# Modeling Advanced Persistent Threats to enhance anomaly detection techniques

Cheyenne Atapour*, Ioannis Agrafiotis, and Sadie Creese
*Department of Computer Science, University of Oxford, UK*
firstname.lastname@cs.ox.ac.uk

**Abstract**

Advanced Persistent Threats (APTs) are characterized by their complexity and ability to stay relatively dormant and undetected on a computer system before launching a devastating attack. Numerous unsuccessful attempts have utilized machine learning techniques and rule-based technologies to try and detect these sophisticated attacks. In this paper, we opt for a more theoretical approach to identify unique APT characteristics, distinguishable from other multi-stage attacks. We model four well-known APTs, based on the kill chain framework, and we identify common behavior to create abstract models which describe generalized APT behavior. We find that attributes from the Command and Control phase of these attacks provide unique features that can be used by any anomaly detection systems. We further validate how expressive our abstract models are by formalizing a fifth APT and examining the behavior that was not captured.

**Keywords**: Advanced Persistent Threats, Modeling, Cybersecurity, Anomaly Detection

## 1 Introduction

As computer security is ever-changing, that is, system architectures are constantly updated, and new security bugs are found all the time, fully securing a computer system remains an unsolved problem [1]. One open question within this is whether we can detect all security attacks, or attempts in causing malicious or unwanted behavior. The potential for such an attack may rise from insiders, who are usually entities with privileged access to a particular computer system, or outsiders, who may need to perform extra steps in order to gain such access, and must launch their attack from somewhere outside the immediate network of the computer system, for example, from the public internet. These are usually referred to as internal and external threats, respectively. This paper focuses on a type of threat which intersects these two realms, namely, Advanced Persistent Threats (APTs).

An APT is a particular type of malware that is characterized by its complexity and ability to stay relatively dormant and undetected on a computer system before launching a devastating attack [2]. Malicious actors usually perform their actions in multiple stages [3], starting by gaining access to a system, before laterally moving to other parts of the network. A Command & Control (C&C) center may provide additional instructions on how the attack should be executed before trying to conceal the malicious activities. APTs, due to their sophistication, share common characteristics with other type of malware or legitimate software, hence the difficulty of detecting them with existing anomaly detection systems.

APTs can have serious impact on organizations and nations. A system can be infected, and the APT can stay dormant for potentially years before inflicting any damage. A recent example is that of Russian state actors allegedly deploying APT-type malware on network devices, such as routers, firewalls, and network intrusion detection systems (NIDS), mostly targeting governments, private-sector organizations, critical infrastructure providers and the ISPs that serve them [4].

*Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 9:4 (December 2018), pp. 71-102
*Corresponding author: Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, OX1 3QD, UK

Academic work on modeling APTs and designing systems to detect such behavior is scarce. To fill this gap, we conduct a systematic literature review of relevant work that has been done in this field, and identify lessons learnt in APT detection. We then use this information to create models which capture the behaviors of APTs. We then model five well-known APTs, based on the kill chain framework, and we identify common behavior. We use information from the literature review and the analysis of the case studies to create abstract models which describe generalized APT behavior. Based on these models, we can determine features that will be useful for training machine learning algorithms and/or developing rules to detect APTs.

## 2 Literature Review

There are two general approaches for detecting attacks on computer networks. One is signature or rule-based detection, which uses a crafted pattern or chosen attribute(s) that should only match with malicious traffic. The other involves machine learning algorithms in order to make probabilistic inferences about whether particular network traffic seems suspicious and potentially malicious. This is achieved by analyzing carefully selected features of such traffic [5, 6, 7, 8]. One problem with rule-based detection is that the rules are usually crafted to detect attacks that are already known, and so such an approach struggles with detecting unknown or zero-day attacks. In addition, rules can be rigid, and in some cases the malware can be altered slightly to evade detection by the rule. This may result in the need for another rule to detect the new iteration of the malware (i.e., the case of the Snort community rule for the Satan Scan having to be updated to detect the newer Saint Scan [9, 10, 11]). The machine learning approaches can overcome these issues, but one of the main shortcomings is that the data used, besides the fact that it is usually hard to come by and non-standardized, may reflect only on a particular system. Then the results achieved in a particular body of work using such an approach (e.g. [5]) may be skewed to work well with the data set used and thus not indicate that the system developed is generalizable to all relevant settings.

Commenting on specific approaches, Zhang et al. present a method to determine both malware distribution networks (MDNs) and drive-by download attacks [12]. Their approach is based on the rule-based detection methods where regular expression based signatures using the URLs of the central servers of an MDN are generated. Yen et al. demonstrate their methodologies for tackling the problem of big data in attack detection. Their project involved standardizing a large-scale dataset obtained in the form of logs from various security devices in an enterprise network. They used an adaptation of an unsupervised k-means clustering machine learning algorithm to detect suspicious host behavior within this network data [5]. Their system was able to detect threats that the security analysts at the enterprise were unaware of, even with the use of state of the art tools. In a similar vein, Stringhini et al. explore the idea of using a content-agnostic approach to detecting malware by using download graphs and semi-supervised label propagation with Bayesian confidence to propagate the reputation of a known malicious file to an unknown file without explicitly examining it [6]. They also perform an empirical analysis on their data to determine the best parameters to use in their system.

Kwon et al. cleverly make use of features associated with download graphs to train a random forest classifier to detect a particular type of malware called “downloader trojans” or “droppers”, which download other malware from the internet [7]. This is another example of a very successful approach to attack detection using machine learning, and how the algorithm was chosen carefully in regards to the nature of this kind of data. Invernizzi et al. also use a content-agnostic approach to detect the distribution of malware from malicious sites based on how certain network traffic is produced in a large network of users [8].

Rajab et al. developed an in-browser agnostic malware protection system called CAMP [13]. The idea behind their system was to bridge the gap between whitelist and blacklist approaches to malware detection. The problem with a blacklist approach is that malware authors can slightly change their code or distribution domains in order to avoid the blacklist. A whitelist approach has the problem of maintaining a large and continuously growing pool of non-malicious files. CAMP uses a binary classifier to establish ground truth for malware binaries, but allows for any binary classifier to be used. It considers features given by the browser at the time of download such as URL and IP address of download site as well as size of download, content hash, and download signature.

Giura et al. have re-modeled the problem of APT detection and proposed a methodology to detect attacks on an organization network [14]. They provide the conceptual model of an attack pyramid, which describes the stages of an APT. They explain that while APTs are generally unique and customized for a specific target, they commonly share particular stages in their lifespan. They also claim that while APTs may be extremely difficult to detect using traditional security tools, in most cases, they can be detected during one or more stages of their operation. They support this claim by showing preliminary experimental results of using their method for APT detection. Virvilus et al. studied four popular examples of APTs that have surfaced in the recent past, giving a technical analysis of each, and describing similarities [15]. They suggest reasons as to why current malware detection solutions have failed to detect these examples. This paper is significant because it looks at real-world examples of devastating APTs that were proven to be difficult to detect at the time the attacks were launched.

Vries et al. have made an in-depth roadmap for the development of a system to detect APTs [16]. They describe possible solutions to this problem and assess respective possible approaches. For example, they mention unsupervised clustering algorithms such as k-means clustering and self-organising maps. However, the problem with these approaches is that they can generate many false positive alerts. They also mention the possibility of applying semi-supervised approaches, which operate on a data set that is partially labeled. This is advantageous because it reduces some of the false positives generated by unsupervised algorithms and does not require as much manual processing of data as a fully supervised approach. Furthermore, clustering may be used to label previously unlabeled data, and the resulting data may be used to create rules or signatures in a proposed method called fuzzy rule-based anomaly detection. Boosting, or using multiple algorithms to confirm that behavior is anomalous in a voting approach, is also suggested to reduce false positives.

Vries et al. walk through the eight relevant steps of an APT attack and describe when certain network events can be detected and when they may not. For example, they describe a second step in which carefully crafted phishing emails are sent to employees of a targeted organization which contain malicious links. The presence of a hyperlink in the email, the link address, the source, the target, the IP address of the workstation which accessed the email, and timestamps of relevant events can all be used as features to detect this suspicious behavior. While this particular behavior may not be unique to APTs, similarly future observed network events, as well as other future correlated events may increase confidence of APT presence.

Yang et al. use information fusion for situational awareness and threat projection from large data [17]. They analyze two different systems they created in regards to how well they can detect multi-stage cyber attacks using criteria developed by the US Air Force Research Labratory (AFRL). The two systems that Yang et al. created both rely on a generalized model of all potential attack tracks which is independent of any network structure of a system. The two systems also rely on another model which captures when certain network entities and information may be exposed to an attacker, such as hosts, servers, access privileges and sensitive files. The two proposed systems attempt to correlate observed network behavior with the two models in order to effectively detect malicious activity.

We have presented research which covers potential strategies for APT detection, including features that security experts have deemed effective for this purpose. However, work on modeling well-known

| Authors | Reference | Detection Method |
|---|---|---|
| Zhang et al. | [39] | Rule/Signature-based |
| Yen et al. | [38] | ML: Clustering |
| Stringhini et al. | [34] | ML: Semi-supervised learning |
| Kwon et al. | [28] | ML: Random forest classifier |
| Invernizzi et al. | [27] | ML: Content agnostic |
| Rajab et al. | [32] | ML: Binary classifier |
| Giura et al. | [25] | Rule/Signature-based |
| Virvilus et al. | [35] | N/A (Suggested Defenses) |
| Vries et al. | [22] | N/A (Suggested Methods) |
| Yang et al. | [36] | ML: Information fusion |

Table 1: List of papers presenting a signature based approach or a machine learning approach

cases of APTs has not received much attention by the academic community. This paper will attempt to further this type of work, and specifically pinpoint the differences between APTs and other types of multi-staged attacks.

# 3 Methodology

The main objective of this paper is to determine that an APT has infected a machine, and in particular, to distinguish APT behavior from other less sophisticated attacks. To accomplish this, we decided to initially study four different APTs, one botnet, and one ransomware. The initial four APTs studied were Stuxnet, Flame, Red October, and Duqu. We then studied a fifth APT, Gauss, for validation of our created models to see how well they were able to capture generalized behaviors of APTs. The botnet studied was Mirai, and the ransomware studied was WannaCry. These were chosen because they were of great significance to the security community, and because there were many papers that covered these malwares in great detail. We knew, in particular, that APTs differed from ransomwares and botnets in their Command and Control phase of the kill chain [18], so we emphasized on this area.

Based on the case studies, we created models to describe the behavior of each malware. We made three different models for each of the malware studied: a verbose model of the kill chain, an LTS diagram showing the overall movement of the malware and triggering events, and an MSQ diagram showing the command and control activity of the malware. An LTS diagram (Labeled transition diagram) is a diagram which contains states and arrows. Such a model may be used to describe how software operates [19]. A software may be performing a particular task at a certain period of time, which can be modeled by the software being in a particular state at that moment. Then, the software may perform a triggering activity which leads to a change in behavior. This triggering activity may then be modeled using a transition from one state to another with a corresponding label in an LTS diagram. An MSQ diagram (Message Sequence Diagram) is a diagram which captures the sequential communication between two entities [20]. We make use of such a diagram to show the sequential communications between malware running on a victim machine and an attacker controlled command and control server.

We decided to use these three models because we felt they each convey the relevant information from sufficiently different perspectives as to assist in the process of finding similarities and differences between the studied threats. The verbose model helps to categorize most of the relevant information of the particular threat into the stages of the kill chain. For this diagram, there is a focus on events which occur *within* each stage of the kill chain. This diagram also serves as a reference for the other two models,

particularly the LTS diagram. Once a structural similarity was found between another diagram type, we could easily refer back to the verbose diagram to determine the exact activities which were responsible for the structural similarities. The LTS diagram strips away the wordiness of the verbose model, and helps to see the movement of the threat between the different stages of the kill chain in a simplified manner. From this diagram, it is made clear which activities cause the threat to move from one stage to another. Finally, the MSQ diagram highlights only the activity performed by the threat in the Command and Control stage of the kill chain. For our purposes, this was an important diagram because the only data to be considered was network data, and such data is mostly relevant for the Command and Control stage of the kill chain. This diagram helped us visualize, sequentially, which activities in this stage might correspond to malicious behavior, and from what source machine we could expect those activities to originate.

We then made three abstract models, one for each model type, which describe the generalized behavior of the APTs studied. We then studied an additional APT to see if our generalized models would also capture its behavior. Using these abstract models, we were able to extrapolate features for use in a detection system.

# 4 Case study analysis of APTs

## 4.1 Stuxnet

Stuxnet was an APT that affected the industrial control systems (ICS) in uranium enrichment plants and ultimately caused the centrifuges running in these plants to operate at unsafe speeds and destroy themselves. Stuxnet is speculated to have been created by the U.S and Israeli governments with the intent to sabotage the Iranian Nuclear Program, and specifically to target the uranium enrichment plant in Natanz. This is believed to be the case because of Stuxnet's objective, its complexity, and because its payload was only activated in controllers at Natanz uranium enrichment plant [15]. The earliest observed sample of Stuxnet dates back to June 2009. Stuxnet was significant in that it was incredibly complex, relying on four zero-day vulnerabilities, forging digital signatures, and an intimate knowledge of the environment of the infrastructure it would operate in, and also because Stuxnet proved that attacking infrastructure with software is possible [21].

