMIC) is a built-in tool for administrators to access and manage system information. WMIC is used in combination with ntdsutil, a tool for managing Active Directory (AD) domain services, but also has the ability to extract critical information. Volt Typhoon has been observed running the following command in cmd.exe, which uses the tools mentioned to copy the AD database file ntds.dit. [6]

wmic process call create "ntdsutil \"ac i ntds\" ifm \"create full C:\Windows\Temp\Pro

The database ntds.dit contains critical information such as user accounts, password hashes, group policy objects (GPOs), trust relationships, and more about AD. With access to the copied database, Volt Typhoon can then extract the password hashes back to their C2 and attempt to crack them offline using

a number of different techniques like brute-force, dictionary, and rainbow table attacks to name a few. Depending on configurations within the environment, Volt Typhoon could spare the trouble of exfiltrating the hashes to crack them and instead perform a Pass the Hash attack. Pass the Hash is an authentication bypass technique where the actor inputs the hashed password into the password field in a log in attempt. This tactic often works on systems configured for quick seamless accessibility. Often, systems are at risk to this technique if they use single sign on (SSO) and unsalted NTLM hashes.

While most of Volt Typhoon's post-exploitation tactics rely on LotL, there are a few instances where they have used non-LotL tactics. The group has been observed using Impacket python scripts, Mimikatz, ScanLine, and variations of a Fast Reverse Proxy [7]. The following are the use cases for the different non-LotL tools.

- **Impacket** – Scripts used for enumeration and remote command execution. [6]
- **Mimikatz** – Binary used to harvest credentials. [7]
- **ScanLine** – Used for command line port scanning similar to nmap. [7]
- **Fast Reverse Proxy** – Used to expose servers hidden behind a firewall to the internet. [6]

While these tools are all non-native to a Windows environment, it's important to note that it is argued that LotL should include the malicious use of any tool that has a non-malicious practical use [13]. By this definition, their use of Fast Reverse Proxy would be considered LotL because the tool is often used by system administrators to create secure connections to devices.

III. Experiments

A. Data

Useful data is difficult to come by in the field of information security, due to the sensitivity of what might be exposed about networks and the environment that it comes from. Often, giving out such information would expose a company to further attacks. Especially when considering the importance of Critical Infrastructure, it is a risk that is rarely worth taking. This issue creates a situation where locating quality data is a challenging process, and it certainly was a difficult part of this project.

We ended up using two different datasets. They both were created for the purpose of training models to detect malicious activity in two different ways. The first dataset is from Elastic and is called the Elastic Malware Benchmark for Empowering Researchers (Ember) [15]. The second dataset is from Adobe [5]. It comprises benign examples from their own internal systems and malicious examples from the Living off the Land Binaris and Scripts (LOLBAS) project [14]. A community project with the goal to document every known Windows LotL binary, script, and library used in attacks.

1) Ember Dataset

The Ember dataset is an open-source benchmark that was put together with the goal of furthering research in malware detection methods [15]. They first released the dataset in 2017, but to adjust with the ever-changing field they re-released a revised version in 2018. This is the version that we chose to use in this project. They mention the lack of available data as a reason why there seems to be a disproportionately low amount of machine learning research in cybersecurity. This issue has been a common theme in my research, and they address this problem by offering a well-structured solution that is their dataset.

Their dataset includes eight groups of key features extracted from Portable Executable (PE) files. PE files are the standard format for Windows Dynamic Link Libraries (DLLs), executables, and other object code. PE files are important to consider when securing Windows systems because they are how both malware and normal system processes are executed. Their dataset includes features extracted from over 1 million binaries [15]. As shown in figure 3 below, the dataset binaries total to 200k unlabeled, 400k benign, and 400k malicious [15]. During model training, the binaries are labelled as either malicious or benign for the training set, and the test set is left unlabeled. The data is provided in the form of raw features that are human readable, but they provide code that can translate the data to be used for modeling, which will come up later. [15]

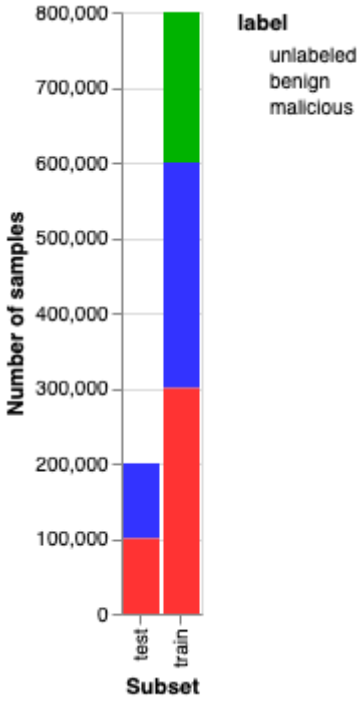

Fig. 3. Ember Data Distribution. Source: [15]

This dataset is particularly relevant to LotL research because it offers a large amount of both malicious and benign examples. Ember presents static features that are present prior to a binary being executed. This provides the opportunity to detect malware before it is executed. A preferred advantage when compared to dynamic models that measure binaries in operation. Static classification is practical for our use of examples where threat actors employ innately benign binaries for malicious purposes.

