Milware: Identification and Implications of State Authored Malicious Software

Trey Herr, Eric Armbrust
The George Washington University

Abstract

The pervasive development and deployment of malicious software by states presents a new challenge for the information security and policy communities because of the resource advantage and legal status of governments. The difference between state and non-state authored code is typically described in vague terms of sophistication, contributing to the inaccurate confirmation bias of many that states simply 'do it better'. This paper attempts to determine if state authored code is demonstrably different from that written by non-state actors and if so, how. To do so, we examine a collection of malware samples which, through existing analytic techniques, have been attributed to a mix of state and non-state actors. Reviewing technical information available in the public domain for each sample, reverse-engineering a sub-set, we determine that there is a set of criteria by which state authored code can be differentiated from the conventional malware of non-state groups. This MAlicious Software Sophistication or MASS index relies on a set of characteristics which describe the behavior and construction of malware including the severity of exploits and customization of the payload. In addition to highlighting these particular differences, the paper discusses several policy implications which arise from identifying a separate class of state-authored code. This is an interdisciplinary effort and pilot project based on a limited dataset however the conclusions drawn have important ramifications for both the information security and relevant policymaking communities.

CCS Concepts

•Security and privacy → Formal security models;

1. INTRODUCTION

Malicious activity is the natural byproduct of technologies which, by default, assume trust between user and information. Stretching back to the design choice that instructions and information be considered the same but differentiated prior to execution, information security has faced a challenge in protecting the majority of users from a malicious minority. There is a recent trend towards more capable malware samples emerging in an otherwise slowly evolving sea of code, often with exotic propagation and compromise capabilities. A shorthand for these samples generally would be useful for analysts; a more comprehensive means of classifying them, relying less on the labor intensive and opaque efforts of information security firms, would serve to benefit the research and policymaking communities. These more capable samples represent an aberrant spike in an otherwise slow evolution over time. According to analysis from information security firms like Symantec and Kaspersky, much of this code originates with states. The implication for well resourced political entities as a more common source of malicious software samples is potentially significant.

Previous work presented has attempted to move beyond the simple "sophisticated/unsophisticated" dichotomy and succeeded in developing a metric that measured social engineering techniques but there remains more to be done. [1] We advance this effort by focusing on common functional characteristics of malicious code and comparing the work of state and non-state actors to better understand what depends on the unique operational demands of the state. Understanding the difference between state and non-state authored malicious software has important policy and academic implications. This paper reviews a number of samples in order to generate an index to consistently classify code as state or non-state authored. We reviewed malicious software for which there is material in the public domain, available through white papers or other technical analysis, and selected several different examples for more thorough analysis, described in detail in the Code Analysis section.

For policymakers, identifying state authored code can help to develop a more robust suite of attribution techniques and may highlight a growing divergence in the evolution of malicious software being authored by these two groups - contributing to what many commentators refer to as an "arms race". For analysts and the information security community, recognizing the degree of sophistication in state authored code more systematically than is current practice may contribute to greater awareness and more rapid recognition of new compromise techniques and infection vectors. Firms may use the information to make different insurance claims or solicit help from law enforcement bodies where it might not otherwise happen. Understanding quickly something is a state authored piece of code may help shape the security response rapidly at beginning rather than after an extended

NSPW'15 September 8–11, 2015, Twente, The Netherlands

ISBN 978-1-4503-3754-0/15/09…$15.00

DOI: http://dx.doi.org/10.1145/2841113.2841116

forensic effort. The academic community can benefit from this work as it served to provide a model for empirically sound investigations with cross-disciplinary teams and conclusions.

The paper starts by evaluating a collection of malicious software samples for which there is information available in the public domain and then proceeds to reverse engineer several more samples. From this analysis, the paper develops a rudimentary set of criteria that consistently distinguishes the samples identified as state or non-state; we collect these critiera into a MAlicious Software Sophistication (MASS) Index. Concluding, we use the evidence of this distinction (rather than the material differences) to suggest several policy implications and highlight opportunities for future work.

2. BACKGROUND

Malicious software analysis has long been focused on individually functional components in code, using information gleaned from particular modules to describe the function of an entire sample in a sound but somewhat limiting bottom up approach. Our method attempts to work top down, establishing three broad components of all malicious samples underneath the PrEP framework; Propagation method, Exploits, and Payload. [2]

Propagation is the act of delivering code to the target system; it could be an email attachment, compromised web site, or USB stick. Propagation may span multiple stages or include a so-called "dropper," code written specifically to land on a target system and phone home to a designated IP address to download other malware components. Propagation is the least software intensive piece of malware, anything that can hold or transmit data can propagate malware. A variety of propagation methods have been observed in the wild including compromised websites, email attachments, and removable storage media. Propagtion methods are sold on web based markets for a variety of services such as Pay Per Install, where a customer pays for the delivery of their payload to a population of target machines and pays the botnet owner any successful installations.

A payload is the core content of malware: malicious software designed to execute on a computer system and achieve some predefined goal such as compromising password files or deleting data. It could take advantage of code libraries present on a target system in order to execute and may combine several modules, each with a different but complementary purpose. The payload is malware's raison d'etre and can vary widely in design and objective.The amateur Wank worm, which infected computers running Digital Equipment Corporation (DEC)'s VMS operating system in the late 1980s, executed a simple operation to change user's passwords and display an anti-nuclear slogan. Stuxnet's payload by contrast, was designed to manipulate machinery into destroying itself. The payload of a piece of malicious software executes on a computer system in order to create some effect; to alter data, remotely activate a camera, create a 'backdoor' for future access, or damage hardware - these actions manipulate the intended function of an information system to achieve the effects desired by an attacker. The term cyber weapon, though misused widely, covers only those malicious software samples with payloads capable of generating destructive effects.

Aiding the propagation method and payload are software programs that take advantage of vulnerabilities in computer systems or surrounding networks called exploits. Exploits are code, written to take advantage of features or flaws in software, and enable the operation of other malware components, either the propagation method or payload. Exploits are the most commodity-like component in malware; bought and sold on the web, their need to be specific to a target system creates tremendous value with quoted prices ranging into the hundreds of thousands of dollars. Increasingly, companies such as VUPEN and a sizable collection of freelancers are selling newly discovered vulnerabilities and developed exploits to governments, including the NSA, and non-state actors rather than the original software vendors like Google.

Distinguishing between the payload and an exploit is important when thinking about their development. A payload is written to achieve a desired effect so focus is on the code's output. An exploit is written to a particular target, emphasizing the target software's structure and function. This distinction makes for divergent practices in developing, selling, and integrating exploits and payloads. Exploits are highly fungible, and can be integrated with different malware components depending on need. Payloads are more time intensive to repurpose; though still code, like a hammer they were written to achieve some narrow range of effects. This difference helps underline the role exploits play in malware, opening the door for malicious code to propagate to and execute on a target system, such that exploits are not themselves malicious. Achieving root access on a machine does not create an effect by itself; rather it makes a further operation possible, credential theft for example.

This terminology is at odds with some usage, which combines the payload's function into the exploit and uses the combination as a verb, to 'exploit' a system. [3] This approach however limits our ability to categorize and understand malicious code; the two are logically distinct both in sequence and form as exploits are written to a particular vulnerability present in some piece of software while payloads are written to achieve a particular effect. Without one or several exploits, a payload would almost never be able to execute on a target computer - these exploits serve to manipulate the target system into giving malicious code access and user privileges necessary to function. Each of the three components of this framework [2] has a distinct purpose and works in combination with the others to constitute a malicious package. Cyber weapons are a subset of malicious software, where the payload is designed to produce destructive physical or logical effects. Thus not all mil-or malware are cyber weapons as many samples from both categories employ payloads designed for espionage and information theft. There are actually very few examples of destructive payloads, thus cyber weapons, in the history of information security.

Differentiating between sophisticated and unsophisticated samples is not a new area of work in the information security community; static and dynamic analysis techniques have been in use and evolving for several decades. [4, 5] Existing techniques tend to emphasize forensic attributes like matching language setting and compiled time stamps to potential threat actors' locations, selection of targets, and attempts to 'fingerprint' developers based on their development styles. [6, 7, 8] Some of these characteristics are useful but all can be spoofed to one degree or another, potentially complicating attribution. Identifying the source of samples has come

to the foreground more in the past decade as efforts have shifted to understand the nature of threats and their activities beyond the network perimeter. As part of this change, the motivations of attackers and their political status has become a factor of interest. [1, 9] Analytically, the next step is understanding the relationship between the complexity and capability of a piece of malicious code and the nature of its authors. We do this by focusing on the discrete components outlined above, the propagation method, exploits, and payload, an approach which has seen other application. [10]

2.1 MASS Index

The MAlicious Software Sophistication (MASS) Index is a set of functional criteria which could be employed in discussing the high level characteristics of a piece of malicious code - architectural and behavioral features. The contents of the index are descriptive and qualitative in nature - observed features of code that has already been attributed to state or non-state actors. It is not intended as a quantified or machine readable tool; such an effort would, at the moment, involve an awkward translation between categorical and ordinal values and potentially specious attempts to create quantitative specificity where none is currently present.