It is unknown exactly how the reconnaissance was done in order to successfully develop and execute Stuxnet, however, it is speculated that there may have been a physical component to the reconnaissance because the malware requires very in-depth knowledge of the uranium enrichment plant to operate. This could have potentially come in the form of an insider, someone who works at the plant, revealing necessary information to the appropriate party. It is known that Natanz's purpose and location was leaked by a dissident group in 2006 [22], so perhaps the creators of Stuxnet took advantage of this information in the malware's development.

The initial infection of Stuxnet still remains unknown, however it is believed that the delivery method was through the insertion of a removable drive loaded with the malware into a machine. This is a likely infection method since Stuxnet is indeed capable of infecting removable drives and did so in order to spread itself [15]. This activity could have been performed by an insider who was perhaps paid or otherwise persuaded to do so. Alternatively, it is possible that an attacker took advantage of natural human curiosity and dropped many USB devices containing Stuxnet in a popular location, such as a parking lot near a target organization [23, 24].

The installation process of Stuxnet is fairly lengthy, technical, and not all is relevant to, or in scope of, this paper. Thus, we describe only the parts we considered to be able to derive features from. The main component of Stuxnet is a .dll file that contains all of the exported functions (we will also refer to these as "exports" in this paper as the literature does) and resources that Stuxnet uses in order to execute its

tasks (Stuxnet's goal remains the same, but it behaves differently depending on the environment it finds itself in (i.e. victim system architecture, installed security products, etc). Along with this .dll file are two encrypted configuration blocks. The configuration blocks are used to keep track of the information Stuxnet gains about the victim machine it is executing on, and then Stuxnet can use this information to operate accordingly. These three items are all bundled together in a section called "stub" in a dropper component which is a wrapper program. This wrapper program is responsible for placing Stuxnet's main .dll file in memory and calling exports. It also passes a pointer to its original stub section as a parameter to the export, and each subsequent export does the same, so that every export will be able to access all the necessary parts of the malware. Stuxnet typically operates by injecting the entirety of its main .dll file into one of a set of trusted processes that could potentially be running on the victim machine.

The first export that is launched once Stuxnet's main .dll file is loaded for the first time performs checks to see if the victim machine is running a compatible version of Windows, is infected or not, if the current process has system privileges, what security products are installed on the victim machine, and what running process is best to inject into. If this export determines that Stuxnet is not running on a compatible version of Windows, the threat will exit, and nothing further will happen. If it determines that it does not have system privileges, it will exploit one of the two zero-day vulnerabilities to elevate its privileges, otherwise it continues. This export will then inject the .dll file into the process and call another export, which is the main export used for installation.

This new running export will now check that the configuration data Stuxnet had collected earlier is valid. Once it has determined this, it will check a specific value in a specific registry key, and will terminate all execution if it is precisely 19790509. This number has some significance because it corresponds to a date upon which the first Jewish person was executed by the new Islamic government. This fact could potentially be evidence that Stuxnet was indeed a political move by a nation-state, or perhaps a different party was trying to frame another. Regardless, we take note of this behavior of Stuxnet making these specific checks on its victim system and revisit this idea later.

Now Stuxnet will check if the current date is later than June 24, 2012, and will stop executing if it is. Symantec's work on Stuxnet [25] does not make any speculations about how this date was chosen or why. Perhaps the attacker decided that in the amount of time that Stuxnet was planned to run, sufficient damage would have been done to the Natanz facility, and hopefully Stuxnet could disappear fast enough that nobody would suspect a malware was responsible for the destruction of the centrifuges.

Then, Stuxnet will reduce the integrity levels of Windows objects [26] on the victim machine so it can then try to create a global mutex for use in communicating with different components and not worry about denied writes. Windows uses a system to restrict access permissions of applications which run on different user accounts. This system is called the Windows Integrity System. Essentially, by reducing the integrity levels of Windows objects, Stuxnet will be able to modify more files and application processes on the victim machine, effectively elevating its privileges.

Stuxnet will then create three encrypted files from the data it has in its stub section, and write these to disk. It will then check again that the current date is before June 24, 2012. It then ensures that its current version is the same by decrypting a file it just wrote to disk, and continues if so. It will decode and write two files from its resources to disk. Stuxnet will change the file creation time on these two files to match the other files in the system directory to avoid suspicion. Then Stuxnet will create entries in the victim system's registry to make sure these two files start automatically upon booting the Windows OS on the machine. Finally, two other exports are initiated. One will infect new removable drives and starts the RPC server for command and control activity. The other will infect Step7 project files on the victim machine. After a brief pause, Stuxnet will begin attempting command and control activity.

Stuxnet will now test its internet connection by querying one of two legitimate microsoft URLs, www.windowsupdate.com and www.msn.com, and if successful, will try to receive some information from the command and control center and store this in a log file. It will then send some basic information

it previously collected about the victim machine to the command and control server through HTTP on port 80 and XOR encrypted. Stuxnet has the ability to reconfigure itself to communicate with new command and control servers, but there is no evidence that this functionality was ever used.

The command and control server may respond with pure binary data, also XOR encrypted, which contains a payload module. Depending on the response, the victim client would either inject the payload into the current process or another process and run it. Symantec did not find any modules sent by the attackers in this manner at the time of the writing of their report, but state that this functionality could have been used by the malware to perform additional tasks or to update itself.

The Actions on Objectives phase of the kill chain for Stuxnet involved a WinCC SIMATIC server connecting to specified Siemens PLCs and altered the system to make the centrifuges operate outside safe conditions in order to destroy themselves. Meanwhile, the malware also forced the system not to inform the workers at the plant that it was operating under unsafe conditions so that nobody would suspect anything was wrong and thus the malware could continue for as long as necessary to do damage to the plant.

### 4.1.1 High level kill chain model, LTS and MSQ models for Stuxnet

Based on the description of the malware provided above, we begin by creating a verbose high level kill chain model. The creation of this model assists in visualizing the researched information. In order to properly and effectively correlate the verbose high level kill chain model with the LTS diagram, certain notation is needed to understand the events occurring in each stage. This correlation is necessary because each diagram helps visualize different aspects of each corresponding malware, and combining the diagrams provides a holistic understanding. In order to accomplish this, each stage of the verbose high level kill chain model has near it a small box with a number indicating its corresponding state in the LTS diagram. The black circle with red border acts the “initial state” and indicates the beginning of the malware’s lifespan. The arrows on this verbose diagram are labeled with a minimalistic and formal language which identifies the activity the malware performs that completes the previous state and, by which, the malware proceeds to the next stage.

In general, the LTS models were built from the verbose models since these models were generalizations of the verbose models. The rectangles representing the different steps of the kill chain were used as a template to assist in creating the diagrams, as well as to help regularize the diagrams so that differences and similarities in the malware would hopefully be easier to recognize. The same boxes which labeled the states in the previous verbose diagram are now used here as the states. The black circle with red border again acts as the “initial state” and its purpose is to identify where a reader should begin following the diagram. The legend below the diagram indicates the stages of the kill chain to which the numbers labeling the states refer. Extra information, if any, is also displayed near this area. In the case of the Stuxnet LTS model presented below, it is explained that yellow arrows indicate behavior which is repeated by the malware until a result is achieved. Furthermore, the double-sided blue arrow is the same as in the previous model.

The MSQ diagrams for Command and Control (C&C) communication in general consist of a victim client and the C&C server, along with any other intermediate agents. These are identified by the stick-figures on the left and right sides of the diagram, respectively. Some relevant information may be included which is not communicated between these two agents that we decided may be helpful for potential detection strategies. Such activities are indicated by vertically downwards arrows on the corresponding side of the agent performing the activity. Mostly this kind of activity is relevant if it is done by the victim client, but sometimes activity is provided for the C&C server for completeness.

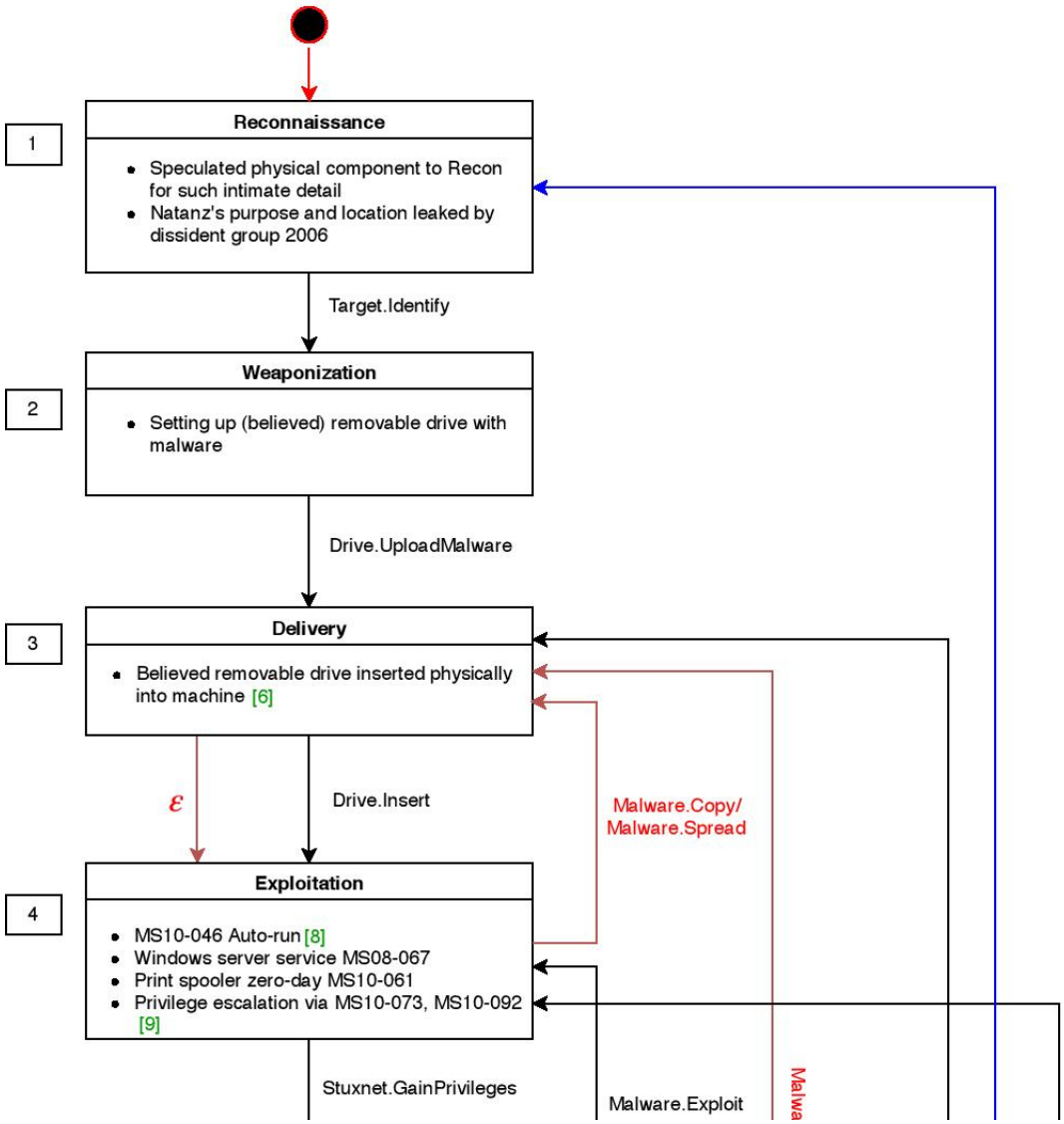



Figure 1: Stuxnet Verbose Diagram Part 1 detailing steps 1, 2, 3 and 4 of the Kill Chain

## 4.2 Flame

Flame was first detected in May 2012, but it is believed to have been active between five to eight years before it was discovered. Unlike Stuxnet, Flame's goal was to steal sensitive information, and it mostly affected systems in the Middle East. Its targets were also more widespread than that of Stuxnet's. Flame had the capabilities of key logging, taking screenshots, intercepting email, and recording conversations from a machine's internal microphone. Flame was a significant APT because its size was unusually quite large at almost twenty megabytes and it had the capability to impersonate a Windows Update Server in order to spread itself to new infection targets. Flame is also speculated to have been developed by a nation state because it is estimated that in order to forge the digital signatures necessary for Flame's execution, a highly skilled team of cryptographers would have had to have been paid from two hundred thousand to two million dollars to work together to find appropriate MD5 hash collisions [15]. It should be noted that Flame was a particular hurdle to examine because it is such a large and complex malware

