A Self-healing Framework for Enterprise networks to combat Botnets infections

Adeeb Alhomoud1, Anitta P. Namanya2, Jules Pagna Disso and Irfan Awan2
1Deanship of Scientific Research, Saudi Electronic University, Riyadh, Saudi Arabia
2SCIM, University of Bradford, Bradford, BD7 1DP, UK
a.alhomoud@seu.edu.sa, A.Namanya@student.bradford.ac.uk, jpagnadisso@nettitude.com, I.U.Awan@bradford.ac.uk

Abstract—Cybercrime costs governments, businesses and individuals a total loss of about $500 billion annually. This makes it one of the most profitable crimes in the world. Botnet is one of the most prominent tools used by cybercriminals to infect or compromise computer networks and steal important information. Infesting a computer is relatively easy nowadays with malware that propagates through social networking in addition to the traditional methods like SPAHM messages and email attachments. In fact, about 1/3 of all computers in the world are infected by malware which makes them susceptible to botnet exploitation. It is therefore important that anti-botnet solutions are implemented at all levels in a network. In this paper we present a self-healing based framework implemented in an Enterprise Network. Our design is inspired by how the human immune system adapts and defends against new attacks. We present the Self-healing architecture, the detailed modules used in the design and how they interact with each other to defend against the impact of botnet infections in the enterprise network.

Index Terms—Botnet, Self-healing, Enterprise, network, detection, cybercrime, immune system, malware analysis

I. INTRODUCTION

There is no doubt that cyber security is now a matter of national security. Government, business and personal lives are all affected by cybercrime. The total loss due to cybercrime is estimated at $500 billion annually[1] by Macfee. Cyber criminals use tools that allow them to commit crime anonymously. The most prolific tool known to date is Botnet. The term 'Botnet' describes numerous computers that are 'networked' by means of the botnet infection creating a virtual network that is controlled by an entity such as an individual, a criminal gang, or a government sponsored/sanctioned group. The concerns raised by Botnet are real and serious for various reasons. It is relatively easy to get infected. Curiosity or ignorance has led many people to click on links coming from unknown sources. Clicking blindly on file attachments to email. SPAM messages and drive by downloads are some of the common methods of Botnet infection. About 31.98% of all computers in the world, I.e. almost a 1/3 of all computers, are infected by malicious software[2].

The consequence of such a large-scale infection has caused serious concerns at all levels of society. Governments, businesses and financial institutions fear attacks such as distributed denial of service, information stealing and information disclosure on public forums. Home users are concerned with their financial transactions when using online banking. Zeus, one of the most recent Botnets, has been used to steal money from Internet banking users. The losses by such attacks are felt at all levels of society. It is imperative that solutions against Botnets are found and implemented at all levels to protect against the proliferation of such malicious activities.

Collective efforts are made by law enforcement, companies and government to tackle Botnet activities. The take down of Zeus in 2010[3] and Koobface in 2012[4] are just a few of the examples. These organised solutions are quite expensive, take a lot of time so they do not provide a quick or cost effective solution. The takedown of a botnet involves shutting down the central Botnet server and giving centralised advice to host users on how to clean the systems. The process of cleaning the host computers requires a lot of manual intervention which can result in some devices not being completely clean thus leaving them vulnerable to the next attack.

In this paper, after analysing the infection process of botnets, we designed a nature-based concept of a self-healing architecture that can be applied to fighting botnets in an enterprise network. Our solution was restricted to an enterprise network due to the known boundaries and the advantage of having all computers centrally monitored and accessible. Understanding how the body uses its immune system to heal from numerous diseases that can be considered attacks to the body, can be used to fend off botnet infections or repair compromised systems in a network. The self-healing concept is subdivided into integral parts that are defined to give the overall suggested architecture to strengthen the resiliency of the network against botnet attacks and effects.

II. EVOLUTION OF BOTNETS

For the last 20 years, Botnets have been very challenging to defeat because of the creativity of the botnet masters in trying to evade detection. Their propagation and detection evading techniques kept getting more complex over time. In 1993(Chao et al., 2009), EggDrop, the first bot was used to manage chat sessions in IRC channels. The communication features of this bot were then replicated for
the first malicious bot attack on IRC by implementing Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. This first evolution was realised in 1998 [5, 6] with the development of Global Threat bot - GTbot. This botnet changed the known characteristics of the bot because it run custom scripts, was based on mIRC propagated and accessed the hosts through TCP and UDP sockets while still responding to IRC events[7].

The year 2002 saw the birth of commercialisation and staged attacks by botnets, where C++ written botnet SDbot was sold by its creator [6, 8]. Agobot[8] defined the three stages of turning a host into a botclient;

1- Create back door in the system.
2- Disable antivirus software.
3- Block access to security vendor websites.

The two famous botnets which surfaced in 2003; Rbot, and Spybot [5-7] introduced additional functionalities such as key logging, data mining and sent out spammed instant messages as a way of propagating. They utilised remote access tools [9] which already included the key logging and connection forwarding functionalities to exploit the weaknesses in the Microsoft Remote Procedure call processes[10]. This changed the known propagation techniques of the botnets from random scanning to hit lists like email lists, buddy lists in AIM, etc. Rbot [7] introduced compression and encryption to evade detection. The success of Rbot led to arbitrary modification of instruction sets, polymorphism and metamorphism introduced to avoid detection [5, 6].

In early 2004, Agobot was taken down using code analysis and reverse engineering by Microsoft and law enforcement authorities [11]. The reaction to this takedown from SDbot saw a high rise of new variants, botnets attacking the antivirus and firewalls directly and migration to other communication protocols like HTTP (80) and P2P [8, 10, 11]. P2P was used to change the known architecture of botnets from centralised server to the botnets connecting to each other based on the botnet lists they have stored [5, 10, 12, 13]. HTTP is an essential protocol for Internet browsing and is also used for upload and download of binary data files which made it very viable and the idea of blocking this port is out of question in any network. Botmasters took over legitimate HTTP web servers to propagate their malicious code, by using a technique called a drive throw download. This was achieved by hacking a legitimate website, altering all the URLs links for all the files to a different web server that distributes the malware of choice. In some cases the targeted website was used as optional command and control servers. Botmasters registered and owned Domain Names that are similar to well known websites to take advantage of an innocent typo mistake by users in the web browser for example, google.com for google.com. Other techniques included using "fast flux" where one fully qualified domain name is allocated to different IP addresses to trick the users into opening these malicious sites [12, 14-16].