The hope is that a model from the Ember dataset might be able to recognize malware that includes the use of LotL binaries such as PowerShell.exe or Wmic.exe, and correctly classify it as malicious.

2) Adobe Dataset

Different from Elastic's goal of providing a benchmark, Adobe's dataset was created for the purpose of training their provided model. Their goal is to provide a different and improved alternative to AV solutions for detecting LotL [5]. The data used by Adobe includes both raw Linux Bash and Windows CMD commands. With the observed attacks of Volt Typhoon taking place primarily in Windows environments, the data from and relating to Windows was the focus. They provide in their GitHub the malicious command lines with information on their purpose. They did not disclose the benign commands they used to train their model because they are gathered from the company's own infrastructure. Mentioning the previously covered data sensitivity concerns as the reason. [5]

- **Negative**, or Malicious examples numbered to 1609 examples, with potentially benign uses being filtered and removed [5].
- **Positive**, or benign examples number to approximately ~8,043,391. Considering they don't provide the actual number, but mention 1609 is 0.02% of the total dataset [5].

They choose this distribution to only include 0.02% malicious command-lines because they believe that it properly represents the data one would see in real life [5]. In addition, it promotes a low false-positive rate, an important consideration for building detection models. Figure 4 represents Adobe's data distribution in logarithmic format to adjust for the large data imbalance.

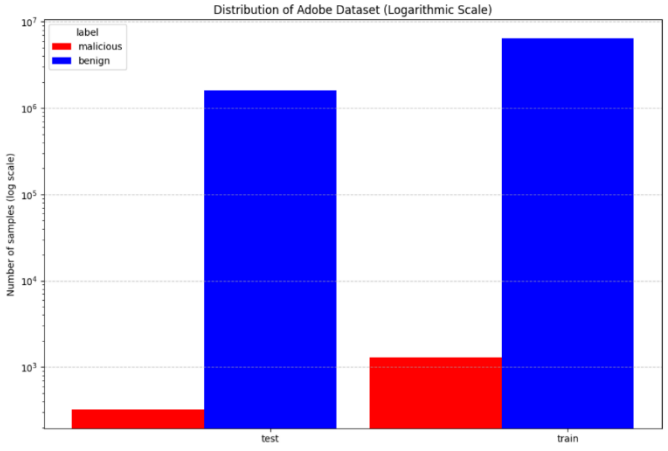

Fig. 4. Adobe Dataset Logarithmic Distribution

B. Tools Used

Through the course of this project, many different tools were used for a few different purposes. The two main purposes being to gain a better understanding of the group with simulation and to recreate the open-sourced models from Elastic [15] and Adobe [5].

Game of Active Directory (GOAD) is a Windows Active Directory environment that is run on multiple networked virtual machines [16]. It comes preconfigured to have many flaws that an attacker might come across when searching for vulnerabilities in a real environment [16]. Many of these vulnerabilities are similar to those abused by Volt Typhoon. To avoid any system damage from emulating attack techniques, it is necessary and important that we experimented with the tactics within a sandboxed environment. GOAD is a great tool for this with the use of virtualization to segment the processes from the host machine. There are a few different versions of GOAD available. Due to hardware limitations we chose to use GOAD-Light, which is very similar to the full lab, but scaled down with the number of virtual machines (3 instead of 5), forests (1 instead of 2), and domains (2 instead of 3) [16]. A diagram of the network with trust relationships and user information can be seen in figure 5.