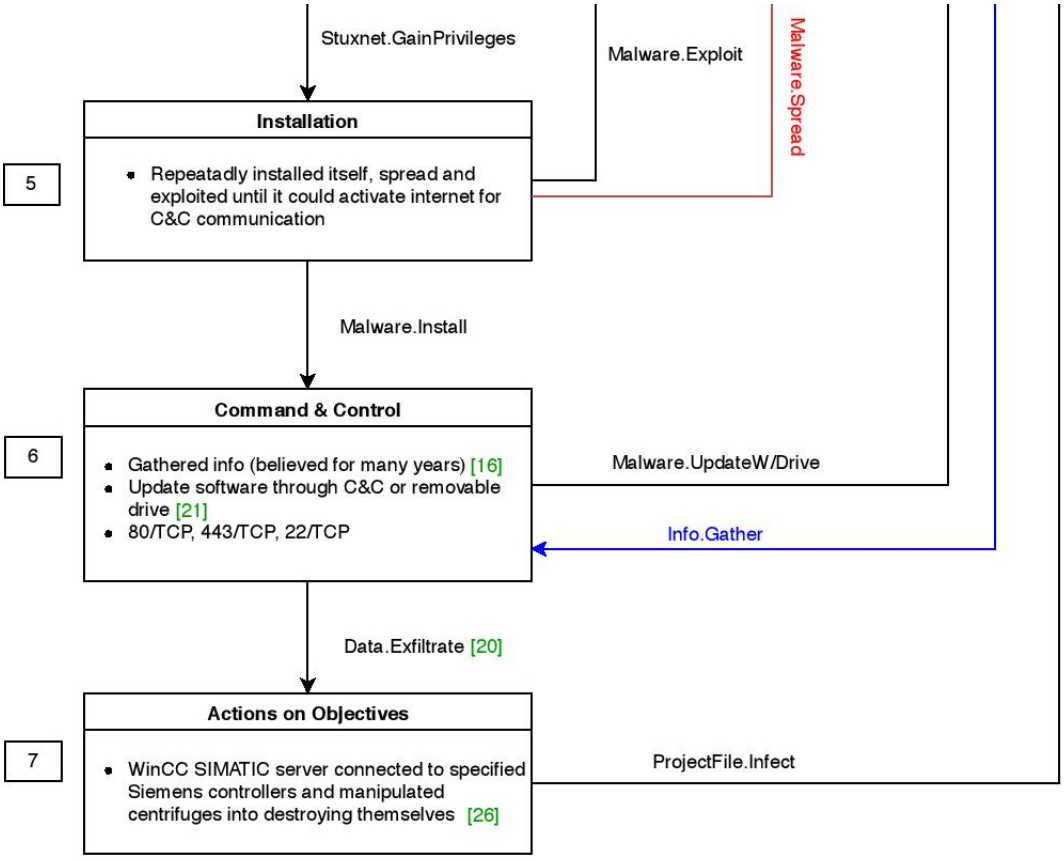



Figure 2: Stuxnet Verbose Diagram Part 1 detailing steps 5,6 and 7 of the Kill Chain

and it has the functionality to determine that it is running on a virtual machine and it will auto sleep, ceasing execution, making it difficult to analyze properly and extensively. We were able to find some relevant sources that cover important aspects of Flame, but it is worth mentioning this fact as an obstacle to potential APT research in the future, as threats may become increasingly more complex and able to avoid not only initial detection, but make thorough analysis a bit more difficult as well.

The initial reconnaissance that was used in order to execute Flame is not known. It would be interesting to learn if Flame's authors performed some reconnaissance activity before its execution, but we speculate that it might be too specific to derive an abstract feature describing the general behavior of APTs from, or it might not be significant enough to make a feature which would be a strong indicator that an APT has infected a network.

It is unknown how Flame initially infected a machine, but it is speculated to have either been delivered to a machine through an infected removable drive or a spear-phishing attack. In fact, at the time of the writing of [27] no dropper component of Flame was made available to the research community, and it is possible that no dropper was identified at all.

Flame took advantage of the same two vulnerabilities that Stuxnet did: the Print Spooler vulnerability (MS10-061), which was used to spread the malware, and the Windows Shell vulnerability (MS10-046), which allowed remote code execution. Flame also took advantage of a forged digital signature obtained through appropriate MD5 hash collisions against Microsoft's Terminal Services Licensing certificate authority in order to disguise a victim machine as a Windows Update server. Another previously uninfected

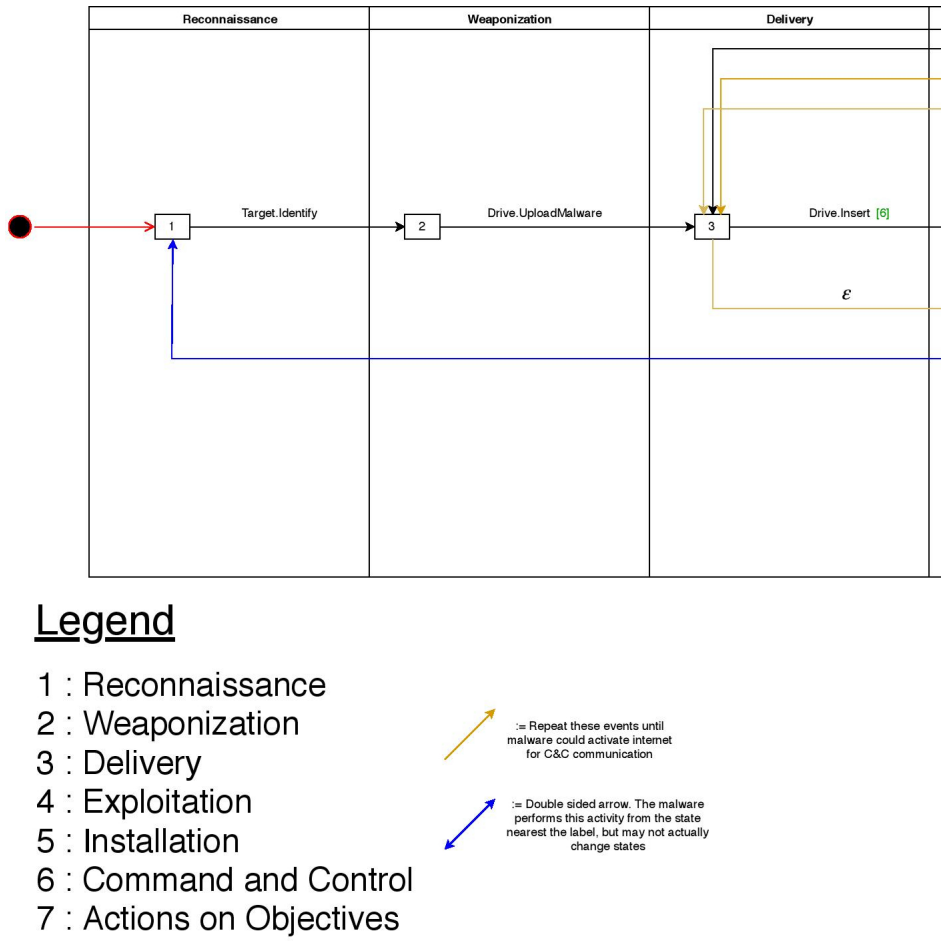



Figure 3: Stuxnet LTS Diagram Part 1

machine on the same network could then attempt to download a Windows update from this infected machine, and in doing so would infect itself and spread the malware.

In order to install itself on the machine, the malware accesses a particular URL to download a file with a .ocx extension and an encrypted header. The Crysys report [28] indicates that the malware sets the user agent to the value "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.2150)" and indicates that this value could be used to create a rule for use in detection, as it is unusual and does not show up in a google search. However, for our purposes, a rule like this is much too specific to this particular APT [28]. Using .ocx extensions for its binaries enabled Flame to bypass many antivirus engines because these types of files were generally not scanned by them in real time during the years Flame was propagating and executing. Flame also had the capability to switch to using .tmp file extensions in the case that McAfee McShield antivirus software was installed on the victim machine [27]. Flame used a stealthy technique to inject its modules into running processes in order to execute them. A memory trace revealed that certain areas of a victim machine's memory had all permission flags set (READ, WRITE, EXECUTE), which is indicative of dynamic memory allocation during Flame's execution [27]. Flame would install many of its modules onto the victim machine and keep them on disk [27]. The threat also alters the victim machine's registry in order to force the machine to run Flame upon system boot.

Flame was observed to communicate with over eighty different command and control server domains. Most of these servers ran Ubuntu and Linux operating systems. Interestingly, they were built to imitate

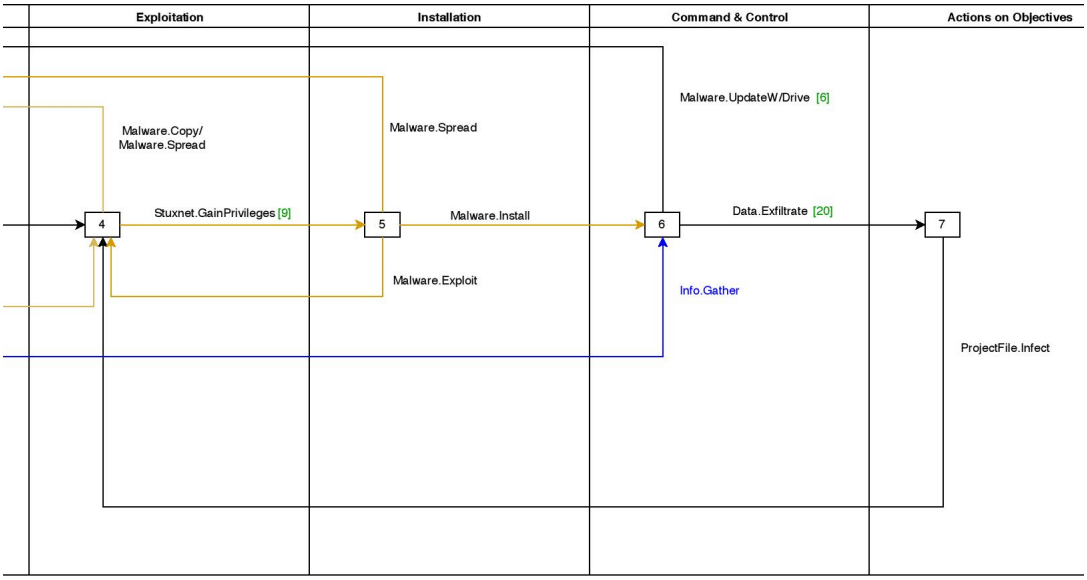



Figure 4: Stuxnet LTS Diagram Part 2

naive botnet command and control servers, but they actually would not accept commands to send to the victim server through direct use of their user interface [27]. The command and control communication between victim machines and servers was through HTTP, HTTPs or SSH on ports 443 and 8080 [27]. Flame used a rootkit in order to hide its network connections. The threat also hid the contents of its communications through XOR encryption, substitution ciphers, and the RC4 stream cipher.

A usual command and control communication session would begin with the victim client sending a request to the server. The server would then log the connection information and protocol version of the threat sending the request. Then the server would first decode the client request and process its metadata, before encrypting it for storage on the local disk. All metadata about files the command and control servers received from victim clients would be saved on a MySQL database with InnoDB tables [29]. An attacker with access to a command and control server would have the option to send a command or a set of commands to a victim. They could do so by uploading a specially crafted tar.gz archive to the command and control server, which would then execute the appropriate commands to the victim client. Note that Flame used mutexes to ensure there was only one instance of the threat running at a time. Infected machines were controlled using a message-exchange mechanism based on files [29]. To avoid detection, the command and control servers performed a well-known fluxing technique commonly used by botnets to frequently change the IP address of the particular host/domain name used in communication.

Flame has different modules which are used by the attacker to perform different operations, including recording microphones, keylogging, taking screenshots and intercepting email. Flame also had capabilities to scan devices in bluetooth range of the infected machine, infect removable drives inserted into the victim machine, create a backdoor on the machines on the network domain, collect network traffic by listening on available network interfaces, and scan for security products installed on the victim machine. There were other modules identified whose purpose was not known at the time of writing the update from Kaspersky Labs on the Crysys report [28]. Interestingly, the Lua scripting language was used to develop some information stealing attacks within the malware, and these were stored in SQLite and unknown CLAN databases. Crysys notes that it is unusual to use databases to store attack related information inside the malware, but does not speculate as to why this was a design decision for Flame. Finally, Flame creates its own whitelist for files to be executed on the victim machine, and places all of


Figure 5: Stuxnet MSQ Diagram Part 1

its own files on this list in order to evade detection. Crysys recommends creating rules for these files to use in a rule-based detection scheme for detecting Flame, but once again, such an approach is too specific to Flame for our purposes. However, it is interesting to note that the naming conventions of files and classes in the scripts used are not common for Unix developers who usually use camelcase, even though most of the command and control servers were running Linux. Instead, the naming convention used was to capitalize all first letters. This kind of out-of-place behavior could potentially be made into an abstract feature to detect malicious activity on a general level.