HTTP based Zeus, released in 2005, and was mainly used for data stealing. By 2007, Zeus was used by main cyber criminals to steal millions out of bank accounts [7]. It had a user interface and when new versions were released, older versions were released to the public for free. This meant th at it didn’t requir ehigh technical skill or cost a lot to set up a botnet centre. Cybercriminals used hard to defeat botnets defined by hybrid architecture and customised communication protocols e.g. Conficker, Waledac, variants of Zeus, MegaD, Storm, SpyEye, and Mariposa.

Table 1.

<table>
<thead>
<tr>
<th>Year</th>
<th>Changes</th>
<th>Botnets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993</td>
<td>1st unmalicious bot developed for IRC channel management</td>
<td>Eggdrop</td>
</tr>
<tr>
<td>1998</td>
<td>IRC malicious use of DoS attacks. Access Raw TCP and UDP sockets Run custom scripts</td>
<td>GTBot</td>
</tr>
<tr>
<td>2002</td>
<td>Commercialisation of Botnets C++ built program. Very robust and sophisticated programming. Sequential delivery of attack payload. Remote vulnerability scan enabled 1st P2P architecture adopted Centralised to Distributed C&amp;C topology</td>
<td>SDBot, Agobot</td>
</tr>
<tr>
<td>2003</td>
<td>Keylogging and connection forwarding. Data mining. Change of lists Compression and encryption to avoid detection. Changing the DNS host settings</td>
<td>Spybot, Sinit, Rbot, Slammer</td>
</tr>
<tr>
<td>2006</td>
<td>First HTTP botnet. Function targeted like stealing bank details. Botnet published updates Successful P2P botnets 1st Web service based botnets</td>
<td>Storm, Cuttwail, Srizibi</td>
</tr>
<tr>
<td>2007</td>
<td>Security vendor targeted attacks High processing power in botnets able to send large number of spam out per day. IP fast flux enabled SQL injections in legitimate sites. Hybrid topology adopted to avoid detection. 1st Social network site based botnets</td>
<td>Confi cker, Waledac, Mega-D, Koobface, Mariposa, Yxes, Ike B</td>
</tr>
<tr>
<td>2009</td>
<td>First known Mac and mobile botnets is discovered</td>
<td>TDL3,4, Agobot, PonyBotnet</td>
</tr>
<tr>
<td>2010</td>
<td>1st fully DNS based botnets Customised encrypted communication protocols.</td>
<td>SpyEye, Zeus variants, Zeus in mobile (ZitMo), Fakeplayer</td>
</tr>
<tr>
<td>2011</td>
<td>Zeus source code leaks leading to many variants. Mobile/ SMS based botnet in phones.</td>
<td>DroidDream, Cawitt, ZitMo, SureMo, Flashback, Dorkbot,Zeus Variants</td>
</tr>
<tr>
<td>2012</td>
<td>Smartphone botnets using social networking and attacking phone operating systems. Botnet in Mac.</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>Increased use of P2P and use of malicious URLs for malware propagation. Botnet Bitcoin mining, Brute force attacks using more than 90,000 IP address as seen in the attack on WordPress sites. First known abuse of Yahoo ads to propagate malware.</td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td>The creation and design of botnet crafted to steal crypto-currencies.</td>
<td>Ponny Botnet</td>
</tr>
<tr>
<td>2016</td>
<td>IoT botnets becomes an effective sources for DDoS attacks.</td>
<td>Mirai Botnet</td>
</tr>
</tbody>
</table>
A Botnet crackdown was started because of the high number of cybercriminals that used botnets. An example is the takedown of McColo which housed MegaD, late 2008, Mariposa, Waledec and Zeus in 2010 [3, 17]. Zeus source code leaked in 2011 [6] which resulted into a high number of Zeus variant used in criminal activities.

When Web 2.0 services like Facebook and twitter became a norm for many enterprises, botnets found new C&C surrogate and file storage servers; for example Amazon Elastic compute cloud (EC2) was used for Zeus configuration files storage in 2009 [7, 18, 19].

Successful social networking sites like Twitter and Facebook were manipulated from late 2008 to steal user information and the sites were used for storage and bandwidth for the C&C capabilities [20]. These botnets were difficult to track because they constantly changed their characteristics; for example, KoobFace, one of the most popular botnets using Facebook, Twitter and MySpace changed the architecture, its binary components constantly and had different updates. It used the Man in the Browser attack method where it solved captcha to send spam and defeated virtual keyboard by using sequential capture. In 2009 it was found that some infected machines are following twitter account feeds under the handle name "upd4t3" [21] which indicated that these applications were used as C&C servers through the use of RSS. The takedown of KoobFace in 2012 [4] gave the web 2.0 services some relief and botnet masters are now conquering the smartphone.

The first malware with botnet characteristics for iPhones, Ikee.B was discovered [22] in 2009. This discovery highlighted a new advanced in botnet platforms. Mobile computing continues to be the platform of choice for many end users. Given that protection of phones against cyber-attacks is a low priority, botnets are now taking advantage of the advancements in mobile / smartphones with 38% of all mobile users reported to have already experienced mobile cybercrime [23]. Zeus in the mobile (ZitMo) has been noted to be used in mobile banking to capture user information and to intercept the 2 way authorisation method introduced by banks to mitigate theft [24]. With the growing trend of Internet of Things(IoT), big data and cloud computing, botnets become a bigger threat to both the business and personal cyber world[25] with the botnet continuing to get smarter and harder to detect. In 2016, IoT devices was the target of a well effective and designed botnet called Mirai. It was reported that this botnet has infected more than 49,657 IoT devices and spotted in 164 countries[39].

III. EXISTING SOLUTIONS:

There has been a lot of interest in finding solutions against botnet. This section discusses some of the popular techniques used to fight botnet. These detection methods are classified as:

A. Passive Techniques

Packet inspection: This is basically matching different protocol fields or payloads of a packet against a predefined suspicious or abnormal content which is well implemented in Intrusion Detection System (IDS). This method has some drawback since it uses signature based detection. These are; 1) high number of false positives (some legitimate packets maybe classified as malicious packets) 2) By dividing malicious code into different payloads, some packets may evade detection. 3) It is difficult to do a full packet inspection especially in a high traffic networks.

Domain name server (DNS) traffic analysis: Botnet simultaneously start to query for new command and control server once their initial C&C has failed or was taken offline (group activity), an observation from the anomaly-based detection approach [27] that uses an algorithm to detect IRC based botn ets and "BotHunter"[28] where both inbound and outbound packets were observed and dialogue-based correlations were executed to detect infections. Snort, with enhanced and customised rules and two malware related plugin were used.