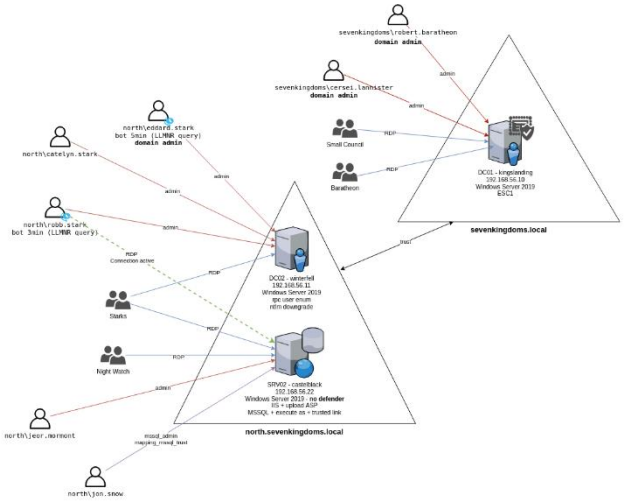

Fig. 5. GOAD-Light Network Diagram. Source: [16]

Jupyterlab is a web-based coding tool from Project Jupyter, a community driven open-source project. Jupyter lab acts like a coding notebook with different sections of code that can be run individually. This makes it a great tool for data science since you can separate code and try many different approaches. Overall, it helped this project to flow well and stay organized throughout. In a practical sense, this project used Jupyterlab for testing models and creating visualizations.

Python was the primary coding language used and required a few dependencies to get the two models running [15][5]. The following is a list of dependencies that were used for core ML uses and data processing, along with the model that required them.

- **Scikit-learn** – Machine learning library used to create classification models [5] [15]
- **Lief** – Tool for PE file feature extraction. [15]
- **LightGBM** – Machine learning framework for gradient boosting and decision trees. [15]

- **NumPy** – Package for numerical computations. [5][15]
- **Pandas** – Library for data manipulation and analysis. [5][15]
- **Matplotlib** – Plotting library for Python, used to create visuals and graphs.
- **SciPy** – Provides advanced mathematical and scientific functions. [5]
- **JobLib** – Provides efficient data workflows. [5]

The following are the dependencies used in relation to natural-language processing for the Adobe model [5].

- **NLTK –** Library for natural-language processing of English. Support methods for classification and tokenization. [5]
- **RegEx –** Tool for matching regular expressions. Compatible with the re python module. [5]

C. Machine Learning Models

Machine learning (ML) falls within the field of Artificial Intelligence due to its ability to take in data and learn from it by recognizing patterns and relationships within the data. This reduces the need for direct programming, as the models teach themselves with the data. This learning process then prepares the model to be used for a given purpose like prediction, classification, or different forms of decision-making. There are many different types of ML, with the main difference typically relating to the nature of how the model is trained. The two most common types of ML are supervised learning and unsupervised learning.

Unsupervised Learning (UL) uses a dataset for training that is not labelled in advance. This forces the model to discover hidden patterns without assistance from labelled data. Without the use of a specific variable for the model to predict, outcomes are more focused on general structure. For this same reason, and with its ability to find potentially hard to notice relationships UL is very good for uses like exploratory data analysis. Some examples of this in practice include clustering, association rule mining, and dimensionality reduction. While technically still quite different, UL is relatively similar to the methods used to train the popular generative AI models.

Supervised Learning (SL) goes in the opposite direction of UL as it uses labeled training data to find connections and patterns that point to a specific result or output of data. With feedback throughout the training process, SL can become very accurate at determining an outcome based on input data. Two common uses for SL are classification and regression. Classification is where a SL model assigns data input to one or more of the classes provided. For example, a fruit classifier might take a picture of fruit as input and classify it as an Apple or Orange based on various identifiable features. Regression models are used to predict continuous numerical values. An example might be a model that could predict sales based on things like advertising expenses and time of year.

It is important to measure the success of a ML model before being put into practice. Based on the counts of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) from a model. There are a few different metrics that can be derived from these counts to measure and summarize the effectiveness of ML models. Metrics such as accuracy, precision, recall, specificity, F1-score, and receiver operating characteristic (ROC) curve to name a few [3]. The equations for these metrics are as follows.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{FP + TN}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

ROC Curve
X-axis: $1 - Specificity\ (or\ False\ Positvie\ Rate)$
Y-axis: $Recall$

Looking at one metric alone like accuracy would not provide a full picture on the true model effectiveness. A model could be 97% accurate, but if the 3% inaccuracy is mostly from false positives, then the accuracy metric is deceiving. In our case where an analyst would receive alerts based on the classifications, a threshold that allows too many false positives is bad and can even be dangerous for an organization. It can waste time and resources sending security teams chasing non-existent threats. False positives can also lead to alert fatigue, when a security team does not respond adequately to true positives because they assume the alert is another false alarm by the Intrusion Detection System (IDS). This highlights the need to use multiple metrics to properly evaluate a model.