This index is intended to serve as a collection of characteristics which might be intelligible to business leaders and the policy community in discussing the origins and nature of state-authored code. The first portion of the paper aims to develop, through analysis of these different samples which have been identified as state or non-state, a more consistent shorthand set of features useful to differentiate products of the two sources. The work in this paper considers the non-state samples Lohmys, Upatre, `Tinba`, and `GOZ`as well as the state samples Stuxnet, Red October, Duqu, Duqu 2.0, and Turla.

2.2 Milware

The following analysis, which feeds into the MASS Index, supports the hypothesis that there are systematic differences between state and non-state authored code and proffers the term 'milware' as an alternative descriptor for this former category. The intent of the term is not to draw a firm line in the sand or to suggest that all states write code at equal levels of capability or towards the same end. Milware is intended to describe the more developed engineering effort evidently behind samples like Duqu, Turla, and Stuxnet; their persistence and innovative means of compromise in particular denote a more developed adversary than many non-state examples. Capturing these more developed samples under the term "milware" provides a shorthand for analysts and policymakers to describe this collection of capabilities without relying on vague references to "sophistication".

The design of a piece of malicious software like Duqu demonstrates a more mainstream software development process with a core platform complimented by functional modules capable of being updated based on user need rather than a single bespoke product intended for one purpose. This shift towards an almost operating system like development process builds on the rudimentary multi-component design effort found in tools like the Blackhole exploit kit where efforts are made to integrate propagation methods and exploits into a single package. Our analysis does not consider malicious software for use on the battlefield and does recognize that other states writing code, such as Italy or Pakistan, have yet to deploy tools of this nature. Our purpose in declaiming a category called milware is to identify a simpler shorthand for the espionage, and in one case destructive, tools increasingly found by information security firms and attributed to state intelligence and military organizations. It is important to note that the resource considerations and operational demands of the state apply as well to firms working for the state. Thus while milware may have been developed, in whole or in part, by defense contractors or other private companies, we still categorize the code as state developed.

Almost all of the code considered by this paper to be milware is from Russian or US sources thus the technical characteristics we identify may be a product of the cultural influence or operational priorities of each states. The importance of the claim that there is a gap between state and non-state authored code is not to claim that all state code looks the same, indeed there is substantial variation between the software platforms deployed by North Korea, the United States, and China. Even within states, the priority and assessed vulnerability of a target can contribute to different tactics and malware component use; escalation to employing zero days for example has been a multi-step process observed of some Chinese groups. [11] Recognizing the difference between Duqu and the Wanker worm is not to suggest Duqu is common or indicative of the capability of all states but rather to to highlight significant difference between samples in what is otherwise collectively referred to as malware. The argument for what milware is in this paper is thus not a claim of what may always be present but what is possible amongst edge cases and the attendant implications of this new category.

3. OPEN SOURCE ANALYSIS

This section reviews samples which have open source technical and analytic documentation as well as previous claims of attribution, to discern patterns that may exist in the construction and behavior of state code. In so doing, the paper attempts to expand the array of samples under investigation beyond what might be possible with the direct functional analysis methodology employed later.

3.1 Non-State Code

These samples are developed from analysis conducted by existing information security firms and the larger research community.

3.1.1 Lohmys

Lohmys is malware which targeted Brazilian banks and financial institutions during the World Cup. The code uses phishing campaigns to reach and infect its target as well as a dropper to propagate a malicious executable onto the victim's machine. [12] Lohmys then attempts to gain elevated permissions on the victim's computer and begins scraping web form data for banking information. [13]

Lohmys payload requires elevated permissions on the machine in order to exfiltrate data but is propagated using a set of low severity exploits which has already been disclosed at the time of discovery. [14] The code relies on a dropper to pull down an executable file, making the sample somewhat easier to detect and more uncertain to successfully compromise the target machine. Lohmys shows little emphasis for

specific individuals or information, infecting approximately 700,000 users throughout Brazil during phishing campaigns carrying World Cup themed content during weeks leading up to the event. [15] The content of the campaign demonstrated little specificity other than being written in Portugese, a low cost approach to maximize potential victims if the expected rate of compromise is low.

3.1.2 Upatre

Upatre is an unusual piece of malware with more extensible propagation capabilities than other observed samples. Its rise followed a strong dip in use of the Blackhole Exploit Kit; Upatre spreads through similar propagation techniques, such as malicious attachments, direct downloads, and malware served by compromised web hosts. What makes Upatre unique is a persistence architecture which facilitates the delivery of other malware samples onto compromised machines. This feature creates a tie between Upatre and the other samples explored. By acting as a dedicated dropper, Upatre bridges the traditional gap between propagating a payload using the segmented process of combining an exploit kit with a custom dropper and pushing code directly through a pay per install service.

Upatre's initial propagation method is similar to previous samples, with a combination of file format exploits supporting email phishing campaigns and compromised web hosts delivering the code. [16] Once a toehold is established on the victim's machine, Upatre will drop it's payload on the system and delete the original running copy. [17] Upatre will then sit on the victim's system and listen for commands from the C2 server without, initially, trying to exfiltrate data beyond basic reconnaissance information like the operating system version and hostname. The payload installed functions primarily as a propagation method for follow on code as Upatre is intended to operate as a persistent dropper, allowing non-state actors to dynamically spread their own malware and control infections with more specificity. [17]

3.2 State Code

This set of samples have been identified by information security firms and research organizations as authored by states, based variously on analysis of the code's features, targets, and contextual factors like the likely working time of the developers or their computer language settings. This analysis, as with our work, is subject to speculation and inaccuracy, as none of these factors are intrinsic or immutable but we make an effort to include material from multiple different firms so as to try and offset some of this bias.

3.2.1 Red October

Red October was a sophisticated espionage package discovered by Kaspersky in October of 2012. Likely developed by Russian speakers, Red October targeted research and diplomatic organizations across Central Asia and Eastern Europe with a payload of more than one hundred separate modules intended to collect system information and copy user data such as email files, credentials, and keystrokes. [18]

Propagation - Red October had a three-stage propagation method. Initially spread through spear-phishing emails, later adding both compromised attachments and a malicious URL, all were used to download a second-stage, dropper program. [19] This second propagation stage would pull different payload modules from a set of command and control (C2) servers and store them on the target's local disk or system memory depending on the program's function. [20] Once injected, some of the downloaded modules functioned as a third propagation method to spread over infected machine's local networks.

Exploits - Red October employed four separate exploits. [20] Three were used to enable compromised attachments while a fourth allowed the embedding of a malicious URL in emails. Each of these exploits had been previously discovered and published but remained unpatched on targeted systems, one of the few state sponsored samples with no exploits leveraging zero day vulnerabilities. Two of the Excel exploits used in the first stage of Red October's propagation method were originally developed by Chinese hackers, likely as part of unrelated attacks on Tibetan activists. [20]

Payload - Red October's Payload was stored remotely, waiting for the second propagation stage dropper program to download it from a range of servers. A loader program then decrypted one of more than a hundred different modules, each with a different function. [18] Reconnaissance modules collected information about the infected system, from browser versions to filename and folder trees, to assess the value of data stored locally. Others were responsible for collecting email records, user credentials, password hashes, information about attached mobile devices, and system keystrokes. A set of persistence modules included code designed to write a backdoor into either MS Office or Adobe PDF Reader, allowing Red October's authors to use otherwise innocuous attachments for these programs to reestablish a foothold if kicked off of a target. [20] The variety of possible espionage and reconnaissance capabilities which could be deployed to an infected computer were tremendous.