Analysis of spam records: During a SPAM campaigns, a number of infected machines send spam emails having the same pattern. The presence of infected machines is indicated by the Simple Mail Transfer Protocol (SMTP) generating a considerably higher number of DNS traffic than usual. The work in [30] demonstrated that the analysis of large amounts of spam can be used successfully to map spam bot.

Honeypots: It is an intentionally vulnerable resource that is deployed inside a network with the objective of soliciting attack or be compromised by a malicious entity. It is bound to discover new information about the practices and strategies used by malwares. It can be used to measure the attacks coming from botnets and which will allow network administrators identify the infected machines. These were used to gather information about bots.

B. Active Techniques:

The active measurement techniques are the approaches that involve interaction with the information sources that are being monitored.

Sinkholing: This is a counter measurement for cutting off the ebot or "zomb iemachines" from th ebotnet's command and control centre. Traditionally bots use a domain name to contact the command and control centre, this domain name is resolved by DNS query. Since Domain Name service (DNS) is a core service used to access the internet, it can be used to intercept the commutations between the bots and C&C. This is known by DNS Sinkholing. DNS sinkhole is spoofing the authoritative DNS server for malicious and unwanted hosts and domains and returning
a false IP address for these known command and control centres.

Reverse engineering: This is a technique used to reveal the functionality of the compiled program. It can be used to extract the information about the installation process of a bot and what technique it uses to propagate and communicate with the command and control centre. By using this approach researchers can develop counter measures from the information that is extracted such as detection signatures to be used with IDSs for example. There are two types of reverse engineering of a botnet i.e. Static, and Dynamic. The static reverse engineering is done without the actual execution of the binary file. While in dynamic the binary file is executed in a controlled environment.

It’s worth mentioning that active techniques can backfire because the implementation of any active techniques may notify the botmaster of such action which may result in the attack of the system that implemented this technique. Usually the attack will be a DDoS attack or misuse of any critical information or data stolen from the comprised systems, or simply updating the bot to evade these monitoring techniques.

C. Other Counter Measures:

There are also some other initiatives and counter measurements to fight botnets not classified in the detection techniques. Most of these initiatives focus on the command and control infrastructure of a botnet.

Blacklisting: Some recent approaches work with what is called real-time blacklists; these blacklists contain IP addresses that have been identified as infected machines. Having such list can be used by different institutions by blocking the lists. These blacklists are available as a service open for subscription with lists updated regularly.

Distribution of “fake - traceable” credentials: The most common botnet objective is stealing personal information and data, which is then dropped in the open zone for sale by the botmaster. The objective of this fake information is to contaminate the data stole/harvested by the bot in order to reduce the profitability of the entire botnet and create mistrust between the buyer and the botmaster. The crafted credentials can also be used to track the parties involved in the process by tracing their movement pattern. However, most banks are not willing to cooperate especially when it cross-border level, as even its fake account have to be filled with real money for a transactions to take place.

BGP Blackholing (Border Gateway Protocol): This protocol is widely used in the internet and consequently becoming the predominant technology for routing decisions. BGP is used to maintain the internet routing table which includes information between autonomous systems and the shortest path. Data provided by the blacklisting techniques mentioned previously can be used to change the routing polices and dropping malicious host to deny traffic to and from them and their network. These routing decisions are made at ISP level and affect hosts served by that ISP. The BGP blackholing is suitable for use against botnet that have a static command and control centre. This will result into saliently dropping and no routing communication between bots and their command and control server.

Direct takedown of command and control server: The initial step used to in this approach is to identify the command and control IP addresses. After identifying the IP address of the command and control centre, the service provider is then identified. Once this is done a request for takedown that server is initiated. This can be very challenging as the service provider may not be cooperative or, in some counties, the hosting provider is not allowed to cease the operation by law. A takedown request depends mainly on the country’s law and terms and conditions of the service provider.

Blocking port 25 by ISPs: This is a preventive measure applied by ISPs to cut down the amount of spam mails using their network. This approach is based on the idea that bots/spammers will use open relay mail servers or mail exchange server via port 25 to distribute spam.

The "Sybil Attack" for P2P: This approach is aimed at manipulating the routing in peer-to-peer networks by adding a large number of crafted and independent peers that are controlled from a central component. The Sybils will continually advertise themselves to existing real bots participating in the peer-to-peer network. The bots will answer any route requests with their identifiers and this way, th erouteing tab les will be "poisoned" over time by Sybils.

Other non-technical approaches exist such as raising user awareness, Central incident help desk to provide consultation on the treatment of bot infections, and dedicated laws on cybercrime.

IV. SELF-HEALING SYSTEMS

A self-healing implementation enables a system to recognize that it is not operating correctly and with no or minimal human intervention, restores the system to its normal working state or maintains the system in a working condition until human intervention [31]. This section will cover the definition of the self-healing system along with the various approaches that researchers have been applying in self-healing systems.

Since self-healing systems can be regarded as very general and similar to what is expected of fault-tolerant / survivable systems, it is worth mentioning that fault tolerant systems include stabilization techniques and replication techniques. Some studies, address self-healing systems as a subordinate to fault-tolerant systems. Self-healing systems focus on methods for stabilizing, replacing, securing, isolating and strategies to repair and prevent faults. Self-healing applications’ operation are processes of isolating a faulty component, taking it off line, fixing the failed component and/or reintroducing a replacement/fixed component [32].

The concept of self-healing originates from a paradigm of nature. This biological system has is imbedded within nature, where it allows the biological systems to adapt and heal to changing environments. Self-healing is well defined by how the human immune system works. The immune system is specifically designed to fend off any
infection using the given body resources and has managed to defend against new and old diseases over an evolution of ages. As described in [33], this complex system has cells (lymphocytes) that constantly patrol the body to detect and kill off any unfamiliar cells. This requires them to be able to tell from the body cells (self-cells) and foreign cells (nonself-cells).

By analysis, the human immune system is self-organising, distributed and adaptive. It has the ability to detect, recognise and isolate foreign cells, decide whether to attack the antigen or adapt to the way the antigen works, store the resulting information for future use and kill off un-needed cells in the body.

The role of nature in natural immune systems for protecting animals from viruses, bacteria and toxins is very similar to self-healing in the computer field [34]. Self-healing has been addressed in other area of research such as operating systems self-healing where some techniques such as code reloading, component isolation and automatic restarts have been implemented [35]. Self-healing has also been addressed in embedded systems where mapping between resources and tasks have been considered. This means that on a network of resources, more instances of the same tasks are running simultaneously, some of them as idle and on event of failure these idle tasks take over [36].