Machine learning is specifically promising for threat detection because of its ability to find the hidden relationships and patterns in data. Where signature-based or rule-based tools rely on what is specifically designated as bad, Machine learning can sometimes detect new techniques or unknown malware known as zero-days. Considering many of the advanced LotL techniques used by Volt Typhoon were previously unknown, a capable ML model could identify and prevent new tactics they might use in the future by recognizing similarities.

1) Ember Model

The model provided along with the Ember dataset is a supervised classification model [15]. Their ML model uses the extracted features from PE files to classify the binaries in one of the two possible classes, benign or malicious [15]. The actual

output of the model is a number between 0 and 1. With 0 being more likely to be binary, and 1 more likely malicious. The numbered outputs can be translated into a classification with the use of a threshold. The threshold is the point applied to an output's probability (number between 0-1) that determines if the binary is classified as malicious or benign. In practice, if the threshold is placed at 0.7 then an output of 0.8 would create an alert for malware.

The model specifically utilizes the Gradient Boosted Decision Tree (GBDT) algorithm from LightGBM [15]. The GBDT algorithm uses what's known as an ensemble method where it operates multiple smaller or weaker models which then combine to make a more accurate and refined model. The smaller models are trained in sequential order, constantly attempting to correct the errors made by their predecessors. The result is a reduced chance of overfitting. Overfitting is where a model essentially learns too much of the training data. Giving too much weight to things like outliers and extra noise that can then result in an inaccurate model.

The actual malicious code in malware can often be very subtle, especially relating to LotL. ML malware detectors tend to use ensemble method algorithms like GBDT because of their ability to notice the subtle indicators in malware that might go unnoticed with the use of other algorithms.

Before the data can be processed by the model, it must go through a process called vectorization. This is a process that converts raw data into numerical data that is then readable by ML algorithms. The Ember model uses a technique called feature hashing for data relating to arbitrary features like strings and imported/exported names [15]. This helps with dimensionality reduction for the features, as it doesn't have to assign new weights to features for these arbitrary data points. Even with the feature hashing, the result of vectorizing the Ember dataset is a 2351-dimensional feature vector [15]. This means the vectorized training predefines 2351 different numerical features. LightGBM, the framework used for modeling performs especially well with a high number of dimensions as seen in this case [15]. With these features represented in machine readable vectorized data, a model can then be trained.

For this project we used the pre-trained Ember benchmark model, so we didn't have to go through the process of training. Instead, we can move onto using the metrics mentioned before to evaluate the model outputs. To clarify, the following metrics are based on the model's ability to classify the testing data included in the ember dataset. The evaluation portion coming up in the results section will cover the model's ability to detect specifically IOCs from Volt Typhoon.

Using the results from a Receiver Operating Characteristics (ROC) curve, a measurement can be taken of the area under the curve (AUC) to represent a model's overall effectiveness [15]. An AUC score of 1.0 would be a perfect model that classified all positive binaries with a higher score than all negative binaries. A model with an AUC of 0.5 would represent a model that performs as well as guessing. While this may seem similar to an accuracy measure, it is not. Accuracy relies on a threshold, and from that threshold shows how often the model makes correct decisions. Whereas AUC calculates the overall ability of the model to separate positives and negatives without relying on a specific threshold. For instance, a model could have an AUC of 0.5 and 90% accuracy at the same time if the data was 90% benign and the model guesses benign every time.

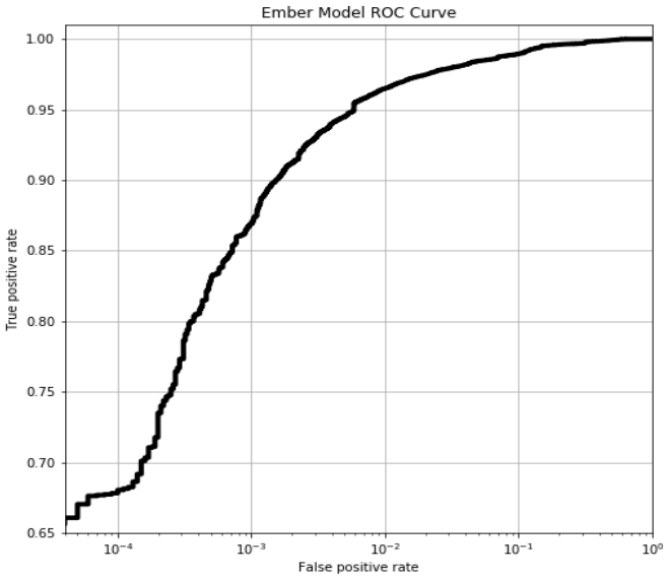

Fig. 6. Ember Model ROC Curve. Source: [15]

Figure 6 is the ROC curve from the Ember model. The AUC score for the curve was 0.99642, which shows that the model is very effective at discerning between benign and malicious binaries. There are two thresholds that are measured chosen based on achieving a certain false-positive rate

Threshold Comparison		
Threshold	0.8336	0.9996
False Positive Rate(FPR)	1%	0.098%
False Negative Rate(FNR)	3.502%	13.192%
Accuracy	0.9775	0.9336
Precision	0.9897	0.9990
Recall	0.9650	0.8834
F-1 Score	0.9772	0.9377

Fig. 7. Threshold Comparison Table.