## 4.3 Red October

Red October was first discovered in October of 2012, but it was believed to have been active since May 2007 [15]. Red October's objective was to steal information and it mostly targeted diplomatic, governmental and scientific institutions located in Eastern Europe, including former USSR members and countries in Central Asia. Based on Kaspersky's analysis of the malware [30], particularly command and control server registration data and artifacts left in malicious executables, they strongly believe the authors of Red October had Russian speaking origins. In contrast to Flame, Red October had a very minimalist architecture, but it had the capability to download many different modules from its command and control servers depending on its needs in achieving goals from a particular victim machine. There

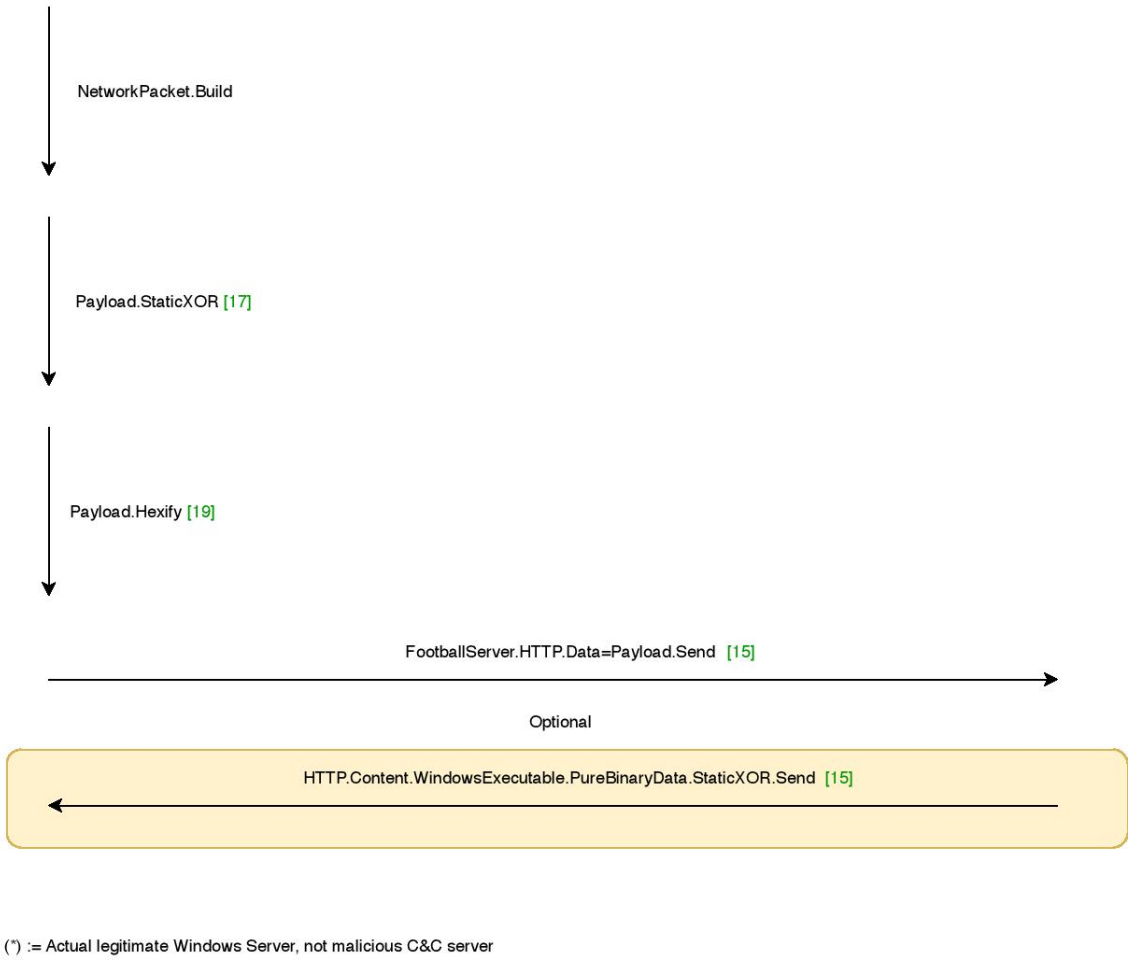



Figure 6: Stuxnet MSQ Diagram Part 2

were over one thousand different modules identified that could have been downloaded and executed by Red October [15]. Red October was able to steal information from nokia phones and iphones, gain access to network devices and even recover deleted files on removable drives. The threat also was able to infect Microsoft Office and Adobe Reader software by installing a malicious plug-in. This plug-in would observe the opening of a specially crafted seemingly non-malicious pdf file on the victim machine. Upon this event, the installed plug-in would parse this pdf file and execute a hidden malicious program from within the pdf file which could give attackers access to the victim machine, even if command and control communication had been interrupted or shut down.

It was public knowledge since the summer of 2011 that organizations of European Union/European Parliament/European Commission used certain cryptographic systems such as “Acid Cryptofiler” to handle their sensitive data. Red October took advantage of this knowledge in order to extract sensitive data from these institutions. We were unable to find any information about further reconnaissance activity performed by the authors of Red October before its execution. Red October’s Actions on Objectives was mainly reconnaissance activity, however, and Kaspersky Lab confirms that, the information gathered by the threat was reused in later attacks [30].

The weaponization method for this threat was to craft malicious Microsoft Word and Microsoft Excel documents that exploited (at the time) known vulnerabilities of these respective software. These

malicious files were attached to emails which were unique and specially crafted for each infection target. The emails used to deliver Red October to victim machines were not found by the researchers at Kaspersky Lab during their investigation [30], but they were able to find top level dropper documents. The researchers speculate that the emails were either sent using an anonymous email service provider available on the public internet, or from victim machines from already infected organizations.

As mentioned above, known Microsoft Word and Microsoft Excel vulnerabilities were exploited by Red October. These vulnerabilities were CVE-2009-3129, CVE-2010-3333 and CVE-2012-0158. Exploiting these vulnerabilities allowed attackers to remotely execute arbitrary code on victim machines. Red October also took advantage of the CVE-2011-3544 Java vulnerability with an exploit called the Rhino exploit. This exploit allows attackers to remotely run arbitrary java code [31], and Red October used this to infiltrate networks [30]. It is important to note that the exploit code used in Red October was exactly the same as the code used in a series of previous targeted attacks with Chinese origins. Researchers at Kaspersky lab speculate that this was purposely done by the authors of the threat to make it harder for other parties to identify them.

Once the malicious document in the email attachment is opened by a user on a victim machine, malicious code is activated through the use of the above exploits which begins the execution of a Trojan dropper. This Trojan dropper will extract and run three files from within the email attachment. One of these files, "LHAFD.GCP", is the main component of the threat. This file acts as a backdoor and is responsible for performing command and control communication. The loader module of Red October will decrypt and uncompress this file, originally encrypted with RC4 and compressed with the "Zlib" library, and will then inject it into system memory.

The loader will then check if an internet connection is established on the victim machine by attempting to connect to three microsoft hosts: update.microsoft.com, www.microsoft.com , and support.microsoft.com. Then, the loader executes the backdoor component that was injected into system memory. The backdoor component will establish a connection to one of the three command and control server domains which are hardcoded in the malware itself, and then begin communications. The backdoor component will identify itself to the command and control server with a special string which includes a hex value, appearing to be an ID unique to the particular victim [30]. The purpose of these communications is to download and execute the appropriate modules on the victim machine to extract the desired information, and then exfiltrate this to the command and control server. XOR encryption was used to encode much of the exfiltrated data, but other methods of encryption and compression were used to exfiltrate data by the different downloaded modules [15, 30]. It is particularly important for us to note that the backdoor module sends an HTTP POST request to the command and control center every one hundred eighty seconds.

Kaspersky Lab identified two different categories of modules: "offline" and "online". Offline modules are stored persistently on the victim machine's local disk. They create their own logs, have the capability to create their own system registry keys, and they may initiate communications with the command and control servers on their own. The online modules are never saved to the victim machine's local disk, and instead exist only in system memory. These modules do not have capabilities to alter victim system registry, and even keep their logs only in system memory, and never on local disk. They also have the capability to send info to the command and control servers on their own.

Many of the domains used as command and control servers were known malicious web servers, mostly located in Germany. Kaspersky identified sixty different domains used as command and control servers. Red October used port forwarding on port 40080 to have some servers act as proxies, hiding the locations of the main servers.

We have already mentioned many of the important Actions on Objectives that Red October performs above. We stress that there are many other modules that the threat can download and use to extrapolate different types of information from a victim machine. Some of these not yet discussed include the

capabilities to access browser history, steal email account information, steal files from a local FTP server, and infect other hosts on the network [32].

### 4.4 Duqu

The fourth and final case study is Duqu. Duqu was first detected in September of 2011, but it is believed to have been active since February 2010 [15]. Even though its objective was to steal information, its similarities to Stuxnet have led many researchers to believe these two malware were developed by the same people, potentially using Duqu to perform reconnaissance for Stuxnet. Duqu was a very targeted APT, affecting no more than fifty total targets worldwide. Most infections were found in Europe and the Middle East [33].

No observable evidence of reconnaissance activity that was performed before Duqu's execution by the authors was found in our research. However, Duqu's goal was reconnaissance, so there may have not been much observable evidence of such activity. The Weaponization phase of the kill chain for Duqu involved attackers crafting a special Microsoft Word document containing a zero-day kernel exploit which, when opened by the victim machine, would allow attackers to install Duqu, unbeknownst to the user. Duqu's delivery method is not precisely known, however it is believed that the malicious file was sent to the victim via a specially crafted and targeted email to the particular victim [34].

The threat made use of a zero-day TrueType font parsing vulnerability (CVE-2011-3402), which was used as the initial attack vector. This vulnerability is a security flaw in the Windows kernel, specifically in the TrueType font parsing engine in win32k.sys. The vulnerability allows attackers to execute arbitrary code remotely through crafted font data in a Microsoft Word document or through a web page. Duqu was also suspected by researchers to have used a compromised key to generate a valid digital signature for one of its drivers [34].

Duqu consists of three files: a driver, a main .dll file, and an encrypted configuration file. It's architecture is similar to Stuxnet's, and contains common exported functions. At the time of writing, the vulnerability exploited by Duqu's shellcode which initiates the installation process is undisclosed by Symantec [34] because it has not yet been patched. When a user opens the malicious Microsoft Word document on the victim machine, kernel mode shellcode will execute and check for a particular value in system registry to determine if the machine has already been infected. If not, the shellcode will decrypt two executable files from within the malicious Word document: the driver file and the installer .dll. Then the shellcode will run the .dll driver file, which will inject code into services.exe running on the victim machine. The installer .dll file is executed before the shellcode replaces itself with all zeroes, as if it never existed. After installation is completed, the only files that will remain on the victim system are the main .dll file and the .dll driver file, along with another configuration file.

The installation process will not begin immediately. Instead, the threat will wait until the computer is idle for ten minutes and will only proceed if other conditions hold. Once these conditions are met, the installer .dll file will decrypt three files from within itself: the main .dll file of Duqu, a .sys driver file that acts as the load point, and a configuration file. At this time, the installer will check two timestamps from within the decrypted configuration file. If the current date on the system does not fall within the two timestamps, installation will terminate. Otherwise, the installer will pass execution to Duqu's main .dll file by hooking the ntdll.dll file to a trusted process that is currently running on the victim machine, similarly to Stuxnet.

The main .dll file will use one of the malware's exported functions to place the load point driver into a system folder and create a process that will load it every time Windows starts up. Then the main .dll file is encrypted and placed in the %Windir%\\inf\\ folder of the victim machine. This file will be decrypted and executed by the driver whenever the system boots. Finally, the main .dll file reads a configuration file from within itself, encrypts it, and places it in the same %Windir% folder. During this

entire installation process, Duqu remains decrypted only in memory, except when the load-point driver is written to disk.

Now, the main .dll file will test the victim machine's internet connection by attempting a DNS lookup for a domain which is stored in the threat's configuration data. There are many variants of Duqu, but the one analyzed by Symantec used the domain Microsoft.com, and it is likely that the other variants used common domain names (with the victim machine in mind) for this purpose to be as stealthy as possible. If this DNS lookup fails, the threat will perform an additional DNS lookup for kasperskychk.dyndns.org.

The main .dll file will then inject itself into one of four trusted running processes on the victim machine. The threat will scan running processes, and depending on the security products that run on the victim machine from this scan, it will inject itself into an appropriate process to evade malware detection. It is also possible for the malware to stop and not inject itself into any process if the victim machine is running either Etrust v5 or v6 [34].

The threat then decompresses the .zdata dll which is used for command and control communications. The purpose of Duqu's command and control communication is to update the threat, and to download auxiliary modules to collect data, and then exfiltrate the collected data to the command and control servers. The downloaded modules could either be executed in memory, or they could be encrypted and then written to the victim machine's local disk. Command and control communication takes place using either HTTP over port 80 or HTTPs over port 443, or directly sending network traffic over port 443. Duqu also used a peer-to-peer protocol so that only one machine was connected to the command and control servers at a time. For this type of communication, usually an SMB protocol was used [34]. Duqu used its own custom encrypted protocols for communication. In particular, to exfiltrate data, Duqu would append its collected data to a small .jpg file before sending through either HTTP or HTTPS. When traffic was sent directly through port 443, the custom Duqu protocol was used, without encapsulation with HTTP or HTTPS, and an additional eight bytes were prepended to the packet, and used by the receiving command and control server as a validation key. Duqu would request modules from, and send data to, the command and control servers using HTTP GET and HTTP POST requests, respectively.