The self-healing systems are designed to enable the continuous availability of the resources. This system design's main objectives are to ensure that a healthy system is always in operation and that it can survive any kind of attack. These objectives are achieved by attributes which are further defined by the applied processes. The self-healing solution works on the condition that it will always use defined policies to perform checks on the resources in the system it controls at intervals to ensure smooth operation and detect any anomalies. It is also worth noting that although self-healing systems can be designed to be self-reliant, human intervention still remains a very important component.

V. PROPOSED SELF-HEALING FRAMEWORK

This section introduces the proposed self-healing architecture which aims to resilience any botnet infection in the Enterprise network. Since the scope of our proposed model is the Enterprise network, the system is designed to fit and work in an Enterprise network. The proposed self-healing framework will take advantage of the characteristics of the domain network such as having the agents running as a service. It will address network survivability and mitigation of any botnet infection. The proposed system is integral to the "security in depth" concept as some IDS systems might not detect any issue in the network; they work on a signature based method [37], and, in addition to that, some botnets may mimic the human behaviour in performing some tasks that will not be detected by IDS systems. This section begins with an overview of the proposed Framework and the modules that it consists of.

A. Proposed Design

The proposed framework is designed to work in an enterprise environment. This can only work in a controlled environment where the initial network state is always known [38]. The workstation/agent clean state is uniform for all the computers connected to the network. Traffic in and out of the network can be closely monitored. System back-ups are centrally done and therefore any trusted changes to the system are always known. Only the central authority i.e the system administrator is allowed to make system affecting changes. The proposed framework consists of the self-healing server and the agent running service which is run on the workstation.

B. Architecture modules

The proposed self-healing framework consists of the following modules:

- Detection Module: This is the module that captures and compares system state against the clean state. If a system change is detected, it generates alerts that are used by the healing module.

- Healing module: This module is responsible for reversing system changes detected by the previous module. If it encounters any errors, it generates a report and activates the controlling module otherwise; it generates a report for further analysis which is logged by the system.

- Communication Module: This module processes all the communication between the server and the agent components of the framework.

- Controlling Module: This module is responsible for the isolation of the infected node from the enterprise network. It is activated by the healing module when the infection cannot be reversed. It changes the network card configuration to connect the infected node to a quarantined network thus taking it off main enterprise network "offlin". This is done to reduce the infection impact and to avoid further botnet propagation through the network. The takedown is performed by the agent service after the instructions have been received from the system. A report logged and sent to the administrator asking for human intervention.

- Reporting module: This is responsible for any output from the self-healing system. It ranges from logged report files to system alerts sent to the
administrator. It is responsible for archiving reports and system log files for further analysis. It also has an archiving system for all the logs that have been recorded. This module has the ability to generate reports daily, weekly, and monthly.

The proposed framework consists of two main parts; the self-healing server and the agent running service on the workstation. These two parts interact using different methods implemented under the different modules to perform the required operations.

C. Design Scope

Our system is designed to work in an enterprise Environment. The design can only work in a controlled environment where the initial network state is always known. The workstation/agent clean state is uniform for all the computers connected to the network. Traffic in and out of the network can be closely monitored. System back-ups are centrally done and therefore any trusted changes to the system are always known. Only the central authority i.e. the system administrator is allowed to make system affecting changes.

D. Design system Services:

The system consists of the self-healing server and the agent running service which is runs on the workstation. The two interact with each other as shown in Figure 1.

*The self-healing server:* This is the component that keeps all the detailed information about the network self-healing activities. It stores the configuration files and an image of the clean system state that are to be kept in a specific folder with read only permissions. This folder name is the machine's IP address. It then stores other integrity changes in the system in a file named with the prefix of the IP address. The folder is then sorted into different categories. The first function extracts the integrity changes and defines the type and location of the change. Then it sorts this information and saves it in a file named with the prefix of the IP address. This file is then stored in a specific folder with read only permissions. The name of the file is the IP address of the machine. It then adds the IP address of the machine into the machine's list of machines "iplist.dat" file to be contacted later on. The pseudo code for the integrity changes function is:

**Agent running service:** This component runs as a domain service and it has all the modules stated previously (Detection module, Healing module, communication module, reporting module and control module). To ensure continued availability of the network services, this service runs in the background of the hosts in the network. It also runs in a continuous time wait loop monitoring the health of the system to ensure that the system can survive the effects of any attack. The interval time is set by the system administrator.

VI. SELF-HEALING FRAMEWORK IMPLEMENTATION

As described previously in the proposed framework section, it will consists of two main parts, server and agent. The section covers the implementation of this frame work.

*Self-healing server:* The self-healing server consists of a number of functions. One of the main functions is the log analyser. The alerts generated by the HIDS are scattered in number of tables and some fields hold more than one piece of information. Therefore the creation of this function is important as it extracts what is needed to identify the machine IP address and the integrity change of the files and/or registry keys. It is the starting point for our self-healing success.

A. Integrity changes

As the first task of the self-healing server is to extract the alerts and understand the logs generated by the HIDS, a function to sort the results into two different categories is needed. Since the changes in the system are either integrity related or new files added to the OS, two functions are created to handle those categories. The first function extracts the integrity changes and defines the type and location of the change. Then it sorts this information and saves it in a file named with the prefix of the eIP address of the machine e.g. "192.168.0.12-integrity.dat". This file is stored in a specific folder with read only permissions. The folder name is the IP address of the machine. It then adds the IP address of the machine into the "iplist-integrity.dat" file to be contacted later on. The pseudo code for the integrity changes function is:
Figure 2 Communication sequence algorithm

Figure 3 Integrity Fix algorithm

B. New Files added to OS

In this subroutine the function starts by sorting all the logs alerts that are related to newly added files to the OS folder. It extracts the location and name of the file and then stores the information in a prefixed IP address file name with a suffix of "\newfiles.dat\". This file is then stored in a read only folder that named after the IP address of the machine in question.

The function then adds the IP address of the machine in the file named "IP-list-newfiles.dat" to be used by the other modules. Below is the pseudo code for new files extraction

The communication module: This communication module 'th e communicator' is responsible for reading the eplist.dat files that are generated from the previous functions and contacting the agents on the machines in questions. The communication module provides the communication protocol between the agents and the server.