3.2.2 Stuxnet

Stuxnet was developed by the United States, in conjunction with Israel, to damage Iran's nuclear enrichment facilities at Natanz and generally disrupt a potential weapons development program . [21] Targeting centrifuges being used to separate different isotopes of Uranium in order to obtain pure fissile material, Stuxnet was able to jump into an air-gapped network to infect machines directly controlling the Programmable Logic Controllers (PLCs) directing each centrifuge. [22]

Propagation - Stuxnet included a two-stage propagation method; computers at the centrifuge facility were networked together but likely not connected to the internet. [23] A separate set of Windows machines, configured to communicate directly with the Siemens-built PLCs managing the centrifuges, were air-gapped (given physical separation) from all other machines with no direct link to the internet.[22] The first propagation stage, including two exploits that had not yet been publicly released, manipulated each computer's network services into replicating the code to all connected computers. In all, this first stage propagation had at least five different propagation techniques associated with it, including a peer to peer communications and update protocol, local area network shares, and the Microsoft local print network software. [23]

In order to jump the gap to those Windows machines with access to the centrifuges, Stuxnet had a second propagation method, employing a small software package that lived on removable USB sticks. This program would run, automatically, when connected to a new machine, allowing Stuxnet

to jump between computers even those kept physically separate. [24] The coordinated efforts of the local network and USB stick propagation methods proved highly effective in delivering Stuxnet's payload.

Exploits - Part of the Stuxnet propagation method used an exploit in the Windows Print Spool network service to spread between computers. From an infected machine, Stuxnet would submit a specially formatted print request to another, uninfected, machine. Instead of supplying information for a print job, the request would trigger a remote procedure call, injecting Stuxnet's code from the original infected computer over the print network to the target machine. [23] Here the propagation method was the network-based, remote-procedure call, but the exploit was necessary to take advantage of a vulnerability in the Windows print software. The first stage of Stuxnet's propagation method employed at least two previously unknown exploits to spread its payload through the Microsoft Print Spool, Windows Server Service, and USB sticks.[22] These exploits allowed the propagation method to hijack existing network services and replicate Stuxnet.

Payload - Stuxnet's payload was designed to damage the centrifuge systems at Natanz without alerting facility staff. One module infected the PLCs for centrifuge rotation; running the affected machines up to just over their maximum design speed for 15 minutes then returning to normal for 27 days before slowing down almost to a standstill for 50 minutes. [23] This month-long cycle was written to repeat until centrifuges began to crack from what would appear to be metal fatigue. A second payload module opened and closed feed and exhaust valves controlling the flow of gas to other centrifuges, disrupting the multi-stage enrichment process. Stuxnet also had an obfuscation module that took normal diagnostic information on the centrifuge rotation speed and fed it into the machine's Supervisory Control and Data Acquisition equipment so that nothing would appear to change until the machines had broken down . [24] These two payload modules represented a significant investment in time and resources to write and test the code. [21]

3.2.3 Duqu

Duqu is an espionage platform identified by the Laboratory of Cryptography and System Security (CrSys Lab) at Budapest University in Hungary in September of 2011. In a detailed report, later reinforced by analysis done at Symantec, Duqu was revealed as a small but capable espionage focused malware platform with a highly customized payload employing multiple nested installers for security. [25]

Propagation - Initial infection was using a compromised windows .doc file sent via spear-phishing email. [26] Duqu did not self-replicate, instead infecting new machines based on instructions from the related command & control (C2) servers. This human in the loop propagation process occurred over network shares rather than an internet based technique.

Exploits - Duqu leveraged a vulnerability in the Windows TrueType font parsing engine (CVE-2011-3402). This allowed the initial dropper to gain access to the targeted machine. This exploit was unknown at the time of use and appears to have been the only one employed by the sample.

Payload - Duqu contained modules for communication, reconnaissance, and a keylogger. Communications took place primarily with other infected machines over a peer-to-peer protocol using both HTTP and IPC (Inter Process Communication) over SMB (Server Message Block). [26] This allowed for communication to a set of C2 servers from a more limited number of computers, thereby minimizing the number of machines emanating unusual network traffic. A custom protocol involved uploading .jpg files with appended encrypted data to exfiltrate information and receive commands. This allowed other functional modules to be downloaded into existing Duqu infections according to the attacker's needs. Duqu used a valid certificate from C-Media Electronics, a Taiwanese firm, to sign its code. [27]

3.2.4 Duqu 2.0

Duqu 2.0 is part of an espionage campaign targeting Kaspersky Labs and hotels and conference facilities surrounding negotiations behind the 2015 Iran/P5+1 nuclear non-proliferation agreement. The software is closely related to the Duqu platform; analysis from Kaspersky as well as the CrySys Lab in Hungary suggest that while the payload modules had changed, the underlying architecture of the sample for propagation was substantially similar. [28] Demonstrating many of the same behaviors, Duqu 2.0 allowed for a more expansive set of espionage capabilities and included a new certificate.

Propagation - Duqu 2.0 propagated through targeted spearphishing emails, in one case penetrating the security firm Kaspersky through a small affiliate office. [29]. The attack then moved laterally through the firm's network using Microsoft Windows Installer Packages to spread to and infect new machines. The firm also noted that, "modules were also observed performing a 'pass the hash' attack inside the local network, effectively giving the attackers many different ways to do lateral movement". [29]

Exploits - The sample used an exploit similar to the original Duqu code, based on a vulnerability in Windows True Type fonts. [29]. This exploit was used to gain access to the Kernel on affected machines, representing a serious escalation of privilege event in the native environment.

Payload - Payload modules for Duqu 2.0 included a means to initiate, freeze, and bypass a host of intrusion-detection and anti-virus applications as well as a network reconnaissance tool and more than 100 configurable modules available for download as a single package. These modules include the capability to gather credentials, network metadata, and to read/write files on the affected machine. [29]

3.2.5 Turla

Turla, also known as Snake or Uroburos, is a sample that has been in the wild since early 2008 and is of suspected Russian origin. [30, 31] For much of its life, Turla was found exclusively on Windows machines however it recently made a jump to Linux as well. While the abilities of Turla may be observable in other, non-state authored, samples, the severity of its exploits and use of the code are what set it apart. While Turla was used to compromise several servers over the years to gather information, [32] it also established a presence in Department of Defense networks. The fact that this sample was able to penetrate DoD and perform a complex intelligence gathering campaign is evidence of a capable adversary. Beyond the DoD compromise, in early 2014, samples of Turla were found to be on Ukrainian devices performing the same intelligence gathering. [31, 33]

Propagation - Turla employs a combination of targeted

spearphishing and watering hole attacks to compromise its targets. [34] The emails include infected .pdf attachments with language specific details about conferences and other events while the websites are specific to the region and interest issue of the target.

Exploits - The exploits in use by Turla include several known Java, Flash, and Internet Explorer vulnerabilities for web compromise as well as two previously unknown. The first, targeting Adobe, enables arbitrary code to support the propagation method. The second, a privilege escalation technique against Windows XP and 2003, supports the payload. [34]

Payload - Turla's payload, cd00r, allows the new strain to listen on the host for 'magic packets,' without the need for elevated permissions, that allow will establish a connection to the C2 server and begin executing the issued commands via '/bin/sh -c.'. [32] Turla employs this backdoor to maintain persistence on host's machines. While running on the host without elevated permissions does restrict the activity of Turla, it does allow for remote intelligence gathering of the target by simply sniffing internet traffic, exfiltrating the host's files, and other seemingly benign activities. [35]

4. FUNCTIONAL ANALYSIS

In this section, we identify several additional samples and conduct a more detailed analysis in order to evaluate the trends and commonalities identified above with direct inspection of code from both state and non-state examples.

4.1 Methods

To gain a fuller understanding of our samples we reverse engineered each in a controlled lab environment using both static and dynamic analysis. This process leveraged existing and widely used tools, including IDA Pro [36], BOCHS [37], WinDBG [38], Immunity [39], and SysAnalyzer [40], in a live running environment. These tools helped document the changes our samples made to the testing system and parsed each of the binaries, enabling us to work with readable machine code. This analysis also considered how samples interacted with their surrounding network so we included TCPDUMP [41] and WireShark [42] to gather and analyze outbound traffic.

4.2 Data Collection

We selected a set of state and non-state samples for both accessibility and their chronological proximity. Our ability to work directly with the state authored samples was limited by non-disclosure agreement with a commercial firm and so relies in part on open source documentation. Code selection was driven by three key factors: scope of impact, availability of samples, and knowledge of lineage. Using non-state samples with a long history of use allowed us to consider crossover between each and select more complete instances of code.

To achieve the clearest possible distinction between state and non-state authored code, we selected two samples from each category. Non-state samples included Game Over Zeus (`GOZ`) and Tinybanker (`Tinba`) while `Sandworm` and the code used in the compromise of Sony Entertainment's networks were selected as our state samples. Each sample has had world wide impact of one type or another and propagates using a different method.