Duqu's custom communication protocol is similar to TCP. It has the capabilities to fragment packets, reorder, and it uses sequence and acknowledgement numbers to handle duplicate and missing packets. The data stream that is sent has a 12-byte header that starts with the characters "SH" in ASCII. This header is followed by segments of data with assigned sequence and acknowledgement numbers. The data segments can be encrypted with AES-CBS, compressed with LZO and potentially another custom compression algorithm. The AES key that is used is hardcoded in the malware itself, and different variants of Duqu used different AES keys. It is interesting to note that the AES keys were never sent to the command and control servers from the clients, but since there were only a small number of infections of Duqu, it was feasible for attackers to guess the key used upon receiving communications from victim machines.

A sample communication run may proceed as follows. The victim client will make a simple HTTP GET request to the command and control server with a Keep-Alive header and a hardcoded IP address for the Host field. Upon receipt of this request, the server will check the Cookie field of the request, which is to be unique for every such request, and only proceed if it is valid, sending an HTTP 200 OK response that contains a small blank (white) .jpg file. The client then similarly ensures the fields in this received response are valid, and will send an HTTP POST request with a .jpg file contained in the .data dll, appended by the desired data, encrypted and compressed, to exfiltrate to the command and control server. The command and control server will then respond with another HTTP 200 OK response.

One of the modules that Duqu will download from the command and control servers is a keylogger. This keylogger not only logs keystrokes, but also regularly saves screenshots and other information. The keylogger stores the information that it collects using XOR encryption. Using this keylogger, attackers will learn more about the target network and accumulate passwords of various devices and computers.

Attackers can use this information to compromise other computers on the network and manually spread malware across the network by copying it into network shares. Then, newly infected target computers will be instructed to check a configuration file related to Duqu. This configuration file will instruct the infected computer to first connect back to the computer that the infection came from, rather than communicating with the command and control servers directly. A scheduled task is created on the new computer, which will run the copied version of Duqu from the new computer, so that it may connect directly with the command and control servers at a later time. Researchers have determined, based on observed SSH connection patterns, that the connections to command and control servers by victim machines on a target network seem automated. Almost every time communication drops, a new connection is immediately established [34].

Duqu's Actions on Objectives phase of the kill chain included information stealing activity, particularly targeting industrial infrastructure and system manufacturers, amongst other organizations not in the industrial sector. The attackers seem to have been looking for design documents in order to launch a future attack on different target industries, including industrial control system facilities, which is one of the reasons why Duqu is believed to be a precursor to Stuxnet [34]. The stolen information from these organizations was also used to compromise more systems and devices on the target networks in order to obtain even more information.

# 5 Towards abstract models and feature selection for detection

Based on the diagrams presented in Section 4, we obtained a holistic understanding of the behavior APTs exhibited. We have detailed every activity, and based on the kill chain framework, we created models in order to identify similarities. We presented three types of models for every case study. The first was a verbose, high level kill chain model, which focused on the important and potentially detectable events that occur within each stage of the kill chain. This model summarizes the overall characteristics of malware and is instrumental in designing the next two models. The second model is an LTS diagram which focuses on the events that occur as the threat is transitioning between phases of the kill chain, rather than events occurring within a phase. This model is more compact allowing for higher level analysis of how the threat operates. This model type is fundamental in determining similarities and differences between the malware. The final model type is an MSQ diagram which displays the command and control communication activity between the victim client infected by the corresponding malware and the command and control server. This model is designed to assist us in determining what type of behavior we could expect to find in network data that would be indicative of an APT infection. [1]

## 5.1 Abstract verbose high-level kill chain model

This abstract, verbose high-level kill chain model covers many of the important, potentially detectable, and common behaviors of the APTs studied in this paper. Although some of the APT models we created have multiple occurances of particular stages, we focused on the seven stages of the kill chain, since these extra stages were not shared by all APTs. Generally, the extra stages corresponded to unique activity of the particular APTs, for example, Duqu's custom peer-to-peer protocol, which we modeled with an extra Command and Control phase. We also made some of the transitions more generalized, eliminating the need for the epsilon transitions. For example, in the transition from the Delivery phase to the Exploitation phase, we named the transition "File.Open", as the malware is run in some way on the machine, and this could either be through a USB, other removable drive, or another method, such as an attacker copying a

[1] All malware models with additional explanation may be found at https://github.com/CheyenneAtapour/MalwareModels.

malicious file to a network share on the victim network. Using this design decision, the transition may apply for these alternative delivery methods.

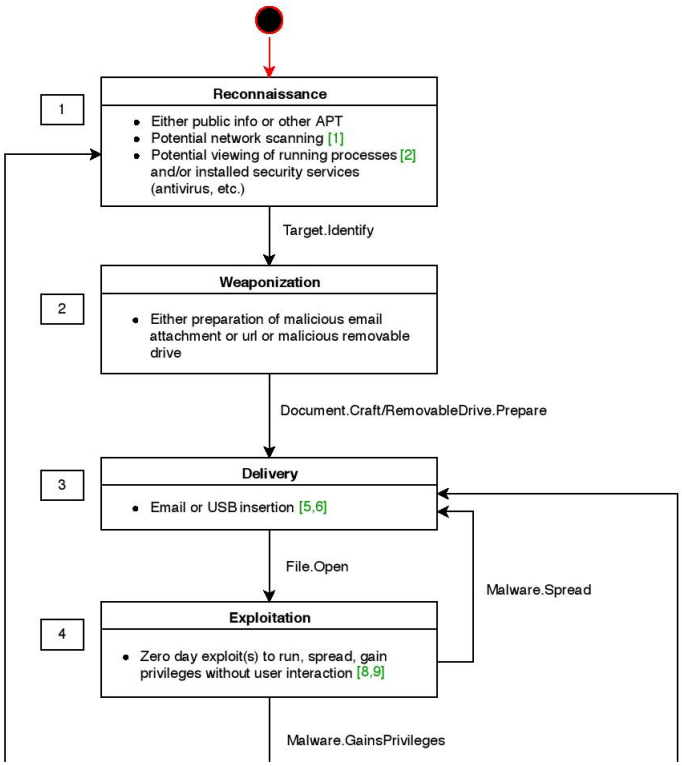



Figure 7: Abstract Verbose Diagram Part 1 detailing steps 1, 2, 3 and 4 of the Kill Chain

## 5.2 Abstract LTS model

The creation process of this abstract LTS model was similar to that of the previous LTS models, except this time, the abstract verbose model was used as a reference. We have included a generalized zero day exploit which is represented by a labeled circle with an incoming epsilon transition. The label also captures the potential for APTs to use more than one zero-day exploit, as was the case with many of the studied APTs.

## 5.3 Abstract MSQ for Command and Control

Our abstract Command and Control MSQ diagram takes inspiration from the other Command and Control MSQ diagrams we have created for the other APTs. The overall behavior which we found common in the majority of the studied APTs was essentially testing for an internet connection before entering a communication loop with the command and control servers, which involved the exfiltration of data, and the receipt of modules and instructions.

## 5.4 Determining features for detection

Using the malware abstract models and information collected from the case studies, we determine a set of features that have the potential to detect APTs from non-malicious network traffic, and to distinguish APT characteristics with specific focus on command and control behavior over the network. We further

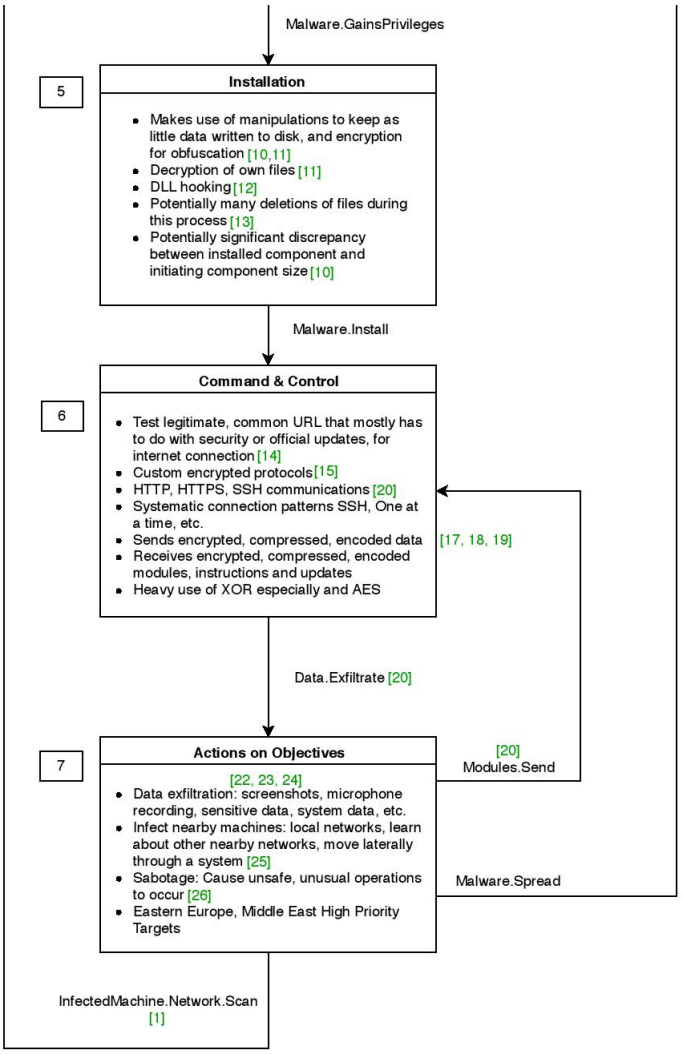



Figure 8: Abstract Verbose Diagram Part 2 detailing steps 5, 6 and 7 of the Kill Chain

identify features which have the potential to detect APT behavior from system logs, and through other forms of data, such as binary analysis of files. We suggest forms of data that may be used to further detect the presence of APTs that are not regularly collected, or made publicly available for use in research. For the purposes of this paper, we focus mostly on the features which show up in network traffic for use in our detection method, as this is the type of data we are able to find publicly available. We present all the features considered for use in order of, and separated by, the stages of the kill chain.

In the Reconnaissance stage, we consider four features; one of which could be detected through network data, while the other three would probably require binary analysis or access to system logs. The first feature is network scanning; usually, an APT that performs network scanning will start this activity after a machine is infected, in order to understand the network topology and target the next victim machine. There are tools to detect network scans from pcap data, such as snort rules, and we can use results from these rules as a feature in our detection method. However, network scanning is not a feature that is unique to the behavior of an APT, and may, for example, be performed by any malware trying to execute an SSH brute force attack. Furthermore, an APT may not perform a network scan because its intent may only be to infect a particular set of machines. As we showed in our case studies

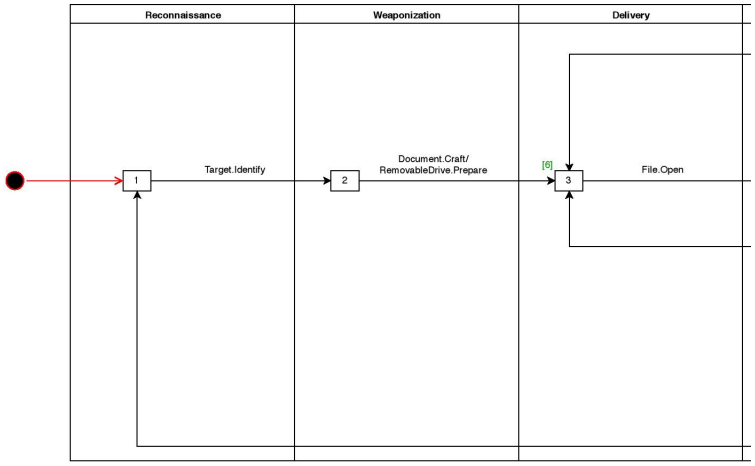



Figure 9: Abstract LTS Diagram Part 1

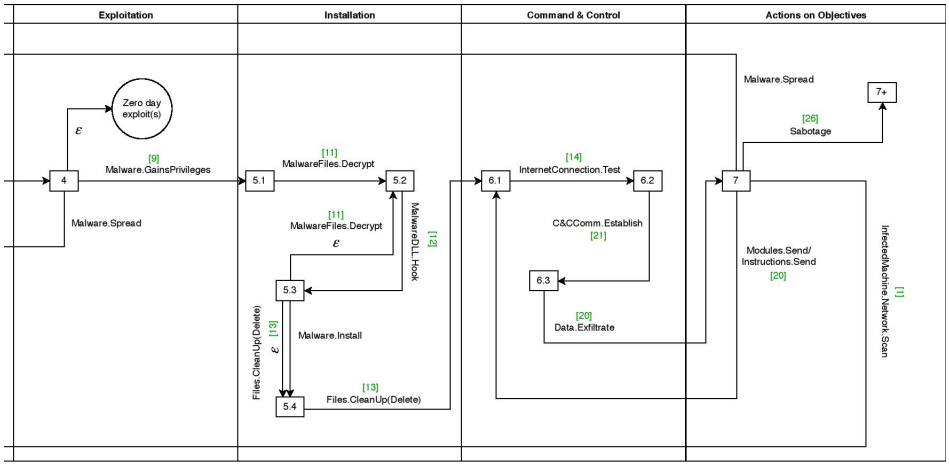



Figure 10: Abstract LTS Diagram Part 2

there were instances where the attackers had knowledge of the network topology based on IP addresses. Thus, this would not necessarily be a strong feature to use, unless we can be confident that an APT has already performed activities elsewhere on the network, and is now trying to spread itself.