The Self-healing Agent: The self-healing agent is a domain running service running on all the domain workstations. This service can be configured to run using the GPO (Group Policy Object) in the active directory. Similar to the self-healing server it is coded using Perl language. This takes advantage of speed and low resources utilization characteristics of the language. This section details the algorithms built for the self-healing agent.

Figure 4 Communication sequence algorithm

Just like the server, the agent has a communication module. This module ensures that the machine is always listening on port 7777 for instructions. Once the agent receives the instruction, it determines what algorithm to follow. If the instruction received is related to integrity check, it will initiate the integrityfix() function.

Since the Self-healing system is designed to allow read only access to the folder associated with the IP address of the machine to be healed. The integrity fix function starts by verifying the IP of the host machine. Once the IP address is determined, it locates the folder associated with its own IP address in the self-healing server and reads the files saved there line by line. The agent determines the file integrity change and the registry change. It then gets all the PIDs and names of running processes and compares them to the white listed processes. These white listed processes must not be killed, such processes are the HIDS agent, self-
healing agent. Once it identifies the white listed processes it force kills all the other running processes.

Since the self-healing server is a Linux based, the syntax used in file path differs from that used in windows OS system. Therefore, the healing agent first swaps slash from "\" to "\" in order to locate the infected file. The next step is to check the location of the file, this is important as the process of healing the file will differ if it is inside the system32 folder. The reason is that MS Windows have a hidden folder %SystemRoot%\System32\DllCache that runs as a backup for the system files in system32 folder, the OS will copy automatically any deleted file in the system32. Therefore, the self-healing agent renames the target file in the system32 to filename.old, force deletes the file located in the DllCache folder then copies the genuine file to DllCache folder, sleeps for 5 seconds (to give time for the system to copy) then copies the genuine file again to system32 folder. It then cleans the leftovers by deleting filename.old. The agent checks if there are any errors. If an error is encountered, the agent notifies the self-healing server and then initiates the quarantine function to change the IP of the machine to 10.0.0.X and have it connected to a quarantined network.

VII. EXPERIMENTS:

Due to the constraints of length on this paper, we will try to summarize the implementations, results and findings of the experiments. The test bed used in the experiments simulates an enterprise domain network running with all the main components that are in an enterprise network. The test bed consists of DHCP server, DNS server, Domain controller, 16 workstations running on three main PCs (illustrated as blocks) under VMware. An industrial HP Procurve managed switch is used to monitor all the network traffic by assigning a monitoring port. And finally self-healing framework, which consists of MySQL server, File server, HIDS server, Self-healing server. Additional tools were used such as Wireshark and Volatility which is a completely open collection of tools for the extraction of digital artifacts from memory (RAM).

The two test scenarios used to verify the designed framework were chosen based on the common types of botnets. HTTP and IRC type botnets are very common since they are easy to setup and manage. HTTP based botnets are quite hard to guard against since they use the essential Port 80 for their communications while IRC is easiest and most common protocol used by botmasters. An IRC and HTTP botnet were used to test the design in scenarios Alfa and Bravo respectively using the setup test bed. Each scenario went through three phases. In each phase the system state was captured, analysed and the results were compared. The following figure illustrates the experiments phases.

**Scenario Alfa:**

The first scenario involved having all the workstations in the test bed infected with an IRC botnet. This required the introduction of two other components to the test bed which were an IRC server running on Linux Centos 6 and Traffic monitoring machine running Wireshark on Linux OS to capture all the traffic passed in the network. UnrealIRCd which is an open-source IRC server daemon was used. Once installed, an initial test was carried out by using an open-source IRC client to make sure that the IRC server was running with no issues. The next step was to configure the bots binary file with the IRC server address and channel the infected nodes needed to join in order to receive instructions and provide report or any data extracted from the infected machine. In this experiment all the bots are asked to join a channel called "botnet" on the IRC server. Once the machines in the test bed are infected with the IRC botnet, they are able to communicate and log into the channel called "botnet" on the IRC server and then wait for further instructions. We then compare the clean state with the infected state and below are the highlights of the findings:

When examining the captured image of the system using the parameter Active connections, an active connection is listed between the infected machine and the command and control center of the IRC bot. The below figure illustrates that there is an active connection between the examined machine and the server with the IP 192.168.0.2, which is the botnet command and control IRC server that we setup for this experiment.

![Figure 5: Experiment Phases](image)

![Figure 6: Volatility- Active Connections Post infection](image)
The highlighted process is a malicious process that was created after the system had been infected. RxBot is known to create random process names on the infected machine. The random file/process name created for this infected machine is shown below.

`0x02f2914 kbenzk.exe 1428 844`

The captured traffic during the experiment showed that there was communication between the IRC botnet server and the infected nodes.

After all the infected machines are registered and logged into the bot C&C channel, the Healing phase is started. Once the healing process is completed, Volatility is used to capture the system state. The state capture called the Healed state is compared to the infected state. The Self-healing framework successfully managed to resilience the botnet by removing any integrity changes and reverting the system to the clean state. This is confirmed by monitoring the registry files, the random added file created by the botnet, and analysing the network traffic using Wireshark. Wireshark was setup to monitor the test bed for 72 hours, and then examined to see if there were any connections established from any machine (previously infected) after it's been healed. The examined traffic did not show any traffic initiating to or from the healed machines other than the normal traffic.

The following screen capture shows that the process named kbenzk.exe no longer run on the system.

Examining the botnet command and control centre showed that it had lost contact with the previously infected machines on its channel as shown in Figure 11.

**Scenario Bravo:**

In this section, all the workstations are infected with an HTTP botnet. The setup was more complicated because the botnet needs to have internet access so that it can obtained the botnet server IP address from a compromised website. This required setting up a public botnet C&C server on the internet. A dedicated private server was setup for the duration of the experimentation and immediately disconnected for legal purposes. A gateway was also introduced in the test bed in order to facilitate the communication between the workstations and the botnet server.

Once we had the HTTP botnet command and control up and running, and we captured a clean system state and then infected the test bed machines with Warbot botnet.
Confirmation that all the machines are infected and registered with the botnet C&C server by monitoring the connection state and the network traffic. Figure 12 illustrates that there was an active connection between the infected machine and the server with the IP 188.121.62.xxx port 80 (Botnet server IP and HTTP standard port). It is obvious that this is a bot activity originating from the infected machine to the botnet command and control centre.

Figure 12 Volatility- Active Connections Post infection

Examining the process with PID 1668 that initiated the active connection, the process is found to be smss.exe. MS windows has a genuine smss.exe therefore, to make sure that this is not the genuine process, the process needs to be further examined by taking a closer look into the process list and their parent process by comparing the clean and infected states.