4.3 Non-State Code

`GOZ` is piece a of malware that, when installed on a target system, will establish a P2P connection based on the Kademlia protocol. [43] The binary propagates using phishing emails containing links that redirect to compromised web hosts. At these links are servers using the Blackhole Exploit Kit to infect target machines via vulnerabilities in the victim's web browser. A dropper module then deposits a piece of code on the target system to execute the `GOZ` payload which attempts to contact a C2 server. [44]

The use of an exploit kit is not novel and indeed is present in the second malware sample analyzed in this paper. `Tinba` is a banking trojan that has become notorious for its incredibly small size, clocking in at only 20kb. `Tinba`'s propagation method uses a set of compromised websites to distribute code to victim systems, either via linking to a web host in a phishing email or by targeting certain user's browsing habits; the group behind `Tinba` is notorious for compromising pornographic web sites and indeed this now serves as the main mode of `Tinba` propagation. [45] Once a victim visits the compromised web host, `Tinba` uses the Blackhole Exploit Kit to enable installation of a dropper on the victim's system. Lastly, this dropper downloads and installs the `Tinba` payload from a predetermined list of URLs. `Tinba`, unlike `GOZ`, does not emphasize botnet functionality in its payload, instead it focuses on skimming victim's banking information through the use of web-injects and function hooks. [46]

Both samples of malware differ in the type of data targeted and the manner of exfiltration but each makes use of the Blackhole Exploit Kit to successfully propagate and execute their respective payloads. This fact demonstrates some of the limitations of developing code for malicious purposes - designing several payloads may well be less costly in terms of information and human capital than keeping abreast of the latest vulnerabilities present on a particular target systems and continually developing new exploits to match. This encourages malware's relatively broader propagation - the proportional rate of success for all attacks is lower than for more targeted efforts but the victim pool is made substantially broader to compensate. Each group distributing malware, while they might find it cost-effective to develop an original payload, appear willing to use the same vulnerabilities found in many other samples of malware to gain code execution on the target system.

This breadth of propagation is further reinforced by the selection of a kit like Blackhole, which targets users with a variety of potential exploits in any given iteration. Blackhole targets victim interactions with the web browser; by loading an iframe containing malicious JavaScript into the victim's browser, Blackhole can detect which exploits the victim is vulnerable to and craft an environment wherein the victim will be exposed to a condition that triggers the exploit and executes the associated payload. [47] While not a core sample, the Blackhole Exploit Kit is taken into consideration in the analysis due to its significance in `GOZ` and `Tinba`and prevalence in other malware as a means to support propagation.

4.4 State Code

The `Sandworm` group, a Russian linked state sponsored or supported hacking organization, has been running a multinational intelligence gathering campaign on Western states

from as early as 2009. [48] The propagation method typically observed with Sandworm attacks involves spear-phishing email to the target with an attached Powerpoint document that, when opened, will trigger CVE-2014-4114 to compromise the target system. [49]. The Sandworm attacks targeted largely NATO states and affiliated countries including the United States, United Kingdom, and the Ukraine; the breadth of this spread allows us to to analyze the propagation patterns behind single state sample exploiting a single CVE.

The selection of Sandworm in based off the observed similarities to GOZ was botnet functionality and provided a basis to contrast the two with regard to their communication protocol. [44, 48] GOZ's use of the Blackhole Exploit Kit demonstrates a common feature of malware; reliance on pre-existing third-party exploit packages. This is directly at odds with Sandworm's use of a more particularly selected and applied exploit to gain access to target systems. While GOZ and Sandworm do share similarities in functionality, one the quantifiable difference between the two samples is the severity of the exploit(s) and payload(s) used. A CVSS score was used to determine the observed severity of the samples analyzed, Sandworm's CVSS score is 9.3 [50] while GOZ's average CVSS score comes to 6.3. [51, 52] This distinction in CVSS score between samples shows some evidence of the difference in severity. [53]

The second state sample is from the compromise of Sony Entertainment's networks in 2014. While not a coherent, multi-target attack like Sandworm , this particular chain of events provides insight into the functional process of an attack by state actors. The state samples exhibit particular characteristics in their propagation methods, exploit selection, and payload behavior that help delineate them from more conventional malware binaries.

4.5 Sample Analysis

Our methods of analysis focused on differences and similarities among the gathered samples. In addition to traditional reverse engineering techniques, we also chose to study how the samples acted in a network environment similar to those found in corporate and personal targets so that we could observe how non-state samples propagated to, and within, a target differently from state authored code. Comparing Sandworm's use of CVE-2014-4114 with Tinba and GOZ's use of the Blackhole Exploit Kit, we were able to identify key differences between their respective propagation methods.

Starting with GOZ and Tinba was, we used IDA Pro to examine what the code looked like, how it was packed, if it was obfuscated, and what techniques were used to compromise and gain access to the target system. At a higher level, to measure differences such as impact on the system and internal network propagation, we simulated a network of computers and observed how the samples moved through it. The paper's goal is to create a more robust series of identifiers for which to classify malicious samples. To do this we targeted core similarities shared between each milware and malware sample, then classified the strongest found similarities within each category.

To follow the life cycle of an attack, we started with the propagation of code to the target. While GOZ alone used the Cutwail botnet to spread, both GOZ and Tinba used emails containing links to malicious web servers and compromised web hosts serving malware. [44, 45] The emails contained lures to bait the victim to click through to these sites, crafted email templates emulating unpaid invoices, negative account balances and social media advertisements containing logos and identifiers that belonged to legitimate companies. These phishing emails did not include personally identifying information however. To verify this, we set up a honey-pot to try and attract emails from these spammers; what we received was poorly customized and did, in fact, lead to known malicious web hosts serving Blackhole Exploits. [54] Below are some of the URL patterns discovered:

```
[redacted].php?pBzmU=ePRGAAKDWk
  &CMSgsDyzkuFvs=JnhjMIPLmQY
200 OK (application/java-archive)

[redacted].php?DgdAXYmfoKifsN=sNZsfdfdRLslud
  &xpcuSaClYajZ=bsczZZysmLE
200 OK (application/java-archive)
```

GOZ and Tinba both make heavy use of the Blackhole Exploit Kit as a means of facilitating code execution on victim computers. This fact was significant in the analysis; unlike milware, non-state authored code relies largely on third-party exploit kits, following from the intent to infect as many victims as possible. Some of Blackhole's releases make use of 15 to 24 existing CVEs with CVSS scores ranging between 6 and 9.5; however, none of Blackhole's exploits are built around zero day vulnerabilities and most have been patched by the manufacturer. Subsequently, this indicates that those who are using Blackhole do not necessarily need to guarantee code execution on a high percentage of their targets, but instead aim to propagate to as many victims as possible.

Moving from propagation to payload, the paper found that while the purpose of each sample was different, Tinba being a banking trojan and GOZ the payload to set up a botnet, there were key similarities between the two. Most important of these were the two sample's preference for targeting browsers, with the ability to siphon account information and dynamically inject web content. This functionality is seen on a number of malware samples in the wild and requires very little specific information about the victim prior to attack. [55, 56, 57]. In a limited extension of this pilot study, digging deeper into the payload's code base, we found that even when looking at a new strain of these samples, there were a striking number of similarities. [57]

In the more detailed code analysis of each of the milware samples, this paper considered the findings from the malware binaries and used the PrEP framework to draw parallels between the two sets in order to identify key differences in propagation, exploitation and payload functionality. [2] While analysis of the milware samples was limited by relatively greater reliance open source documentation than with the malware samples, this paper was able to utilize an arms-length relationship with an information security vendor which included some analytic support with the source of our samples. Although not in possession of the binaries, we were able to obtain the .text, .code, .data and .bss sections, where applicable, as well as the imports and exports of the binaries to statically analyze each in addition to gathered emails and official documentation.

Working with Sandworm, the paper's analysis looked to the propagation method attackers chose to distribute their

Sample	Propagation	Exploitation Method	Payload	CVSS Score*
`Sandworm`	Tailored Spear Phishing	CVE-2014-4114	BlackEnergy	9.3
Sony Attacks	Tailored Spear Phishing & Physical Access	SMB Worm Tool	Listening Implant Lightweight Backdoor Proxy Tool Destructive Hard Drive Tool Network Propagation Wiper	N/A
Game Over Zeus	Email Spam & Compromised Web Hosts	Blackhole Exploit Kit	`GOZ` Server	5.0
Tiny Banker	Email Spam & Generic Spear Phishing	Blackhole Exploit Kit	`Tinba` Server	6.3

Table 1: Overview of Examined Samples

*Average of observed exploit CVSS

code. Examining multiple email samples sent to victims, there was a generally high level of customization in the prose which specifically targeted their victims including a compromised Powerpoint slide deck. [58, 48] While popular security practice tells users to never download attachments from untrusted emails, the work that the `Sandworm` Group put into crafting legitimate looking messages provided a convincing, tailored, basis for the recipients to trust the content.