Other features in the Reconnaissance stage involve malware requesting information about running processes and installed security services, identifying system and OS information, and searching for 'magic' numbers in files. These are common characteristics of APTs as we demonstrated through our case studies, and in particular, the progression of an a APT may stop if certain conditions in the victim

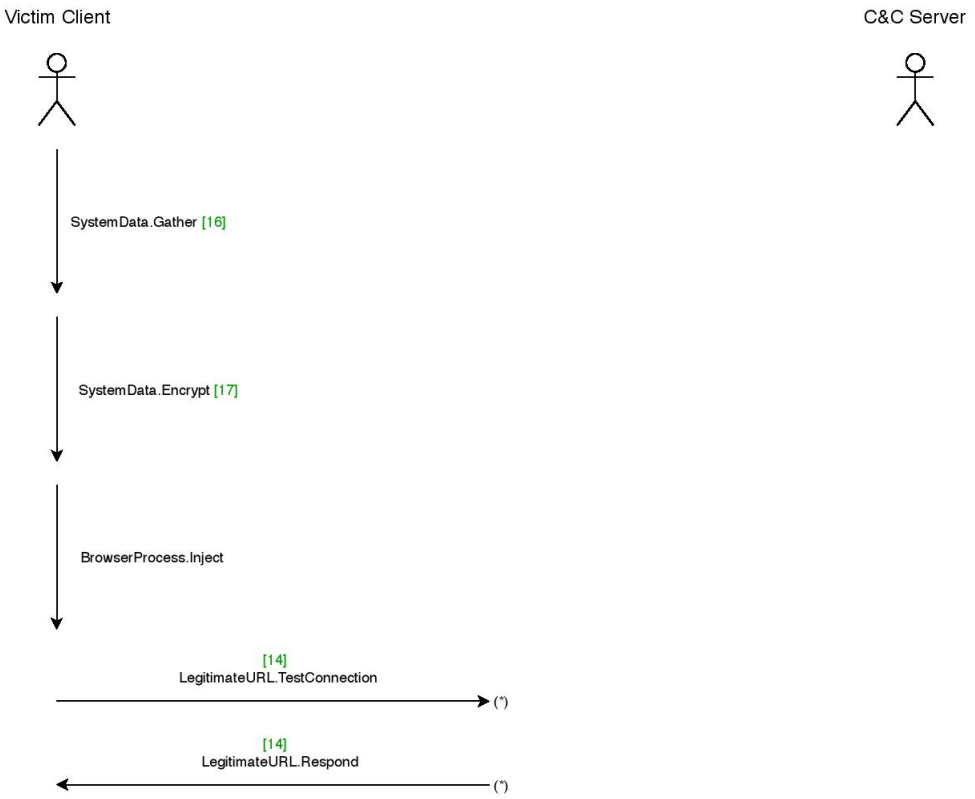



Figure 11: Abstract MSQ Diagram Part 1

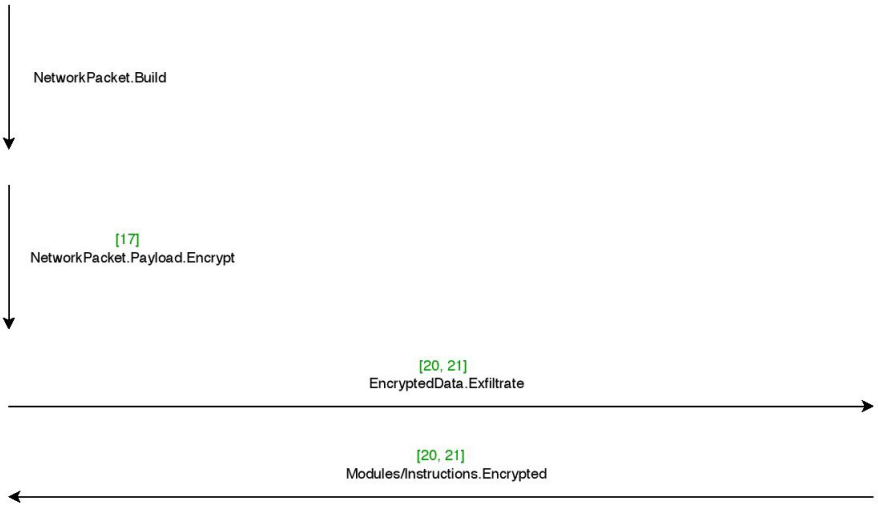



Figure 12: Abstract MSQ Diagram Part 2

machine are met. However, these features would not be detectable through network data, requiring access to system logs (i.e., file accesses) and/or binary analysis. The aforementioned features capture behavior which other malware may exhibit. There are some subtle differences, however, since the WannaCry attack checks filenames on the local machine to determine whether to continue. Furthermore, WannaCry

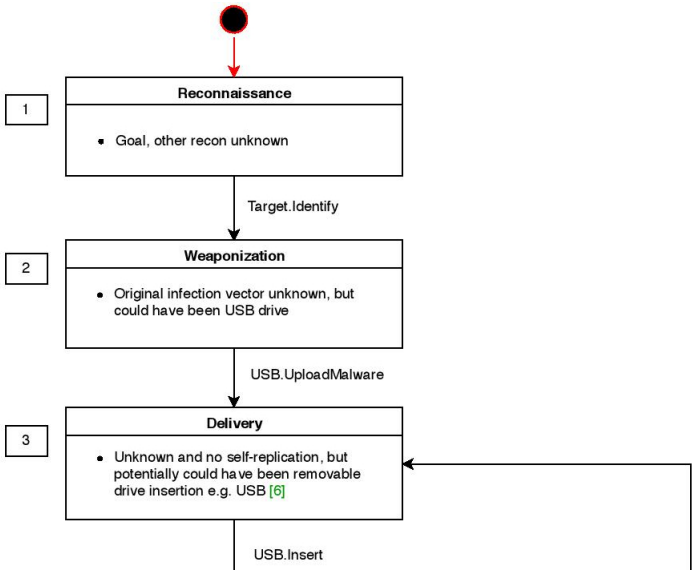



Figure 13: Gauss Kill Chain Diagram Part 1, illustrating steps 1,2 and 3

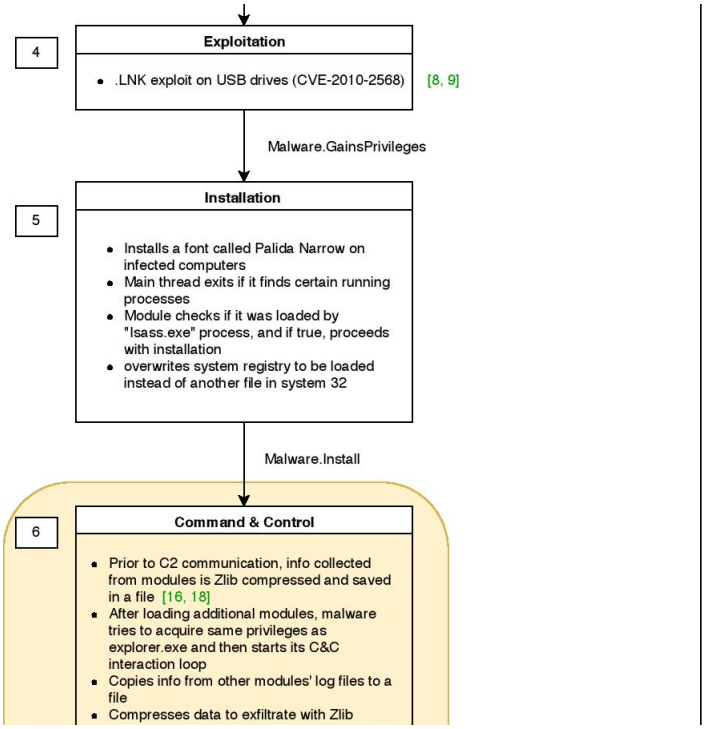



Figure 14: Gauss Kill Chain Diagram Part 2, illustrating steps 4,5 and 6

will not check local system or OS information, nor will look for 'magic' numbers contained inside files [35, 36].

APTs are commonly delivered through email or removable drives, so our features in the Delivery stage include strange email sender address, strange removable drive insertion, and default, random, or empty values in SSL certificate fields supplied by hosts outside an organization's monitored network. The first activity can be detected by using network data of a compromised organization over a sufficiently

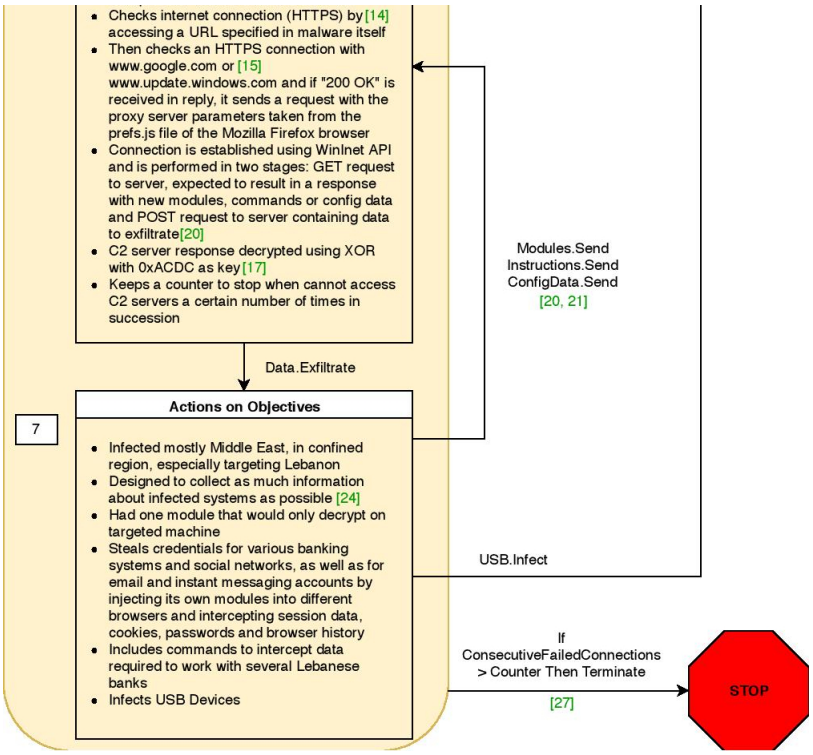
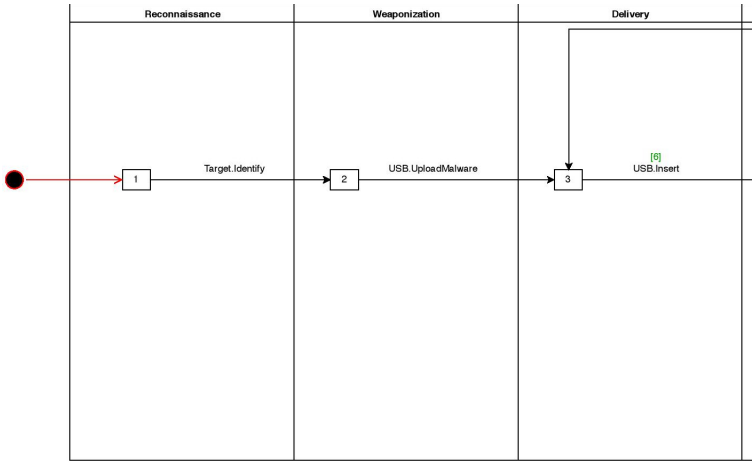



Figure 15: Gauss Kill Chain Diagram Part 1, illustrating step 7

long period of time. One can observe the header fields of email packets and compose a list of email sender addresses. Then, an email with a new, strange sender address can be identified with mainstream anomaly detection techniques. Email activity should then be linked to other characteristics of an APT. In fact, this is a common delivery method for many types of malware, so relying heavily on a feature like this for APT detection may lead to false positives. It is important to note that not all APTs use this delivery method, and if they do, the email sender address could be spoofed and designed to look similar to a particular legitimate address from a person within the victim organization. Finally, a strange email sender address may correspond to a new client that an organization is working with, and thus be a by-product of non-malicious behavior. Identifying removable drives requires a system log of inserted removable drives and a list of approved removable drives. In a similar vein, any new removable drive inserted into a machine can be flagged as malicious. This feature has the potential for false positives in the form of human neglect. The presence of SSL certificates can be detected by using a rule for network traffic.