Figure 12 Volatility- Process Scan- Post infection

In Figure 12, the highlighted process smss.exe PID 1428 with PPID 620 is the process that is making the connection while the genuine process smss.exe is not and is always under the parent PID which is system folder as noted in the clean state. Figure 13 shows the Bot’s registering into the command and control server i.e. Warbot server. Once the bot is registered, it is assigned a unique ID and the bot checks the command and control server on the interval time they initially programmed to do so. If there is any instructions awaiting the bot’s action, it is actioned in the next on the next check. For the experiment, the bot is instructed to download Netcat that allows the botmaster to open a reverse shell directory.

Figure 13 Wireshark Traffic- Bot communication with Command and control server

Once we confirmed that the bots were communicating with the command and control centre and receiving instructions, the 3rd phase of the experiment which is the healing was initiated. To confirm that the healing process works, the system was examined using the Volatility tool and Wireshark was used to analyse the traffic. Using the forensic tool Volatility, we examined the active connections and there were only normal connections between the healed machine and the server with the IP 192.168.0.1 which is the enterprise domain server, and there is no suspicious activity as shown in figure 14.

Figure 14 Volatility- Active Connections- After healing

Traffic is captured over 72 hours and analysed for any malicious activity. A Wireshark filter (ip.addr==188.121.62.233) was applied to display all the traffic coming from and going to the command and control centre and there was no communication between the healed nodes and botnet command centre.

The quarantine function which is implemented when the framework is not able to return the infected node to its initial clean state was tested. During the test, the infection machine was taken off the main Enterprise network and sent to the quarantined network. The node disconnection was logged for human intervention.

VIII. FUTURE WORK AND CONCLUSION

The designed self-healing framework utilises the key characteristics and attributes of the nature’s immune system to fend off botnet attacks. It utilises its four main components to heal the infected nodes by replacing the
changed/affected system components. The self-healing components interact with each other to successful heal or take the infected machine off the network in case of unknown errors/obstacles. In event of failed healing, the infected machine is taken off the network to prevent it from communicating with other network nodes. This prevents the node from infecting other nodes in the network or initiating attacks on the network servers.

The framework is designed to be integrated into the enterprise network security infrastructure. It takes advantage of HIDS to detect any system state changes. The self-healing framework integrated into the network assists in maintaining the availability and survivability of the network. The design was tested using two of the main types of botnets; IRC and HTTP based botnets. Experiments were run in a controlled environment and the self-healing framework achieved 100% detection and healing rates for both test scenarios. All the 16 workstations were healed in less than 96 seconds, which includes the 20 seconds waiting time between tasks in both the test scenarios. The success of all the designed components of the self-healing framework shows that the framework works and its implementation in an Enterprise Network would make it resilient against botnet infection.

One of the extensions that can be added to the self-healing framework, is that the nodes can act as a mini self-healing server and heal other nodes. These nodes can be assigned by the success/clean rating. Since, the human factor plays a major role in infecting the workstations, the self-healing server can have a ranking mechanism. This ranking system would be based on the statistics that a specific node has a low infection rate can act as a mini self-healing server and help in generating commands and tasks for the infected node. This is helpful in a large enterprise, where some subnets are not physically in the same area or even city of the main data centre where the self-healing framework/server is located. By having this functionality to the design, it will safe bandwidth traffic between locations. It is also beneficial in a very large number of computers to have a redundancy of mini self-healing system. This idea was inspired by the Waledac botnet, where infected router nodes exchange IP lists. This can be implemented by having the self-healing trusted nodes share lists of the IPs of the infected machines on the network. Therefore, when a "mini-self-healing" node is off-line for any reason, other neighbouring nodes can perform the self-healing.

REFERENCES


A Self-healing Framework for Enterprise networks to combat Botnets infections (POC)

Dr. Adeeb M. Alhomoud
Outline

• Brief background
• Motivation
• News update
• Solution
• Framework Design
• Experiments
What is a Botnet

Combined of two words: the first part “bot” is derived from the word “robot” while the second part is short for network.

*It can be defined as:*

- a software robot that operates as an agent which is capable of performing certain commands automatically, repeatedly and simulate human activities.
- Network of infected computers controlled by a botmaster.
Botnet terminology

**Bot**
- An application that listens and performs actions sent from the herder
- Installed on a victim machine
- Allows you to use your own exploits and install other malware

**Herder**
- Botmaster or controller
- Owns and sends command via the C&C center via control channel
- Power or Money

**Control channel**
- Used to send and receive data to bots
  - IRC
  - HTTP
  - HTTPS
  - Peer 2 Peer
<table>
<thead>
<tr>
<th>Year</th>
<th>Changes</th>
<th>Botnets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1993</td>
<td>1st malicious bot developed for IRC channel management</td>
<td>Eggdrop</td>
</tr>
<tr>
<td>1998</td>
<td>IRC malicious use of DoS attacks. Access Raw TCP and UDP sockets Run custom scripts</td>
<td>GBTot, AGBot</td>
</tr>
<tr>
<td>2002</td>
<td>Commercialisation of Botnets C++ built program. Very robust and sophisticated programming. Sequential delivery of attack payload. Remote vulnerability scan enabled 1st P2P architecture adopted Centralised to Distributed C&amp;C topology</td>
<td>SDBot, Agobot</td>
</tr>
<tr>
<td>2003</td>
<td>Keylogging and connection forwarding. Data mining. Change of bot lists from random scanning to target lists Compression and encryption to avoid detection</td>
<td>Spybot, Siuit, Rbot, Slammer</td>
</tr>
<tr>
<td>2004</td>
<td>Metamorphism and Polymorphism. Mass-mailing. DNS tunnelling, typosquatting and Domain flux</td>
<td>Polybot, Bobux</td>
</tr>
<tr>
<td>2006</td>
<td>First HTTP botnet. Function targeted like stealing bank details. Botnet published updates Successful P2P botnets 1st Web service based botnets</td>
<td>Rustock, Zeus</td>
</tr>
<tr>
<td>2007</td>
<td>Security vendor targeted attacks High processing power in botnets Ability to send large number of spam out per day. IP fast flux enabled</td>
<td>Storm, Catfowl, Snabot</td>
</tr>
<tr>
<td>2008</td>
<td>SQL injections in legitimate sites. Hybrid topology adopted to avoid detection. 1st Social network site based botnets</td>
<td>Conficker, Waledac, Mega-D, KoolHase, Mariposa, Yax, Ikey B</td>
</tr>
<tr>
<td>2009</td>
<td>First known Mac and mobile botnets is discovered</td>
<td>TDL3.4,</td>
</tr>
<tr>
<td>2010</td>
<td>1st fully DNS based botnets Customised encrypted communication protocols.</td>
<td>SpyEye, Zeus variants, Zeus in mobile (ZeMo), Fake_taxonomy, DroidDream, Cottol, ZipMo, StibMo, Flashback, Dorkbot, Zeus, Variants</td>
</tr>
<tr>
<td>2011</td>
<td>Zeus source code leaks leading to many variants. Mobile/ SMS based botnet in phones.</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>Smartphone botnets using social networking and attacking phone operating systems. Botnet in Mac.</td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>Increased use of P2P and use of malicious URLs for malware propagation. Botnet Bitcoin mining, Brute force attacks using more than 90,000 IP address as seen in the attack on WordPress sites. First known abuse of Yahoo ads to propagate malware.</td>
<td>Dorkbot, Zeus, Variants</td>
</tr>
<tr>
<td>2014</td>
<td>The creation and design of botnet crafted to steal crypto-currencies.</td>
<td>Pony Botnet</td>
</tr>
<tr>
<td>2016</td>
<td>IoT botnets becomes an effective source for DDoS attacks.</td>
<td>Mirai Botnet</td>
</tr>
</tbody>
</table>
The first bot was created as a useful feature in Internet Relay Chat (IRC) by Jeff Fisher in 1993.