These same measurements could be taken at any of the thresholds between 0 and 1 to fine tune the alert system. The main idea is figuratively deciding how large of a net you want to throw. The lower threshold will catch roughly 10% more malicious binaries, but it should be considered what a 1% false-positive rate looks like in a real environment. For the other option, the chances of alert fatigue are slim with a low 0.98% false-positive rate, but the model would miss a lot of true negatives with a 13.192% false-negative rate.

2) Adobe Model

The model from Adobe is another supervised classification model [5]. Their model's purpose is to classify command lines as either LotL, or non-LotL. They tested three different algorithms for the model. The three algorithms were Random Forests, SVM, and Logistic Regression [5]. They found the best results based on a few metrics and model training time with the Randon Forests (RF) algorithm, so that is the one they used for the pre-trained model they provide [5].

Like the GBDT algorithm, the RF algorithm also uses an ensemble method. With many of the same benefits like the model being less prone to overfitting by combining multiple decision-trees to create an output. Instead of sequentially improving from one tree to the next, RF provides each tree with a subset of data that it bases its individual output on. The final classification comes from a majority vote by the decision trees. They specify the model to use 50 separate individual decision trees that make up the model [5].

Instead of preparing data for modeling with vectorization, this model uses a different method of feature extraction. They choose to use an approach they describe to emulate how a human analyst would approach recognizing a malicious command [5]. They essentially break down the commands into different identifiable parameters such as known binaries, IP addresses, file paths, port numbers, etc. They then assign tags like "private_ip" or "python_script" to the identified parameters in the command line. Then with these tags applied, the model can be trained to evaluate with the context of the tags in a command to determine if it is being used for LotL or not [5]. An example of what this process looks like can be seen below in figure 8.

Command: *export RHOST="127.0.0.1";export RPORT=12345;python -c 'import sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("11.12.133.14"),int(os.getenv("RPORT"))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn("/bin/sh")'*
Extracted features: **IP_LOOPBACK IP_PUBLIC PATH_/BIN/SH COMMAND_EXPORT COMMAND_PYTHON COMMAND_FOR KEYWORD_-C KEYWORD_SOCKET KEYWORD_OS KEYWORD_PTY KEYWORD_PTY.SPAWN python_spawn python_socket python_shell import_pty**
Prediction: **BAD**

Fig. 8. Example of tags for a reverse shell. Source: [5]

They don't provide the ability to extract measurements from the test data, so to understand the model's effectiveness we have to rely on the metrics they provide. They used 5-fold cross validation, which splits the data randomly into 5 equally sized sections. Then the model iteratively moves through the separate folds to train and test the model. This resulted in a f-1 score of 0.95 for their RF classifier. A score that reflects a low false-positive rate and a low false-negative rate. The other metric provided was a standard deviation of 0.013. This shows model consistency throughout the different folds of data. Overall, from their test results provided this looks to be a robust model that should be effective at discerning LotL attacks from normal usage.

IV. Test and Results

As covered in the previous sections both models use similar algorithms but are aimed at detecting different things using different methods. This section will cover both models and the validity of their approaches in identifying Volt typhoon tactics by classification.

A. Ember Model

Choosing the method to use for determining the ember model's ability to classify LotL tactics provides a challenge. Testing the model with normal system binaries would be futile because PE files contain information only pertaining to the make-up of binaries and not the way they are used. This means the main binaries used for LotL tactics (CMD.exe, WMIC.exe, PowerShell.exe, etc…) wouldn't contain anything malicious within the information contained in PE files to detect. This is why for our testing of the ember model; we will adopt the more relaxed definition of LotL presented by Barr-Smith et al. [13]. That describes LotL as any tactics that are often used for normal purposes, including foreign binaries if they also have a non-malicious practical use. We also tested binaries that don't fit into this definition but were still leveraged by the group.