Looking at the Sony compromise, this paper took a different approach to analyzing the propagation of the attack due to the fact that the initial compromise was most likely due to an insider and not through an email campaign as seen in our previous samples. [59] The core analysis however was the same, still focused on the degree of customization present in the code. Looking at how the milware propagated through Sony's network, it's likely that, due to the tool set used [60], the attackers attempted to breach higher value targets in order to further cement their presence in the network. This process involved targeting network administrators' computers and using the stolen elevated credentials to breach the Active Directory and Domain Controller servers to gain access to credentials used network-wide.

Finally, we looked at the design and functionality of each sample's payload and supporting exploits. In both cases, the actual functionality of the payloads were tailored to a particular purpose. In the case of `Sandworm`, the attackers used a version of the BlackEnergy remote administration tool that supported the use of delete, ldplg, unlplg, update, dexec, exec, updcfg. These commands allowed the attacker to gain and maintain persistent access to the victim's machine through its backdoor functionality as well as facilitating the execution of any new payload sent to the system. [61] The payload deployed on victim machines in Sony's networks contained specific functionality, designed for the purpose of exfiltrating corporate data, further propagation of the payload through the network, dynamic access to compromised nodes, and the destruction of victim hard drives. [60]

The exploits supporting each of these payloads were tailored to their function. In the case of `Sandworm`, an Office exploit (targeting CVE-2014-4114) was used via an email attachment to give the attacker time to create a backdoor in the system to enable further code execution. [49] Within Sony, the SMB Worm Tool was used to propagate itself through the network after the initial victim was compromised. [60] This allowed the attackers to traverse network undetected due to its zero day classification. [60]

5. FINDINGS

The sum of this open-source and functional analysis are a set of criteria for differentiating milware from malware, the utility for which extends beyond the information security community to policymakers. This paper proposes the MAlicious Software Sophistication or MASS Indexas a basic tool to help identify the authorship of malicious code samples. The MASS Indexconsists of four main categories by which to classify malicious software: method of propagation to the victim, movement with the target network, exploit severity, and degree of payload customization.

5.1 Propagation Method

Propagation methods can be classified according to their scale and specificity. Scale determines the total possible target pool i.e. how many computers and devices in the world are accessible. Code which propagates over the internet is likely to be found much farther afield than that spread over compromised storage media. The scale of a compromised web site would be tremendous if the site in question is Google or tiny if a research page on an academic network, visible to users in that institution only. Conventional botnets may have tens of millions of slave machines and present an excellent method of propagating to targets indiscriminately [62]. A propagation method which targets all internet connected computers (large scale) is thus different from one which targets only users connected to a particular local area network (small scale). `GOZ`'s server contains tools such as DGAs, USB, NFS, Samba spreading tools, designed to propagate as widely as possible.

Specificity measures the targeting constraints placed on malicious code, determining how much of the possible target pool is of interest or 'active'. These could be technical limitations, focusing on a particular operating system or software version, based on personal information like account credentials or details about co-workers, or the presence of certain file names on the victim's machine. Specificity can

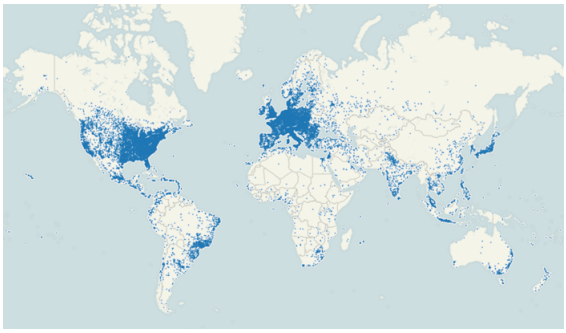

Figure 1: Geographic distribution of nodes in the ZeuS P2P network by Bot ID. (Source: Dell SecureWorks)

help to contain the spread of malware infections, lowering the likelihood of detection and limiting defensive response.

In propagating to the victim, milware tends towards medium to small scale and highly specific propagation methods while malware employs large scale methods with little to no specificity. Propagation scale can be relatively easily established by looking at the format code is spread in while specificity is well established by the degree of customization in the delivery vector. This can range from the use of the Targeted Threat Index [1] with regard to email propagation, to examining how much prior access or knowledge the attacker had about the target. In the case of `GOZ` infections, we see a tendency for `GOZ` to spread to any other computer where code execution can be achieved.

The first major indicator across samples was the method by which code propagated to the victims. Referring to the Targeted Thread Index, emails containing malware had a Targeting Score from 1 to 3 while those propagating milware had a Targeting Score from 4 to 5. [1, 63] The method used to expose victims to malicious code also varied, for example `GOZ` was very large scale and employed something akin to the 'shotgun technique', prioritizing spread of malicious emails to as many victims as possible. Milware incidents such as `Sandworm` used similar scale methods but with far more restrictive specificity, pushing code to a highly constrained group of targets. [58]

Propagation within a network, after the attacker has compromised the target, demonstrates further variation between mil- and malware samples. Milware demonstrated a trend towards attacking higher value targets before further propagating to lower value targets, cementing its position in the network. This stands in contrast to malware, which tended to spread to as many hosts as possible and typically moved to topographically proximate peers instead of high value nodes within the network. Milware's lateral movement evinced a human in the loop, prioritizing users with valuable credentials and network permissions. As demonstrated by the Sony compromise, careful propagation within the entertainment company's network infrastructure was key to the attacker's success. The presence of the Network Propagation Wiper in the attacker's toolkit indicates and interest and investment in covert navigation of Sony's network. [60] The selected malware samples by contrast, propagated without obvious human input or regard for the value of individual nodes, proceeding in a more geometric fashion determined by the initial point of infection.

5.2 Exploit

The severity of the exploits used to compromise the targets serves as another major indicator in the MASS Index. Reuse of exploits or the use of exploits with a low CVSS generally indicated malware while use of higher scored and chained exploits was associated with milware. [53] Because `GOZ` relies so heavily on third party exploit kits, the CVSS score, while generally critical, is predictable across the board. Consequently, the wider exposure of these exploits via Blackhole will substantially increase their likelihood of discovery and mitigation, leading to their use as signatures in many IDS/IPS solutions.

Milware samples employ exploits which do not exhibit the same bias towards web-facing software as malware and there are generally higher CVSS scores across the board; `Sandworm`had an over all rating of 9.3 with a propagation subscore of 10 and exploitability subscore of 8.6 [64].

Both of the malware samples considered in functional analysis relied on the Blackhole Exploit Kit to support propagation and gain access to target systems. While these different exploit kit versions did contain exploits with a relatively high CVSS scores, many had already been patched. The milware samples used exploits with similar or higher CVSS ratings and included vulnerabilities which had generally not yet been disclosed. This reliance on smaller numbers of high quality exploits demonstrates milware's target focus, which necessitates substantially higher probabilities of successful infection. This stands in opposition to malware which is sensitive less to the particular target than the tool which is accessible and so employs a wide array of exploits in order to develop a financially viable hit rate of compromised machines without investing in particular vulnerabilities.

5.3 Payload

Examining the tool set and functionality of the payload in question, the level of capability and customization in milware payloads to be higher on average. This involves the payload in question using custom developed tools for post-compromise activity on particular machines rather than equivalent functionality broadly available in the marketplace i.e. a payload for a web server, a payload for a desktop, and a payload for a Domain Controller. `Sandworm`, employs a very small set of tools intended to guarantee persistence and data exfiltration. With regard to the Sony attacks, Table 1 shows that while the payload did contain some generic tools, they were all tweaked to the constraints of the firm's network and limited their functionality to that network. By contrast, `GOZ`'s payload was identical regardless of the target in question.

Component	Behavior
Propagation	Highly specific propagation method Lateral movement within target
Severity of Exploit	Target specific exploit
Customization of Payload	Customized Payloads

Table 2: Core Set of Differentiating Features between Malware and Milware

5.4 The MASS Index

These trends suggest a set of criteria to distinguish state and non-state authored code which we build into the MAlicious Software Sophistication (MASS) Index. Using commonalities between state samples, we were able to generate a set of criteria which differentiate milware from malware. Table 1 provides an overview of our findings with regards to the core set of features present in each sample while Table 2 identifies the divergence identified between malware and milware.

Using this tool, we can begin to consistently distinguish between mal- and milware. These differences refer to code sourced from the United States, Russia, and possibly Israel, so do not represent the complete spectrum of state capabilities but rather, are concentrated on the most advanced edge of what has been observed so far.