Malicious bots started to surface as the first malicious bot running on IRC was called GTBot.

Peer-to-Peer bots started in 2000.

2006 HTTP based botnets were developed.

2016 IoT botnets.
Botnets Topologies

Centralized (Http/Https, IRC)

E.g. Zeus, Warbot, BlackEnergy / RxBot
Botnets Topologies

Peer 2 Peer
E.g. Storm, Waledac
Botnet Usage

- Spam
- DDOS
- Email Harvesting
- Installing other malware/botnet as a service to other herder
- Proxies (can be rented by other attacker)
- Click fraud
- Phishing attacks
- The infrastructure can be rented or sold
- … etc
Means of propagation
(infection vectors)

- Emails that use social engineering to pursue the user to download a file.
- Instant messaging tools such as MSN messenger,
- Vulnerabilities of the OS.
- Script injection
- USB keys, SD cards
- Peer-to-Peer sharing protocol.
- Human curiosity.
- Network shares.
- Weak / null passwords
Spam Emails From Top 10 Botnets
(millions of messages)

Source: McAfee Labs, 2016.
Top Network Attacks

- Denial of Service: 35%
- Browser: 30%
- Brute Force: 18%
- SSL: 4%
- DNS: 5%
- Scan: 2%
- Web: 2%
- Others: 2%

Source: McAfee Labs, 2016.
Botnet getting smarter

- Use Social network sites as C&C.
- Encrypted
- Change the C&C when feeling compromised
Some of the well known botnets

**STORM:**
Notable for being one of the first peer-to-peer botnets

**Torpig:**
- Famous for data theft, Domain generating algorithms. collected about 70GB of data which included thousands of bank account credentials belonging to hundreds of financial institutions in a period of 10 days
- 1st algorithm computes a weekly domain name
- 2nd algorithm computes a Daily domain name
- Then it attempts to connect to the hardcoded domains

**Waledac**
- uses a robust communication method with both P2P and HTTP. Although Waledac is considered as a pure peer to peer network it utilises a backend-server. Once a computer gets infected with the Waledac malware, it is classified as Spammer node or Repeater node

**Conficker:**
- Dictionary attacks on administrator passwords
- Updates itself every 2 months
- Patches MS08-067 to open backdoor in Server service to trick Microsoft that it already patched the system!
- Reappeared again in late 2015 in police held cameras.
- 6.5 million approx computers infected

**Zeus:**
- Variants, lots of Variants.
- 1.2 million computers were infected
Motivation

- Cyber crime estimated to cost the world $500 Billion annually.
- Cyber crime cost in UK reached £27 Billion per annum.
- Botnets are one of the popular/powerful tools used by cyber criminals.
- Takedown always involve Microsoft and government agencies (FBI) to hunt and shutdown- **a better and quicker solution is needed.**
- Botnet attacks focus on .organisations network (use of resources).
- Botnets are aware of what you visit (knows online banking).
World's largest 1 Tbps DDoS Attack launched from 152,000 hacked Smart Devices

Do you know — Your Smart Devices may have inadvertently participated in a record-breaking largest cyber attack that Internet has just witnessed.

If you own a smart device like internet-connected televisions, cars, refrigerators or thermostats, you might already be part of a botnet of millions of infected devices that was used to launch the biggest DDoS attack known to date, with peaks of over 1 Tbps of traffic.

DDoS attack that disrupted internet was largest of its kind in history, experts say

Dyn, the victim of last week's denial of service attack, said it was orchestrated using a weapon called the Mirai botnet as the ‘primary source of malicious attack’
A Linux botnet is launching crippling DDoS attacks in excess of 150Gbps

The XOR DDoS botnet can generate attacks more powerful than most businesses can withstand.

Security

White hats, FBI and cops team up for Dorkbot botnet takedown

Your four-year reign of terror is (temporarily) over
Ponmocup is the '15 million' machine botnet you've never heard of
Skilled VXers have built 25 plugins, made 4000 variants, say crack security team

Chinese ad firm pwns Android users, creates hijackable global botnet
Horrid marketing outfit roots user phones, exposes devices to malware hell
Pony up: Botnet successfully targets Bitcoin

Password-lifting network converted to cryptocoin-thievery

By Richard Chirgwin, 25 Feb 2014

CBR

HUNDREDS OF POS SYSTEMS INFECTED WITH NEMANJA BOTNET

Security

by CBR Staff Writer | 26 May 2014
Solution

IT MUST BE:

• Automated i.e. self-healing with minimal human interaction.
• Generic – works on any botnet.
• Quick.
• Have all the self-healing attributes.
• Easy to maintain and expand.
• Easy to implement and integrate in any Enterprise network Be an addition to the mainstream.
• Addition to the security in depth.
• *Designed/implemented/test in an Enterprise network.*
Framework Design

[Diagram showing the framework design with details on Workstation (Agent) and Self-Healing Server modules, including detection, healing, control, and communication modules.]
Experiments

- Two scenarios: Alfa (IRC botnet) and Bravo (HTTP botnet).
- Each scenario went through three phases.
- Network Traffic was captured and analysed.