The test included four different binaries. Two included in the malware analysis report [8] from CISA et al. A fast reverse proxy (FRP) client agent and a credential extraction tool called Mimikatz. Then two control test executables made using the tool msfvenom with the respective purposes of performing recon and creating an obfuscated TCP reverse shell. The Fast Reverse Proxy Client (FRPC) is the original version of the binaries SMSvcService.exe and Brightmetricagent.exe. The difference between Volt Typhoon's binaries to the original FRPC includes obfuscation and a few small modifications [6]. Both variants of FRPC used by Volt Typhoon were not available anywhere.

File Name	Malicious Score	LOTL	Description
FRPC.exe	0.0181	Yes	Fast Reverse Proxy Client Agent
Mimikatz.exe	0.9906	No	Credential Extraction Tool
Recon.exe	0.9914	Yes	EXE that runs CMD.exe Recon
EncodedShell.exe	0.9999	No	Obfuscated TCP Reverse Shell

Fig. 9. Ember Score for Tested Binaries.

The scores predicted by the model show both control binaries are classified as malicious (Recon.exe and EncodedShell.exe). With the higher threshold both binaries used by Volt Typhoon would go unnoticed. However, with the lower threshold Mimikatz.exe would be classified as malicious with the FRPC still being classified as benign.

The FRPC agent is part of a tool commonly used for creating proxies within a network by administrators. This tool along with many others used for LotL is dual purpose. For this reason, it is disappointing but also somewhat expected that it was classified as benign. This result points back to the reoccurring theme that LotL is by design extremely hard to

identify. Unfortunately, this also shows that while the Ember model is very effective for classification of malicious binaries in the form of malware, static analysis of PE files is not a great method for classifying binaries used for LotL.

B. Adobe Model

The Adobe model, designed specifically for the purpose of detecting LotL tactics in the form of command-lines, should prove to be a more effective method. We tested the model with command lines specifically used by Volt Typhoon and listed in both reports from CISA et al. [6][7]. This list originally included 84 commands but ended up totaling 63 testable commands due to certain commands not being recognized as valid. Most unrecognized commands can be attributed to unsupported syntax. The test data included commands used for many different purposes in their attacks. Cases like enumeration, credential exfiltration, proxy creation, remote code execution, and much more.

The command outputs from the model all fall within three categories. Either good, neutral, or bad. The model is still a binary classifier, but the way it works is if a command line doesn't have enough recognizable variables, it is listed as neutral. This is the model's way of saying it doesn't know rather than placing an incorrect classification. It is fair to assume that they include neutral and good as not-Lot, and bad as LotL. This would be in line with the general conservative approach first shown in the distribution of its training data, which was only 0.02% malicious in order to avoid false positives [5].

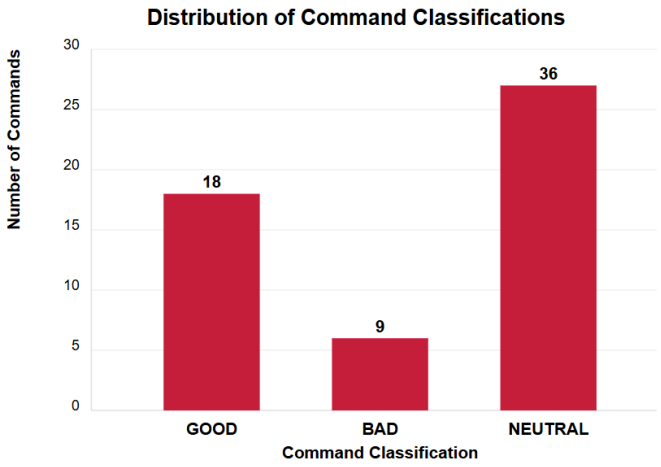

Fig. 10. Distribution of Adobe Command Classifications.

In Figure 8 the distribution shows a very large portion of the commands being labelled good and neutral. Most of the commands that received this classification were those used for recon with common tools, file compression, and other tasks that are similarly often used by administrators. Many of them would be suspicious to a trained eye, but for a model that is trained on only 0.02% malicious data is not a surprising result. For the nine LotL (bad) classifications the model performed well with commands that do the following:

- Combine wmic and ntdsutil
- Directory creation
- Commands using mimikatz.exe
- Shadow copy manipulation
- Unauthorized file transfers

These results show that the Adobe classifier is very effective at identifying and classifying tactics related to credential extraction and other late-stage operations. It also points out weaknesses, with how the model is not as good at detecting early actions such as enumerating the environment. Like the Ember model this points out the difficulty with LotL and why it is such an effective tactic. It is a balancing act between trying to catch every malicious action while also trying not to wear out the alarm and the analysts that have to triage the alerts.