In all our case studies, APTs utilize zero-day exploits to elevate privileges and run the malware. This characteristic cannot be captured by a single feature because it is not known beforehand what type of data to observe, and what to observe in the data. Side effects of this behavior can be observed, such as a file running on an open request and triggering other applications without explicit user input or escalation of privileges in a number of processes. The former action would require access to system logs for running processes. One could establish which processes usually trigger other processes to run, and use an anomaly detection approach to determine when a process triggers another unusual process. The latter would perhaps require logs of running processes as well as their privilege level. With this data, an anomaly detection approach can determine which privileges certain processes should have, and identify unusual escalation of privileges. Alternatively, a list of machines or users with associated privileges can be maintained to indicate when a privilege is wrongfully granted. The main problems with these features,




Legend

:= This transition may be made from any of the states which share its color

1 : Reconnaissance
2 : Weaponization
3 : Delivery
4 : Exploitation
5 : Installation
6 : Command and Control
7 : Actions on Objectives

Figure 16: Gauss LTS Diagram Part 1

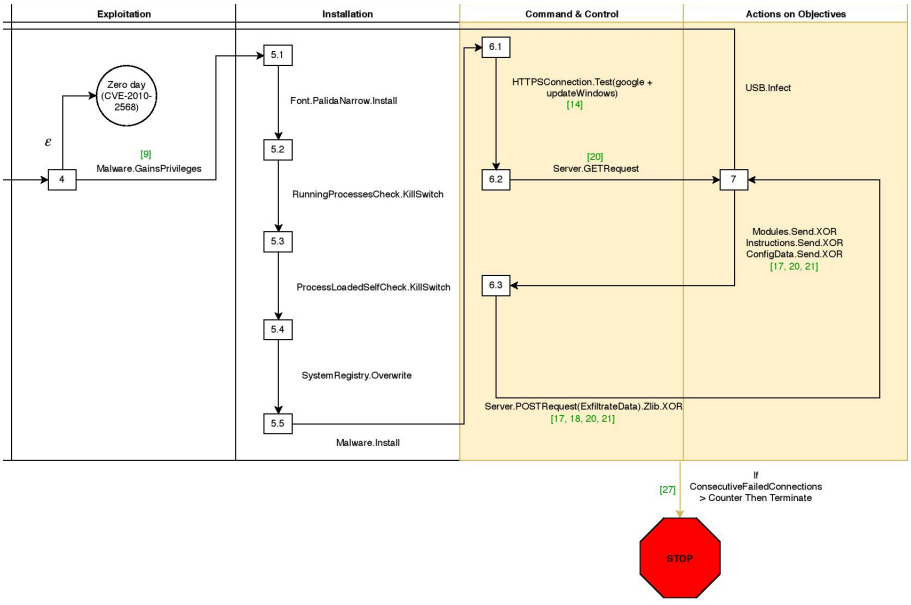



Figure 17: Gauss LTS Diagram Part 2

besides the lack of data, is that they describe behavior which is not unique to APTs, and could potentially be exhibited by other types of malware.

In contrast to other malware, APTs will strive to remain undetected for a long period of time. behavior that corresponds to this priority is captured in the Installation phase. In most of our case studies,

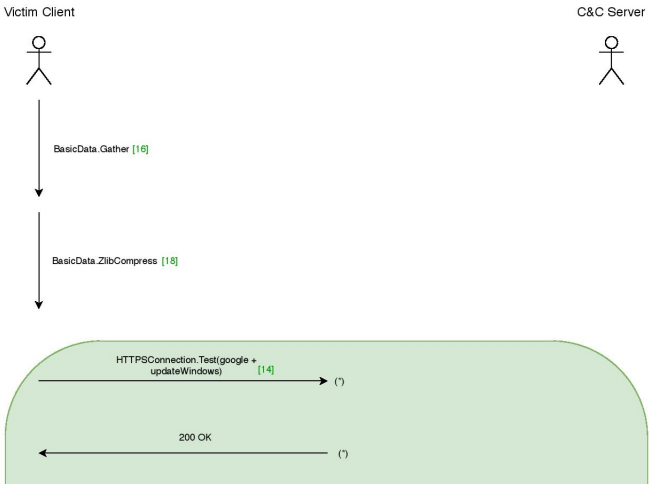



Figure 18: Gauss MSQ Diagram Part 1

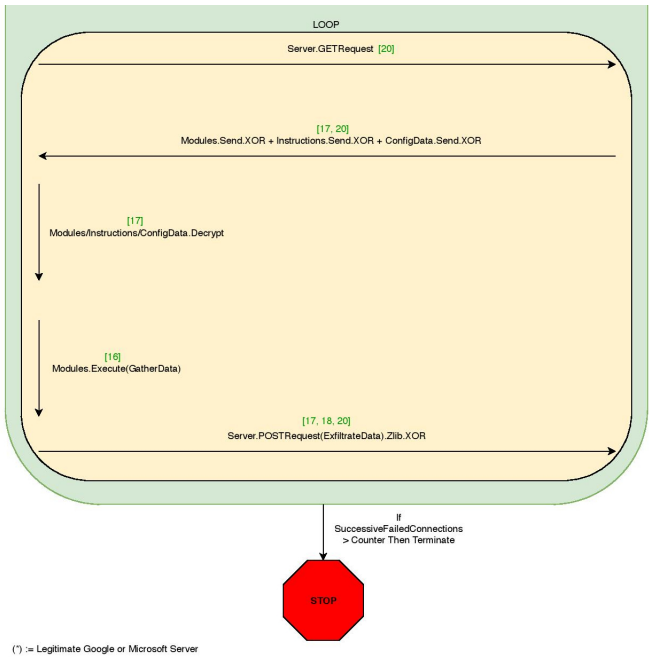



Figure 19: Gauss MSQ Diagram Part 2

APTs tried to avoid writing to disk and placed themselves in RAM. For this reason, the first feature we have selected for detection is virtual size and raw data size discrepancies. Comparing the size of a file to be installed to the total size of files written to disk during this installation, should flag large discrepancies as suspicious. In particular, this behavior may be indicative of sophisticated packers, non-well-behaved compilers, or dynamic unpacking within memory during execution, which is behavior that would potentially otherwise require binary analysis to identify. During installation, APTs may also perform encryption, decryption, DLL hooking, and file deletions. The first three actions may require binary analysis to detect. In particular, for encryption and decryption, one can observe certain libraries in the codebase used for powerful encryption. File deletions may potentially occur with any installation as part of clean up, however, APTs seem to delete files throughout the installation process, and not during

one specific phase in installation. Thus, using system logs that show file deletion timestamps as well as installations, one may determine an installation that is correlated with sporadic file deletions, and this may be flagged as suspicious activity.

In the Command and Control phase, the features we have identified include: attempted internet accesses (especially from machines that do not normally connect to internet), custom encrypted protocols, file accesses, encryption (especially XOR), compression, encoding, a chronological loop of file accesses (file accesses, compression, encryption), HTTP GET request, HTTP POST request followed by repeating GET requests and communication with unknown/strange IP addresses. For machines that do not usually connect to the internet, system log information based on network packets can reveal unusual internet access. Network traffic data over a long period of time for an organization can determine whether a strange URL was accessed (one that differs from URLs accessed in the past). Network data could also reveal traffic through unusual ports as suspicious, as well as incorrect headers on packets such as HTTP GET requests.

Non-ASCII characters in these types of packets may be indicative of the use of a custom encrypted protocol for command and control communication. To determine file access activity taking place in this stage of the kill chain, network data can be combined with system logs. Patterns identified where file access is followed by compression, encoding and encryption with a subsequent HTTP GET request, are indicative of APT behavior. This specific pattern, which may be repeated several times is a strong indicator of an APT's command and control activity. Compression, encoding and encryption would possibly require binary analysis to observe. Finally, communications with unknown/strange IP addresses can be determined given network data for an organization from a sufficiently long amount of time, and observing network packets with sender/receiver headers that do not match that of traffic seen in the past data.

In the final kill chain step, Action on Objectives, we identified the following features: unusual processes taking screenshots, recording microphones, accessing files, computers on the same network exhibiting similar behavior, unusual, unsafe operations on hardware, hardware checks, location checks, and timestamp checks. Detecting unusual processes would require system logs that contain information about when screenshots were taken from webcams, microphones which started and ended recordings, and file access logs with associated timestamps. It would also be beneficial to have data that captures the name of the process(es) that triggers these activities. It should be possible to distinguish processes which are responsible for triggering certain activities and identify if any new, unusual process triggers any of these activities. Suspicious behavior on a specific machine can then be correlated to the behavior of other machines to establish an overall unusual activity on all machines present in the same network. Unusual hardware behavior may be identified using system logs and an organization's rules on safe hardware configuration. For this activity, it would be very important to obtain and store logs in different ways to ensure integrity.

# 6 Validation of abstract models and selected features

In order to validate the abstract models we designed and the appropriateness of the selected features in distinguishing APT network behavior, we examine another APT and try to capture whether or not our models and features successfully describe its characteristics. Note that in a future work, we could also validate the abstract models through formal model verification by designing a signature-based detection system inspired by the features presented in this paper, and running it on data generated by various APTs. The case study we chose is Gauss because the security community has provided in-depth analysis of its command and control activity.

| Number | Kill-Chain Stage | Feature |
|---|---|---|
| 1 | Reconnaissance | Network scanning |
| 2 | Reconnaissance | Checks for Running Processes/Installed Services |
| 3 | Reconnaissance | Identifying OS/System info |
| 4 | Reconnaissance | Highly specific content searching (”magic numbers”) |
| 5 | Delivery | Strange email sender address |
| 6 | Delivery | Strange removable drive insertion |
| 7 | Delivery | Strange SSL certificate fields |
| 8 | Exploitation | Unusual application launch |
| 9 | Exploitation | Unusual privileges/escalations |
| 10 | Installation | File size discrepancy |
| 11 | Installation | Encryption/Decryption |
| 12 | Installation | DLL Hooking |
| 13 | Installation | File deletions |
| 14 | Command & Control | Internet access |
| 15 | Command & Control | Custom encrypted protocols |
| 16 | Command & Control | File accesses |
| 17 | Command & Control | Encryption/Decryption |
| 18 | Command & Control | Compression |
| 19 | Command & Control | Encoding |
| 20 | Command & Control | HTTP GET/POST requests |
| 21 | Command & Control | Strange IP address comms |
| 22 | Actions on Objectives | Screenshots |
| 23 | Actions on Objectives | Microphone recording |
| 24 | Actions on Objectives | File accesses |
| 25 | Actions on Objectives | Correlated behavior of local machines |
| 26 | Actions on Objectives | Unusual hardware operations |
| 27 | Actions on Objectives | Hardware checks |
| 28 | Actions on Objectives | Location checks |
| 29 | Actions on Objectives | Timestamp checks |

Table 2: List of novel features for APT detection based on the Abstract models.

Gauss activity was occurred between September and October of 2011 [37]. The majority of infected targets were located in the Middle East, with many targets in Lebanon. Gauss was a targeted malware, which resulted in being fairly widespread and aimed in information stealing [27].

The first three kill chain stages for Gauss are not precisely known. Reconnaissance is part of Gauss' Actions on Objectives, but it is not known if other reconnaissance activity was performed by attackers before launching the threat. Similarly, it remains unknown exactly what activities attackers performed in regards to the initial Weaponization stage of the kill chain. It is speculated that a USB drive was used as the original attack vector [37]. If this is the case, then the Delivery phase of the kill chain would involve insertion of an infected removable drive into a victim machine.

Gauss made use of an .LNK exploit on USB devices related to the CVE-2010-2568 vulnerability [37]. This exploit is similar to the one used by Stuxnet (CVE-MS10-046) [37] and enabled Gauss to mask its Trojan's files on a USB drive without using a driver.

Gauss performed some unusual installation activity for which the purpose is not known [27, 37]. This involved installing a particular text font called Palida Narrow on infected machines. It is speculated that this may be related to the True Type font parsing vulnerability that Duqu exploited, but researchers at Kaspersky labs were unable to find conclusive evidence [38]. Furthermore, researchers have developed a method to detect Gauss infections on victim machines by using the presence of this installed font [27]. Perhaps Gauss also uses this information to determine if it has already infected a potential victim.

The rest of Gauss's installation process involves checking if the ShellHW module (the main module) was loaded by the "lsass.exe" process running on the victim machine, which is a requirement to continue the installation. If everything is in place, Gauss will write itself in files within the System32 directory of the victim machine, and modify the system registry to load the attack files when the victim machine boots. This main module automatically loads into processes that use wbemsvc.dll, and if process svchost.exe was started with a parameter value of "-k netsvc", it will start its main thread. The main thread will look for certain processes running on the victim machine, and it will terminate if any are found. The main module will then read an encrypted configuration file in the victim system registry in order to load additional modules. It will then try to elevate its privileges to that of the explorer.exe process. Afterwards, Gauss will copy all of its log data from its other modules to the file " shw.tmp" and compress this file using zlib [27, 37]. Gauss will then attempt its first command and control activity.

Gauss will first check internet connection using HTTPS by attempting to access URLs which are hardcoded into the malware. It will try to make an https connection with either www.google.com or www.update.windows.com . If this is successful, Gauss will send a request with the proxy server parameters from the prefs.js file of the Mozilla firefox browser on the victim machine [37]. Once this succeeds, the threat will establish a connection with the command and control server. Communication proceeds as follows: the victim machine will send an HTTP GET request to the command and control server; then the command and control server will respond with modules, commands, and/or configuration data, which is XOR encrypted; next, Gauss will send an HTTP POST request to the command and control server containing the collected data from the victim machine in the file " shw.tmp"; this exfiltrated data is zlib compressed. During this communication, Gauss will maintain a counter which is decremented whenever an attempted connection to the Command and Control server fails. Communication will cease when the counter becomes zero, and the counter is reset whenever communication succeeds.