5.5 Hacking Team and Private Intermediaries

The compromise of Hacking Team and subsequent full disclosure of their internal emails, business documentation, and product code in the summer of 2015 provided an interesting tension between principle and research value. The release of emails between parties with an entirely reasonable and complete expectation of privacy poses a quandary for researchers looking to use the material as empirical resource. While there is value in the material contained and using the Hacking Team documents can be weighed directly against the virtue of attempting to protect their secrecy, the greater problem is one of principle. Hacking Team's activities are those of a business but the firm's clientele includes repressive regimes whose consideration of political expression and individual liberty stand in stark contrast to the expressive ideals of free society and the academy (at least in its nominal form).

It is not enough to say however that Hacking Team's choice of customers, reprehensible though it may be, was morally depraved enough to warrant voiding each individual employee's privacy interests. To do so would be to invite a cavalcade of consequences for researchers of all stripes from any leaked information now and in future. Rather than invite the creation of such unsound precedent, this paper chooses to use only that information which is of a business, rather than personal, nature on the rationale that the firm as an entity has a far less developed expectation of privacy than its employees. The information obtained and reused in this analyses then is limited to that which centers on, stems from, or directly supports Hacking Team's commercial enterprise as a developer and distributor of malicious software.

On the nature of Hacking Team's products - there is a distinct similarity with the characteristics of the Galileo Remote Control System offering and the descriptors of milware described in this paper. Particularly, the limited use of zero days for propagation to their targets and evidence of some customization between different customers. There is a distributed payload function, with different modules for network reconnaissance and more overt collection activities like keylogging and activation of audio and visual peripherals. [65]

The propagation patterns by contrast are different from both mal- and milware samples investigated, focusing on single user's devices and seemingly paying less attention to their attached networks. While there appears to be some human in the loop of the propagation of Hacking Team's products depending on the client, targets appear to be infected directly rather than via lateral movement through a network. This is indicative of the environment in which Hacking Team is operating - against individuals and small organizations with lower security posture than might be targeted by states.

There also appears to be less effort placed into targeting spearphishing emails from Hacking Team's clients when compared to those associated with several milware samples. Hacking Team's products, and in all likelihood some others offered by private intermediary firms like Gamma Group and BlueCoat, sit in between the code associated with non-state groups like carders and those offering pay-per-install services and the leading edge tech from advanced states like the US, Israel, or Russia.

6. POLICY IMPLICATIONS

Pervasive development and use of milware constitutes not only a direct technical challenge of decomposing and analyzing well obfuscated code but also threatens a set of key assumptions underpinning the current information security research and defense paradigm. States operate in a different legal regime than criminal groups and individuals, inverting the power relationship between attacker and defender and altering what is possible in the defense against and prosecution of sources of information assurance threats. This paper develops the MASS index as a rudimentary tool to distinguish between state and non-state authored code. Given the existence of such a distinction, the paper highlights several potentially significant policy implications:

6.1 Mitigating Public Disclosure

The material impact of public disclosure by information security firms on state's espionage and related operations appears to be minimal. Operations may change tactics but absent the aggregated total of incidents seen between the US and China, states appear to have little to fear from disclosure of their activities and so the traditional paradigm of revealing tactics and techniques to dissuade attackers and aid defenders is less effective. The commercial and academic information security community demonstrates a preference

for a "sunlight is the best disinfectant" model when it comes to threat actors and their code. Reports on the activities of groups from the US to Lebanon to China abound, identifying these "APTs" and their activities under various entertaining monikers ala Putter Panda. [66] While not an industry standard behavior with all attackers for competitive reasons, information about particular techniques, tactics, and exploits has become both an information sharing device and marketing tool for information security firms.

One underlying expectation is that such disclosure will dissuade attackers and aid defenders. While the publication and circulation of information about milware is useful as an academic research tool, several years of slick reports, including a trove of analysis on Chinese [67] and Russian [20] activities and revelations by Symantec [68] and Kaspersky about several alleged US samples, including one espionage framework in place for almost 14 years [69], seems to have done little to dissuade milware development or deployment in any permanent fashion.

This lack of impact is in large part due to the fact that milware reverses the traditional hierarchy of information security, where defenders have the onus of legitimacy and hackers are operating outside of the law - their existence a product of the confluence of fundamental insecurity in most commercial software and opportunities for financial gain. States, to a very rough extent, are the law and have little fear of material harm from the pubic effects of disclosure about their activities.

There may well be a higher threshold for response whereby if states are found engaging in more destructive activities or targeting highly vulnerable groups, revealing these activities may yield a response from the targeted state. Impact on public opinion of these disclosures is also unclear, as yet not well studied; the shift in popular sentiment from publicizing milware's impacts and targets could bring about the rancor and outcry necessary to force government action in response. Aside from these hypotheticals however, public disclosure of state's activities along the lines of those revelations used to discuss non-state groups appears to have had minimal material impact.

6.2 Milware Proliferating to Malware

States have far more resources to develop new techniques and exploits than non-state actors. While this disparity has long been focused on the threat of states developing destructive payloads, an acknowledged resource and expertise intensive endeavor [70], one more immediate threat is that the propagation techniques and exploits developed for milware applications will trickle down to malware authors. [71] The chief threat of malware then is not the prospect of readily available destructive payloads, but that states might inadvertently fund a massive research and development apparatus for non-state groups, further intensifying the disparity in capabilities between attacker and defender.

The exact reuptake rate of code from platforms like Duqu, Turla, and Red October into malware is unclear but components of all three have been reused by criminal groups at one time or another. More work remains to be done to understand the decision making process of non-state actors who choose to reuse code in the wild, buy new components, or build their own. Even a rudimentary game-theoretic model considering the information available to actors and the resources required of each for these three choices weighed against the potential benefit to be obtained would be useful. It is not a novel idea to suggest that there are a small set of sophisticated threat actors in the information security space whose code and tactics may leak into the actions of others but recognizing the source of these innovations as states highlights a key problem. State's financial and human resources are substantially greater than any non-state organizations - meaning the output of innovation, in terms of both variety and volume, is sufficiently great so as to constitute a fundamentally different phenomenon in the perverse 'trickle down effect' from mil- to malware.

6.3 Growing the Malware Market

State's financial resources may price defenders out of the market for exploits and even bring new sellers into play. Even where state resources are not used directly to develop new code, the presence of a market like mechanism for groups to buy, sell, and trade components of malicious software has been established. [72, 73, 74] While estimates of the prices and popularity of different tools is subject to some debate, the resources of state actors will impact these markets. The rise of milware may be pricing software vendors, and other defensive organizations operating through bug bounty programs, out of the market. More insidiously, the presence of states with financial resources to burn and an appetite for the latest vulnerabilities in widely used commercial software may well encourage substantial growth in the number and talent of individuals who participate in this market as sellers. As the prospect for financial gain increases, more and more people join to sell their malicious wares to the highest bidder. Milware then offers the prospect of becoming a driving force in the sophistication and variety of malicious software components, especially vulnerabilities, available on the web. For states, this might already be encouraging an arms race to compete for the most effective espionage tools and weapons. For non-state actors, it may make milware-like capabilities available to terrorist groups or criminal organizations.

A more exact characterization of this change in supply based on state demands is deferred until further work can be done to establish workable answers to at least two key questions about the malicious software market. First, how dense are vulnerabilities in software as a whole and are they relatively more or less so in the types which attract most attention from states e.g. operating systems? Second, while it is clear there are several general groupings of actors in the market, it remains unspecified under what conditions and in what quality information flows between these groups. Evidence from the Hacking Team document release indicates that vulnerability brokers and developers are often unsure how to establish a market price for their goods, leading to inefficient outcomes and failed transactions. This segmentation of the market impacts the potential answer to questions about the role of state purchasing. Understanding the relative scarcity of vulnerabilities as well as information flows in the malware market may also allow evaluation of proposals like Dan Geer's at BlackHat 2014 that the US should propose to "corner the market" on vulnerabilities by purchasing and disclosing them all.

6.4 Harming Software Security

The regulatory apparatus in place in many states, especially the U.S., privileges standards for the defense of networks and information systems rather than holding liable the

manufacturers of software and hardware in place on these networks. States have taken advantage of this emphasis on defensive and information assurance standards over software developer liability to develop, stockpile, and deploy exploits for common commercial software. [75] These vulnerabilities may be in firm's supply chains as well as software, providing more permanent basis for compromise. The key distinction here is that states have the financial resources to select high risk but high benefit means to introduce vulnerabilities into target systems.