V. Related Works

There are a few different papers and resources found in preparing for this project that took on a similar challenge to ours, trying to secure systems from sophisticated actors. Two of these works used or analyzed ML in novel ways to try and detect intrusions [9][3]. Then there was another resource that offered insight and a detailed look at the ember dataset [1].

Ongun et al. provide a similar approach to what we saw with the adobe model. They use data gathered from command lines to determine if a user is acting maliciously. Using text embedding methods to contextually represent command-line tokens [9]. They also introduce a way to continue model training with what they call "active learning". They do this by selecting samples based on anomaly scores and model uncertainties to be labelled by an analyst and then used to further train and update the model. This is a dynamic approach and is similar to anomaly-detection models that create a baseline of normal system behavior and alerts on deviations from that baseline. This process of iteration would be a useful feature in the adobe model to address the neutral outputs. For a common point of reference, they compare their model to the performance of different anti-virus (AV) solutions and find that their model performs considerably better. [9]

Umer et al. present a survey of methods to use ML for intrusion detection in ICS [3]. They don't provide a solution in the form of a model but instead dive into which methods are viable and why. Their paper points out the potential in less common ML types like semi-supervised and reinforcement learning. They do so in a way that not only provide valuable understanding of how to protect ICS but also explain different ML concepts in an effective manner. They also highlight the possible inaccuracy and misrepresentations in using only one metric to measure the effectiveness of models. Their points on how to measure model effectiveness contributed to many of the metrics used for testing the models in this project. Their focus on ICS provides value to understanding Volt Typhoon's efforts to compromise OT systems. The use of machine learning to detect intrusions in ICS is another important measure to consider when protect against sophisticated threats like Volt Typhoon. [3]

Barr-Smith et al. covers the techniques of LotL in detail providing valuable research highlighting its growing usage, effectiveness for evasion, and challenges found in its detection. The paper uses a dataset including over 31 million malware samples. A portion of which are benign examples from the ember dataset [15]. The combined data shows LotL tactics made up 26.26% of all malwares used by sophisticated APTs, compared to just 9.41% in commodity malware. [13] Along with the data related findings, the paper tested the ability of popular anti-virus software to detect LotL. They found that even after discovering and disclosing limitations to providers, the solutions put in place were not comprehensive. Many anti-viruses implemented band-aid solutions that block the specific payloads that were disclosed but without blocking the mechanisms, small adjustments make the tactics just as effective as they were before. These findings contribute to the need for a different approach. Pointing out that while anti-virus is very effective at detecting malware, they struggle with LotL.

VI. Conclusion

This project has examined the threats posed by Volt Typhoon. Analyzing the groups actions to preposition themselves in the IT environments of United States critical infrastructure. An understanding of the true threat posed by Volt Typhoon is gained by looking at the potential outcome if the group gains footholds in their victims OT environments. If they are successful in compromising OT and choose to act destructively, historical evidence of previous sophisticated attacks on critical infrastructure points to severe impacts on the nation's vital functions.

Volt Typhoon's use of LotL tactics provides a great challenge for detecting. By limiting the use of malware and relying on native tools, Volt Typhoon has been able to persist in systems for prolonged periods of time. In certain cases, they have remained in systems for over five years [7]. Their tactics find such success in part due to the limitations of signature-based detection methods. Highlighting the need for multi-layered defense strategies that implement many different methods for detection and threat hunting. In this project, we evaluated two open-source projects that use machine learning for threat detection. With a goal of finding an effective method to detect Volt Typhoon's known tactics, and potentially unseen tactics as well.

The Ember model is very effective at detecting malware. However, similar to the limitations of signature-based tools, the Ember model is not well equipped for detecting LotL techniques. This is due to the model's use of static analysis. With static analysis, the model doesn't look at the way a binary is being used, but only the blueprint of what the binary does. Since LotL relies on the use of benign binaries for malicious purposes, the Ember model is not a great tool for detecting LotL techniques. This is not to say that the Ember model was a "failure" in this project, because it does well at properly classifying the malicious binaries used by Volt Typhoon. Meaning the Ember model is an effective way of identifying the group's tactics, just not specifically their LotL tactics.

The Adobe model presented a more promising result than that of the Ember model. In hindsight, the model's success compared to the Ember model makes sense with it being designed specifically for detecting LotL. The model's ability to evaluate command lines considering the context of the way benign tools were being used made it effective. The model did only classify a small portion of the total number of commands run by Volt Typhoon as LotL. However, considering the models careful tuning to avoid a high false-positive rate, the number of detections and the usage of the commands classified as LotL was a successful result. It consistently classifies the more common everyday commands as benign, and more importantly classified less common malicious commands as LotL. Based on the results, a live implementation of the model should have a good balance between detecting LotL, without causing alert fatigue.