The system state was captured, analysed and the results were compared.

- Phase 1: System state before infection
- Phase 2: System state after infection
- Phase 3: System state after self-healing
Experiment: Alpha (IRC)

Botnet test bed (IRC)
Experiment: Alpha (IRC)

Before Infection

After Infection
Experiment: Alpha (IRC)

After Infection
Experiment: Alpha (IRC)

HEALED

```
volatility-2.3.1$ sudo ./vol.py -f '/media/x/C038-E432/fromSha
Foundation Volatility Framework 2.3.1

<table>
<thead>
<tr>
<th>Local Address</th>
<th>Remote Address</th>
<th>Pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.11:1045</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
<tr>
<td>192.168.0.11:1111</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
<tr>
<td>192.168.0.11:1164</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
<tr>
<td>192.168.0.11:1164</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
<tr>
<td>192.168.0.11:1164</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
<tr>
<td>192.168.0.11:1164</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
<tr>
<td>192.168.0.11:1164</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
<tr>
<td>192.168.0.11:1164</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
<tr>
<td>192.168.0.1:49156</td>
<td>192.168.0.1:49156</td>
<td>720</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Name</th>
<th>PID</th>
<th>PPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x001cd3c0</td>
<td>svchost.exe</td>
<td>1509</td>
<td>708</td>
</tr>
<tr>
<td>0x001df318</td>
<td>wmladap.exe</td>
<td>452</td>
<td>1068</td>
</tr>
<tr>
<td>0x0030f8b0</td>
<td>lsass.exe</td>
<td>720</td>
<td>656</td>
</tr>
<tr>
<td>0x0032d020</td>
<td>svchost.exe</td>
<td>1668</td>
<td>708</td>
</tr>
<tr>
<td>0x007e7a78</td>
<td>TPAutoConnect.exe</td>
<td>280</td>
<td>708</td>
</tr>
<tr>
<td>0x007f3be0</td>
<td>explorer.exe</td>
<td>544</td>
<td>480</td>
</tr>
<tr>
<td>0x00b6d9f0</td>
<td>TPAutoConnSvc.exe</td>
<td>760</td>
<td>708</td>
</tr>
<tr>
<td>0x02def3c0</td>
<td>svchost.exe</td>
<td>1812</td>
<td>708</td>
</tr>
<tr>
<td>0x02e54650</td>
<td>svchost.exe</td>
<td>904</td>
<td>708</td>
</tr>
<tr>
<td>0x02e719c6</td>
<td>ossec-agent.exe</td>
<td>1628</td>
<td>708</td>
</tr>
<tr>
<td>0x02ec5650</td>
<td>vmtoolsd.exe</td>
<td>1840</td>
<td>708</td>
</tr>
<tr>
<td>0x02ed4020</td>
<td>svchost.exe</td>
<td>1668</td>
<td>708</td>
</tr>
<tr>
<td>0x02ee4270</td>
<td>vmacthlp.exe</td>
<td>888</td>
<td>708</td>
</tr>
<tr>
<td>0x02f69f0</td>
<td>TPAutoConnSvc.exe</td>
<td>760</td>
<td>708</td>
</tr>
<tr>
<td>0x02f7e680</td>
<td>explorer.exe</td>
<td>544</td>
<td>480</td>
</tr>
<tr>
<td>0x02f77580</td>
<td>spoolsv.exe</td>
<td>1372</td>
<td>708</td>
</tr>
<tr>
<td>0x02f7e3c0</td>
<td>svchost.exe</td>
<td>1500</td>
<td>708</td>
</tr>
<tr>
<td>0x02f7bda0</td>
<td>winlogon.exe</td>
<td>656</td>
<td>412</td>
</tr>
<tr>
<td>0x030b0d00</td>
<td>DumpIt.exe</td>
<td>3668</td>
<td>544</td>
</tr>
<tr>
<td>0x0318820</td>
<td>alg.exe</td>
<td>1700</td>
<td>708</td>
</tr>
<tr>
<td>0x0318678</td>
<td>svchost.exe</td>
<td>1112</td>
<td>708</td>
</tr>
<tr>
<td>0x0318a690</td>
<td>svchost.exe</td>
<td>1272</td>
<td>708</td>
</tr>
<tr>
<td>0x0318b10</td>
<td>lsass.exe</td>
<td>720</td>
<td>656</td>
</tr>
<tr>
<td>0x0319b0b0</td>
<td>services.exe</td>
<td>708</td>
<td>656</td>
</tr>
<tr>
<td>0x0319d5a0</td>
<td>svchost.exe</td>
<td>980</td>
<td>708</td>
</tr>
<tr>
<td>0x031f08a8</td>
<td>smss.exe</td>
<td>412</td>
<td>4</td>
</tr>
<tr>
<td>0x03315988</td>
<td>GoogleUpdate.exe</td>
<td>1660</td>
<td>1552</td>
</tr>
<tr>
<td>0x033206a0</td>
<td>csrcs.exe</td>
<td>632</td>
<td>412</td>
</tr>
<tr>
<td>0x0333a660</td>
<td>System</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0x04a830e0</td>
<td>vmmacthlp.exe</td>
<td>888</td>
<td>708</td>
</tr>
<tr>
<td>0x04a9d650</td>
<td>svchost.exe</td>
<td>964</td>
<td>708</td>
</tr>
<tr>
<td>0x0421e650</td>
<td>vmtoolsd.exe</td>
<td>1840</td>
<td>708</td>
</tr>
<tr>
<td>0x04246568</td>
<td>spoolsv.exe</td>
<td>1372</td>
<td>708</td>
</tr>
<tr>
<td>0x04293da0</td>
<td>winlogon.exe</td>
<td>656</td>
<td>412</td>
</tr>
<tr>
<td>0x0429f8550</td>
<td>GoogleUpdate.exe</td>
<td>217</td>
<td>1660</td>
</tr>
</tbody>
</table>
```
Experiment: Alpha (IRC)

INFECTED

HEALED

Bots have closed connection and exited the channel
Experiment: Alpha (IRC)

INFECTED

HEALED
Botnet test bed (HTTP)
Experiment: Bravo (HTTP)

Before Infection

After Infection
Experiment: Bravo (HTTP)
Experiment: Bravo (HTTP)

HEALED
Experiment: Bravo (HTTP)
Take off the infected machine of the Enterprise network if the healing was not successful for any reason.
Thank you for listening