Gauss is able to steal credentials for various banking systems, social networks, email, and instant messaging accounts by injecting its modules into different browser processes running on the victim machine [37]. After this injection, the threat is able to view session data, cookies, passwords, and browser history. Gauss also has capabilities to infect USB devices, and has specific commands to intercept data required to work with several Lebanese banks [37]. Interestingly, Gauss contains one particular module called Gödel, which can only be decrypted on the specific target machines for which it was intended. This module is responsible for infecting USB drives and stealing data [37].

## 6.1 Models for Gauss and comparison with the abstract models

A comparison of the generalized, abstract diagrams to diagrams modeling Gauss, shows that there are small differences. Therefore, we believe that the abstract models can be expressive enough to describe APTs we have not studied. Figures 16, 17, 18, 19 present the LTS and MSQ models for Gauss.

Although we examined a small number of APTs, our confidence is informed by the fact that we studied APTs with diverse characteristics, for which a plethora of high quality technical descriptions were available. Furthermore, since APTs are highly sophisticated attacks, there is a very small sample in our disposal to examine. As was expected, the minor differences in diagrams are due to some unique behavior exhibited by Gauss. Areas that could further enhance the richness of the abstract models can

consider behavior relating to how APTs check certain running processes, installed security services, and other information regarding victim machines before continuing during certain stages.

# 7 Conclusion and Future Work

Advanced Persistent Threats (APTs) are characterized by their complexity and ability to stay relatively dormant and undetected on a computer system before launching a devastating attack. Numerous unsuccessful attempts have utilised machine learning techniques and rule-based technologies to try and detect these sophisticated attacks. In this paper, we opted for a more theoretical approach to identify unique APT characteristics, distinguishable from other multi-stage attacks. We examined four case studies of well-known APTs and created three models which provide different insights for every case study. We further identified common characteristics across all models to synthesize abstract models able to describe generic APT behavior. Furthermore, we were able to derive key features which are unique to APTs, and can be detected using publicly available network traffic data. Throughout this process, we have also been able to suggest particular areas where more data may be collected on computer systems in order to facilitate APT and potentially other malware detection. We finally validated our abstract models and the novel features by formalizing a fifth case study. Our formalizations indicate that our models are expressive enough to capture the majority of the characteristics which the APT exhibited in this fifth case study. As part of our future work, we intend to develop an intrusion detection system which will rely on the features and characteristics identified in this paper.

# References

[1] M. Gasser, *Building a secure computer system*. Van Nostrand Reinhold Co, 1988.

[2] Techopedia, "What is an advanced persistent threat (apt)? - definition from techopedia 2018," https://www.techopedia.com/definition/28118/advanced-persistent-threat-apt, [Online; Accessed on December 1, 2018], 2018.

[3] P. Bhatt, E. T. Yano, and P. Gustavsson, "Towards a framework to detect multi-stage advanced persistent threats attacks," in *Proc. of the 2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE'14), Oxford, UK*. IEEE, April 2014, pp. 390–395.

[4] National Cyber Security Centre, "Advisory: russian state-sponsored cyber actors targeting network infrastructure devices," https://www.ncsc.gov.uk/alerts/russian-state-sponsored-cyber-actors-targeting-network-infrastructure-devices, [Online; Accessed on December 1, 2018], April 2018, alerts and Advisories.

[5] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proc. of the 29th Annual Computer Security Applications Conference (ACSAC'13), New Orleans, Louisiana, USA*. ACM, December 2013, pp. 199–208.

[6] G. Stringhini, Y. Shen, Y. Han, and X. Zhang, "Marmite: spreading malicious file reputation through download graphs," in *Proc. of the 33rd Annual Computer Security Applications Conference (ACSAC'17), Orlando, Florida, USA*. ACM, December 2017, pp. 91–102.

[7] B. J. Kwon, J. Mondal, J. Jang, L. Bilge, and T. Dumitras, "The dropper effect: Insights into malware distribution with downloader graph analytics," in *Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15), Denver, Colorado, USA*. ACM, October 2015, pp. 1118–1129.

[8] L. Invernizzi, S. Miskovic, R. Torres, C. Kruegel, S. Saha, G. Vigna, S.-J. Lee, and M. Mellia, "Nazca: Detecting malware distribution in large-scale networks." in *Proc. of the 2014 Network and Distributed System Security Symposium (NDSS'14), San Diego, California, USA*, vol. 14. Internet Society, February 2014, pp. 23–26.

[9] "Snort.org. (n.d.). snort community rules sid 1-359," https://www.snort.org/rule_docs/1-359, [Online; Accessed on December 1, 2018], 2018.

[10] "Snort.org. (2018). snort community rules sid 1-358," https://www.snort.org/rule_docs/1-358, [Online; Accessed on December 1, 2018], 2018.

[11] "Snort.org. (2018). snort rules and ids software download," https://www.snort.org/downloads#rules, [Online; Accessed on December 1, 2018], 2018.

[12] J. Zhang, C. Seifert, J. W. Stokes, and W. Lee, "Arrow: Generating signatures to detect drive-by downloads," in *Proc. of the 20th international conference on World Wide Web (WWW'11), Hyderabad, India*. ACM, April 2011, pp. 187–196.

[13] M. A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, and N. Provos, "Camp: Content-agnostic malware protection." in *Proc. of the 2013 Network and Distributed System Security Symposium (NDSS'13), San Diego, California, USA*. Internet Society, February 2013.

[14] P. Giura and W. Wang, "A context-based detection framework for advanced persistent threats," in *Proc. of the 2012 International Conference on Cyber Security (CyberSecurity'12), Washington, DC, USA*. IEEE, December 2012, pp. 69–74.

[15] N. Virvilis and D. Gritzalis, "The big four-what we did wrong in advanced persistent threat detection?" in *Proc. of the 2013 Eighth International Conference on Availability, Reliability and Security (ARES'13), Regensburg, Germany*. IEEE, November 2013, pp. 248–254.

[16] J. de Vries, H. Hoogstraaten, J. van den Berg, and S. Daskapan, "Systems for detecting advanced persistent threats: A development roadmap using intelligent data analysis," in *Proc. of the 2012 International Conference on Cyber Security (CyberSecurity'12), Washington, DC, USA*. IEEE, December 2012, pp. 54–61.

[17] S. J. Yang, A. Stotz, J. Holsopple, M. Sudit, and M. Kuhl, "High level information fusion for tracking and projection of multistage cyber attacks," *Information Fusion*, vol. 10, no. 1, pp. 107–121, January 2009.

[18] A. Rice, "Command-and-control servers: The puppet masters that govern malware," SearchSecurity, 2014, https://searchsecurity.techtarget.com/feature/Command-and-control-servers-The-puppet-masters-that-govern-malware, [Online; Accessed on December 1, 2018].

[19] L. S. Barbosa, "Labelled transition systems," 2018, http://www.win.tue.nl/~jschmalt/teaching/2IX20/reader_software_specification_ch_8.pdf, [Online; Accessed on December 1, 2018].

[20] "Message (sequence diagram)," https://www.sparxsystems.com/enterprise_architect_user_guide/8.0/modeling_languages/sequencemessage.html, [Online; Accessed on December 1, 2018], 2018, sparxsystems.com.

[21] B. Vigliarolo, "Stuxnet: The smart person's guide," https://www.techrepublic.com/article/stuxnet-the-smart-persons-guide/, [Online; Accessed on December 1, 2018], 2018, techRepublic.

[22] "Sans institute: reading room - industrial control systems / scada_2018," https://www.sans.org/reading-room/whitepapers/ICS/industrial-control-system-cyber-kill-chain-36297, [Online; Accessed on December 1, 2018], 2018, sans.org.

[23] E. Bursztein, "Does dropping usb drives really work?" Blackhat.com, 2016, https://www.blackhat.com/docs/us-16/materials/us-16-Bursztein-Does-Dropping-USB-Drives-In-Parking-Lots-And-Other-Places-Really-Work.pdf, [Online; Accessed on December 1, 2018].

[24] S. Nichols, "Half of people plug in usb drives they find in the parking lot," The Register, April 2016, https://www.theregister.co.uk/2016/04/11/half_plug_in_found_drives/, [Online; Accessed on December 1, 2018].

[25] N. Falliere, L. Murchu, and E. Chien, "W32.stuxnet dossier version 1.4," Symantec Corporation, 2011, http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, [Online; Accessed on December 1, 2018].

[26] Microsoft MSDN, "What is the windows integrity mechanism?" 2018, https://msdn.microsoft.com/en-us/library/bb625957.aspx, [Online; Accessed on December 1, 2018].

[27] B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi, "The cousins of stuxnet: Duqu, flame, and gauss," *Future Internet*, vol. 4, no. 4, pp. 971–1003, 2012.

[28] Crysys, "A complex malware for targeted attacks skywiper (aka flame) 2012," 2012, https://www.crysys.hu/skywiper/skywiper.pdf, [Online; Accessed on December 1, 2018].

[29] Kaspersky Securelist, "Full analysis of flame's command and control servers 2012," 2012, https://securelist.com/full-analysis-of-flames-command-control-servers/34216/, [Online; Accessed on December 1, 2018].

[30] Kaspersky Securelist, ""red october" diplomatic cyber attacks investigation 2013," 2013, https://securelist.com/red-october-diplomatic-cyber-attacks-investigation/36740/, [Online; Accessed on December 1, 2018].

[31] Rapid7, "cve-2011-3544 java applet rhino script engine remote code execution — rapid7," https://www.rapid7.com/db/modules/exploit/multi/browser/java_rhino, [Online; Accessed on December 1, 2018].

[32] Kaspersky Securelist, ""red october" detailed malware description 2. second stage of attack 2013," 2013, https://securelist.com/red-october-detailed-malware-description-2-second-stage-of-attack/36842/, [Online; Accessed on December 1, 2018].

[33] B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi, "Duqu: Analysis, detection, and lessons learned," in *Proc. of the 2012 European Workshop on Systems Security (EUROSEC'12), Bern, Switzerland*. ACM, April 2012.

[34] Symantec Corporation, "w32.duqu the precursor to the next stuxnet 2011," 2011, https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_duqu_the_precursor_to_the_next_stuxnet.pdf, [Online; Accessed on December 1, 2018].

[35] L. Newman, "The leaked nsa spy tool that hacked the world," WIRED, 2018, https://www.wired.com/story/eternalblue-leaked-nsa-spy-tool-hacked-world/, [Online; Accessed on December 1, 2018].

[36] Symantec Corporation, "Ransom.wannacry — symantec 2017," 2017, https://www.symantec.com/security-center/writeup/2017-051310-3522-99#technicaldescription, [Online; Accessed on December 1, 2018].

[37] Kaspersky Securelist, "Gauss: abnormal distribution_2012," 2012, https://securelist.com/gauss-abnormal-distribution/36620/, [Online; Accessed on December 1, 2018].

[38] Ethical Hacker, "Why does gauss install palida narrow font? — live hacking," August 2012, http://www.livehacking.com/2012/08/14/why-does-gauss-install-palida-narrow-font/, [Online; Accessed on December 1, 2018].

---

## Author Biography

**Cheyenne Atapour** is a current MBA candidate at the University of Oxford, who recently completed his MSc in Computer Science at the end of August 2018 at the same. In general, his research interests span cybersecurity, machine learning, and the design and analysis of algorithms. In June of 2016, he completed his Bachelor of Science in Computer Engineering with a minor in Mathematics at the University of California, San Diego.

**Ioannis Agrafiotis** is a Senior Cybersecurity Researcher at the Department of Computer Science and James Martin Fellow at the Global Cyber Security Capacity Centre (GCSCC). His research interests include capacity building in cybersecurity, risk analysis and resilience in the cyber domain, cyber insurance, and anomaly detection techniques for internal and external threats. Ioannis completed his doctoral studies in Engineering at the University of Warwick (2012, EPSRC-funded). He also holds an MSc in Analysis, Design and Management of Information Systems from the London School of Economics and Political Science (2008) and a BSc in Applied Informatics from the University of Macedonia in Greece (2006).

**Sadie Creese** is a Professor of Cybersecurity in the Department of Computer Science at the University of Oxford, where she teaches threat detection, risk assessment and operational aspects of security. Her current research portfolio includes threat modeling and detection, visual analytics for cybersecurity, risk propagation logics and communication, resilience strategies, privacy, vulnerability of distributed ledgers, and understanding cyber-harm. Sadie was the founding Director of the Global Cyber Security Capacity Centre (GCSCC) at the Oxford Martin School where she continues to serve as a Director conducting research into what constitutes national cybersecurity capacity, working with countries and international organizations around the world. She was the founding Director of Oxford's Cybersecurity network launched in 2008 and now called CyberSecurity@Oxford, and is a member of the World Economic Forum's Global Council on Cyber Security.