This project has shown that ML is a viable method for detecting Volt Typhoon. In general, ML is capable of filling in some of the gaps and limitations associated with other detection methods. While this is true, the answer is always defense-in-depth. It would be a poor strategy to rely on only one method of detection. An ideal security posture should consist of a combination of many tactics such as ML, anti-virus, yara rules, and more. With implementations such as these, security professionals can protect critical infrastructure and the well-being of people.

The struggle to detect the tactics used by sophisticated actors like Volt Typhoons, and specifically their use of LotL offers many avenues for further research. This project looked at static methods of detecting the group's TTPs, but research focused on dynamic models could be very insightful. For example, a dynamic binary classifier would be able to analyze what a binary does on the system while it is being run. This method is a possible improvement for the Ember model and its limitations with LotL detection. One specific form of dynamic detection that has had success is user entity behavioral analysis (UEBA). UEBA involves gathering a baseline of user and device behavior that trains a model to then recognize anomalous behavior [12]. This is a newer security concept and could be an interesting one to learn about and contribute to going forward.

References

[1] B. Buchanan, *The Hacker and the State: Cyber Attacks and the New Normal of Geopolitics*. Harvard University Press, 2020. pp. 190-207. Accessed via JSTOR, doi: 10.2307/j.ctv3405w2m.J.

[2] T. M. Aljohani, "Cyberattacks on Energy Infrastructures: Modern War Weapons," arXiv preprint arXiv:2208.14225, 2022. Available: https://arxiv.org/pdf/2208.14225.

[3] M. A. Umer *et al.*, "Machine learning for intrusion detection in industrial control systems: Applications, challenges, and recommendations," *International Journal of Critical Infrastructure Protection*, vol. 38, 2022, p. 100516. Available: https://arxiv.org/pdf/2202.11917.

[4] C.-C. Sun *et al.*, "Cyber Security of a Power Grid: State-of-the-Art," *International Journal of Electrical Power & Energy Systems*, vol. 99, 2018, pp. 45-56. Available: https://www.sciencedirect.com/science/article/am/pii/S0142061517328946.

[5] T. Boros *et al.*, “Machine Learning and Feature Engineering for Detecting Living off the Land Attacks,” in *IoTBDS 2022*, 2022. Available: https://www.scitepress.org/Papers/2022/110045/110045.pdf.

[6] Cybersecurity and Infrastructure Security Agency (CISA) *et al.*, “People’s Republic of China State-Sponsored Cyber Actor Living off the Land to Evade Detection,” May 24, 2023. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-144a.

[7] Cybersecurity and Infrastructure Security Agency (CISA) *et al.*, “PRC State-Sponsored Actors Compromise and Maintain Persistent Access to U.S. Critical Infrastructure,” AA24-038A, Feb. 7, 2024. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-038a.

[8] Cybersecurity and Infrastructure Security Agency (CISA), “Malware Analysis Report 10448362.c1.v2,” MAR-10448362.c1.v2, Mar. 17, 2024. Available: https://www.cisa.gov/sites/default/files/2024-05/MAR-10448362.c1.v2.CLEAR_.pdf.

[9] T. Ongun *et al.*, “Living-Off-The-Land Command Detection Using Active Learning,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 442–55, doi: 10.1145/3471621.3471858. Available: https://arxiv.org/pdf/2111.15039

[10] K. Stouffer *et al.*, *Guide to Operational Technology (OT) Security*, NIST SP 800-82 Rev. 3, NIST, Sept. 2023. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf.

[11] The MITRE Corporation, *MITRE ATT&CK Matrix for Enterprise*, n.d. Available: attack.mitre.org/.

[12] Cybersecurity and Infrastructure Security Agency (CISA) *et al.*, “Identifying and Mitigating Living Off the Land Techniques,” Feb. 7, 2024. Available: cisa.gov/sites/default/files/2024-02/Joint-Guidance-Identifying-and-Mitigating-LOTL508.pdf

[13] F. Barr-Smith *et al.*, "Survivalism: Systematic analysis of windows malware living-off-the-land," in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9519480.

[14] LOLBAS-Project, *Living Off The Land Binaries and Scripts (LOLBAS) Project*, n.d. Available: https://lolbas-project.github.io/.

[15] H. S. Anderson and P. Roth, “Ember: an open dataset for training static pe malware machine learning models,” arXiv preprint arXiv:1804.04637, 2018. Available: https://arxiv.org/abs/1804.04637.

[16] Mayfly277, Game of Active Directory. Available : http://github.com/Orange-Cyberdefense/GOAD