This emphasis on vulnerabilities translates into milware prioritizing the acquisition and maintenance of access to targeted systems for long periods of time over deploying particular effects at frequent intervals. The continued vulnerability of most major software families presents a less cost intensive and more obscured operational pattern to enable less frequent but more substantial intrusions rather than engaging in small, regular attacks.

States are daring software vendors to build better software with the expectation that they can continue to beat information security vendors and existing security practices at the network and system level. Malware, distributed by individuals and non-state actors, prioritizes effects over access - the particular machine compromised by an infection is less consequential than the successful execution of code to bring about the manipulation or data exfiltration desired. This is because most malware targets certain resource types within vast networks, banking credentials or PII, rather than the data tied to an individual. Milware, by contrast, is concerned with access to more narrowly tailored targets and particular pieces of information.

6.5 Working Above the Law

Existing legal tools presume the target of law enforcement activities are non-state actors but states are operating under this same regime, allowing them to act with relative impunity. States are largely immune from the existing array of legal tools used to locate and prosecute malware authors and distributors as these resources presume targets that can be subject to a state's jurisdiction. A cooperative, hierarchical model exists in the infrequent collaboration between national law enforcement agencies tasked with cybercrime.

Limiting the use of milware is an inter-state monitoring and enforcement task more akin to conventional arms sales or export control restrictions. Non-state actors can be pursued and prosecuted but states and their milware will largely be subject to the state's own willingness to self-restrain or the ability for other states to compel the same. This constitutes a parallel enforcement and mitigation problem for all parties as malware tends to be large scale and much is indiscriminate. Milware by contrast is focused on small target sets and is distributed by actors who are substantively different in terms of motivation, resources, and dependence on other entities.

7. LIMITATIONS AND FUTURE WORK

This paper represents a pilot effort to identify distinctions between malware and milware, thereby providing a basis for discussion on the implications such a separate category might have for the policy community. The data involved is limited but sufficient for an initial probe of the plausibility of such a distinction. In an effort to broaden the application of this paper's findings, the samples selected were wide spread and/or well documented and had accompanying technical and incident response documentation. The assistance of an anonymous information security organization further aided in the functional analysis of the milware samples for part of our analysis.

The term milware obfuscates other distinctions between state developed code, especially between the organizational structure and culture responsible for deploying code for national strategic effect, tactical battlefield use, and espionage. While these three sub-genres of milware each pursues a distinct end, there is analytic utility obtained from mapping their broad similarities and differences against malicious code developed by non-state actors. The paper's focus is on the most capable end of code developed by intelligence and cyber-conflict organizations within states, like CYBERCOM and GCHQ. This is a sub-set of the larger population of state code but also one which best represents malicious software farthest in advance of that employed within the criminal community. This is a pilot project and so the technical indicators we've identified in the MASS index could be made more robust with access to a more substantial data set of both mal- and milware samples as well as some of the next steps identified below.

Taking a large-N approach, this project could consider a larger number of milware and malware samples. This would allow for more rigorous empirical scholarship, integrating a wider array of target types and styles of code authorship. An alternative involves mapping code lineage, seeking to understanding what distinguishes the evolution of state and non-state authored malware over time. By collecting as many directly related samples as possible of 3-5 variants, both state and non-state authored we can use these lineage maps to mark changes in exploit and feature selection and consider how these changes interact with the previously established MASS Indexcriteria. A third potential extension would attempt to trace payload proliferation from state to non-state code, developing a measure of the time it takes for state techniques to diffuse into the general criminal market. Such a measure could aid in the identification of a proliferation life-cycle and thus provide some evidence of authenticity when new attack methods are discovered. Using this large-N approach could help provide insight on the not only the rates of evolution for state and non-state groups but also the difference in capability between them. This capability delta is likely to shift over time and it would be valuable to understand as each group's rate of change itself evolves in response to things like defensive innovations and the availability of malware components.

There are also other factors which bear consideration for inclusion in the MASS Index. The design considerations made in malicious software to limit collateral infection, i.e. compromising and potentially executing on machines other than the intended target, may provide evidence of the legal oversight and propagation specificity associated with states. Where the limitations appear to provide some operational disadvantage, their presence is even more compelling evidence of state activity as few non-state groups are likely to have the capacity to intentionally ignore potentially lucrative victims. Another modification for the Index would categorize exploits by function and the software they're targeting, to discern patterns in the selection of different vulnerabilities between state and non-state groups. The Index itself is a moving target and will necessarily evolve over time

as the state of the art for mil- and malware changes. Building a sophistication metric of this sort remains a valuable exercise both for the potential analytic output and associated discussion with its production.

8. CONCLUSION

Malicious software has long been used to describe a range of threats. From the trope of basement bound teen-aged hackers clutching Mountain Dew to the much marketed, Advanced Persistent Threat, broad use, even overuse, of the term malware has impacted researcher's ability to specify the range and variety of threats in the information security space. The idea of state authored code, milware, as a separate category highlights a set of challenges to the existing legal architecture and security research paradigm which is fundamentally oriented to combat individuals and organizations. This paper proposes the MASS Index and suggests some policy implications which arise from the existence of a separate category of code authored and deployed by states.

Through the MASS Index, this paper has described a rudimentary means to differentiate between state and non-state authored malicious software. Highlighting the implications of milware as a new category, we find several conventional assumptions in place for information security which should be subject to careful review. Milware constitutes a separate category of malicious software whose priorities, as a tool of state influence, and sophistication are different from code employed by non-state groups. By failing to make a distinction between milware and malicious code written by non-state actors, the information security community risks conflating the capabilities and intentions of criminal groups with those of states, degrading the ability to successfully adapt and respond to either. It also denies the policy community a tool to effectively map and understand this complicated array of threats. Hopefully discussions engendered by this paper are the first step towards a more developed conceptual framework.

9. ACKNOWLEDGMENTS

Thank you to Allan Friedman, Lance Hoffman, Costis Toregas, Herb Lin, Drew Herrick, Ian Wallace, Rob Morgus, and Tim Maurer for the advice and insight into the project. Thank you also to the anonymous reviewers of NSPW 2015 and USENIX 2015 for their feedback on drafts leading up to this one. The authors wish to thank George Washington University for support through the Cyber Security Policy and Research Institute and the Columbian College Political Science department.

10. REFERENCES

[1] Seth Hardy, Masashi Crete-Nishihata, Katharine Kleemola, and Adam Senft. Targeted threat index: Characterizing and quantifying politically-motivated targeted malware. In *This paper is included in the Proceedings of the 23rd USENIX Security Symposium.*, pages 527–541, August 2014.

[2] Trey Herr. PrEP: A framework for malware & cyber weapons. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2343798, February 2014.

[3] P Bright. Massive sql injection attack making the rounds-694k urls so far. http://arstechnica.com/security/2011/03/massive-sql-\\injection-attack-making-the-rounds694k-urls-so-far/, March 2010.

[4] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. Malware analysis and classification: A survey. http://www.scirp.org/journal/PaperDownload.aspx?paperID=44440, May 2014.

[5] U Bayer, A Moser, C Kruegel, and E Kirda. Dynamic analysis of malicious code. http://dx.doi.org/10.1007/s11416-006-0012-2, August 2006.

[6] I You and K Yim. Malware obfuscation techniques: A brief survey. http://dx.doi.org/10.1109/BWCCA.2010.85, November 2010.

[7] A Moser, C Kruegel, and E Kirda. Limits of static analysis for malware detection, 2007.

[8] M Schultz, E Eskin, F Zadok, and S Stolfo. Data mining methods for detection of new malicious executables, May 2001.

[9] D Ddl F Li, A Lai. Evidence of advanced persistent threat: A case study of malware for political espionage. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6112333&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6112333, October 2011.

[10] Nikolaj Goranin and Cenys Antanas. Analysis of malware propagation modeling methods, April 2008.

[11] Andrea Shalal. U.s. firm crowdstrike claims success in deterring chinese hackers. http://www.reuters.com/article/2015/04/13/us-cyberattack-usa-china-crowdstrike-idUSKBN0N41PT20150413, April 2014.

[12] Virustotal. https://www.virustotal.com/cs/file/39df364a0bb19018127e0a258eb65f1d\\1ab2d6c86f1b9ab6fc5d93b8ca8c92f5/analysis/, September 2014.

[13] NightWatcher. http://greatis.com/cleanvirus/remove-malware/w32lohmys-atr-arquivo_solicitado-exe.htm, December 2014.

[14] David Emm, Maria Garnaeva, Victor Chebyshev, Roman Unuchek, Denis Makrushin, and Anton Ivanov. It threat evolution q3 2014. https://securelist.com/analysis/quarterly-malware-reports/67637/it-threat-evolution-q3-2014/, November 2014.

[15] Angelica Mari. Brazil tops banking malware list. http://www.zdnet.com/article/brazil-tops-banking-malware-list/, December 2014.

[16] Brett Stone-Gross and Russell Dickerson. Upatre: Another day another downloader. http://www.secureworks.com/cyber-threat-intelligence/threats/analyzing-upatre-downloader/, October 2013.

[17] Trend Micro. Upatre. http://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/upatre, June 2015.

[18] GReAT. 'red october'. detailed malware. https://securelist.com/analysis/publications/36830/red-october-detailed-malware-description-1-first-stage-of-attack/, January 2013.

[19] Symantec Security Response. Symantec protections for red october. http://www.symantec.com/connect/blogs/symantec-protections-red-october, January 2013.

[20] Kaspersky. 'red october' diplomatic cyber attacks investigation. https://securelist.com/analysis/publications/36740/red-october-diplomatic-cyber-attacks-investigation/, January 2014.

[21] Kim Zetter. Countdown to zero day: Stuxnet and the launch of the world's first digital weapon, November 2014.

[22] Richard Lagner. Stuxnet: dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 2011.

[23] Nicolas Falliere, Liam Murchu, and Eric Chien. W32.stuxnet dossier. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, February 2011.

[24] Ralph Langer. To kill a centrifuge. http://www.langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf, November 2013.

[25] CrySyS. Duqu: A stuxnet-like malware found in the wild. https://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf, October 2011.

[26] Symantec. W32.duqu: The precursor to the next stuxnet.

[27] Kim Zetter. Attackers stole certificate from foxconn to hack kaspersky with duqu 2.0. http://www.wired.com/2015/06/foxconn-hack-kaspersky-duqu-2/, June 2015.

[28] CrySys Lab. Duqu 2.0: A comparison to duqu, June 2015.

[29] Kaspersky Lab. The duqu 2.0, June 2015.

[30] Eduard Kovacs. Newly discovered 'turla' malware targets linux systems. http://www.securityweek.com/newly-discovered-turla-malware-targets-linux-systems, December 2014.

[31] Jen Weedon and Laura Galante. Intelligence analysts dissect the headlines: Russia, hackers, cyberwar! not so fast. https://www.fireeye.com/blog/executive-perspective/2014/03/intel-analysts-dissect-the-headlines-russia-hackers-cyberwar-not-so-fast.html, March 2014.

[32] Kurt Baumgartner and Costin Raiu. The penquin turla. https://securelist.com/blog/research/67962/the-penquin-turla-2/, December 2014.

[33] Dave Lee. Russia and ukraine in cyber 'stand-off'. urlhttp://www.bbc.com/news/technology-26447200, March 2014.

[34] GReAT. The epic turla operation, August 2014.

[35] Kaspersky. The epic turla (snake/uroburos) attacks. http://www.kaspersky.com/internet-security-center/threats/epic-turla-snake-malware-attacks.

[36] https://www.hex-rays.com/products/ida/.

[37] http://bochs.sourceforge.net/.

[38] http://www.windbg.org/.

[39] http://debugger.immunityinc.com/.

[40] http://www.woodmann.com/collaborative/tools/index.php/SysAnalyzer.

[41] http://www.tcpdump.org/.

[42] https://www.wireshark.org/.

[43] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. 2002.

[44] Brett Stone-Gross. The lifecycle of peer-to-peer (gameover) zeus, July.

[45] Peter Kruse. Threat report: W32.tinba (tinybanker) the turkish incident. 2012.

[46] Assaf Regev. Tinba malware reloaded and attacking banks around the world, September 2014.

[47] Fraser Howard. Exploring the blackhole exploit kit. March 2012.

[48] Stephen Ward. isight discovers zero-day vulnerability cve-2014-4114 used in russian cyber-espionage campaign. http://www.isightpartners.com/2014/10/cve-2014-4114/, October 2014.

[49] William Sanchez. Timeline of sandworm attacks. http://blog.trendmicro.com/trendlabs-security-intelligence/timeline-of-sandworm-attacks/, November 2014.

[50] NIST. Vulnerability summary for cve-2014-4114. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-4114, October 2014.

[51] Cisco. Blackhole exploit kit version 2. http://tools.cisco.com/security/center/viewIpsSignature.x?signatureId=2123&signatureSubId=0&softwareVersion=6.0&releaseVersion=S715, May 2013.

[52] David Fiser. Tiny banker trojan targets customers of

major banks worldwide. https://blog.avast.com/2014/09/15/tiny-banker-trojan-targets-customers-of-major-banks-worldwide/, September 2014.

[53] FIRST. Common vulnerability scoring system v3.0: Specification document. https://www.first.org/cvss/specification-document, 2015.

[54] Kafiene. Blackhole exploit kit goes 2.1.0, shows new url patterns, June 2013.

[55] Aurelian Neagu. The top 10 most dangerous malware that can empty your bank account. https://heimdalsecurity.com/blog/top-financial-malware/, August 2014.

[56] Kaspersky Labs. Kaspersky lab statistics: attacks involving financial malware rise to 28 million in 2013. http://www.kaspersky.com/about/news/virus/2014/Kaspersky-Lab-statistics-attacks-involving-financial-malware-rise-to-28-million-in-2013, April 2014.

[57] Dell SecureWorks Counter Threat Unit(TM) Threat Intelligence. Top banking botnets of 2013. http://www.secureworks.com/cyber-threat-intelligence/threats/top-banking-botnets-of-2013/, March 2014.

[58] Critical Intelligence. Sans icsthreat briefing. http://www.critical-intelligence.com/resources/papers/CI-Sandworm-BE2.pdf, October 2014.

[59] Bruce Schneier. More data on attributing the sony attack. https://www.schneier.com/blog/archives/2014/12/more_data_on_at.html, December 2014.

[60] US-CERT. Alert (ta14-353a) targeted destructive malware. https://www.us-cert.gov/ncas/alerts/TA14-353A, December 2014.

[61] Kyle Wilhoit and Jim gogolinski. Sandworm to blacken: The scada connection. http://blog.trendmicro.com/trendlabs-security-intelligence/sandworm-to-blacken-the-scada-connection/, October 2014.

[62] Brian Krebs. Researchers clobber khelios spam botnet. http://krebsonsecurity.com/2012/03/researchers-clobber-khelios-spam-botnet/, August 2013.

[63] Tom Fox-Brewster. Russian malware used by 'privateer' hackers against ukrainian government. http://www.theguardian.com/technology/2014/sep/25/russian-malware-privateer-hackers-ukraine, September 2014.

[64] MITRE. Vulnerability summary for cve-2014-4114. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-4114, October 2014.

[65] 4Armed. Galileo rcs running and espionage operation, July 2015.

[66] Crowdstrike. Putter panda. http://resources.crowdstrike.com/putterpanda/, June 2014.

[67] Mandiant. Apt1: Exposing one of chinas cyber espionage units, 2013.

[68] Symantec Security Response. Regin: Top-tier espionage tool enables stealthy surveillance. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/regin-analysis.pdf, November 2014.

[69] Kaspersky Labs Research Team. Equation: The death star of malware galaxy. https://securelist.com/blog/research/68750/equation-the-death-star-of-malware-galaxy/, February 2014.

[70] Ralph Langner. To kill a centrifuge. http://www.langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf, 2013.

[71] Udi Shamir. The case of gyges, the invisible malware government-grade now in the hands of cybercriminals. http://www.sentinel-labs.com/wp-content/uploads/2014/07/Sentinel-Labs-Intelligence-Report_0714.pdf, July 2014.

[72] Andy Greenberg. hopping for zero-days: A price list for hackers' secret software exploits. http://www.forbes.com/sites/andygreenberg/2012/03/23/shopping-for-zero-days-an-price-list-for-hackers-secret-software-exploits/, March 2013.

[73] Chris Borgen. Regulating the global market for zero-day exploits. http://opiniojuris.org/2013/07/15/regulating-the-global-market-of-zero-day-exploits/, July 2013.

[74] Jaziar Radianti, Eliot Rich, and Jose Gonzalez. Vulnerability black markets: Empirical evidence and scenario simulation. *IEEE*, 2009.

[75] Allan Friedman, Tyler Moore, and Ariel Procaccia. Cyber-sword v. cyber-shield: The dynamics of us cybersecurity policy priorities. http://www.nspw.org/papers/2010/nspw2010-moore.pdf, September 2010.