Tumbling Down the Rabbit Hole: Exploring the Idiosyncrasies of Botmaster Systems in a Multi-Tier Botnet Infrastructure

Chris Nunnery, Greg Sinclair, and Brent ByungHoon Kang
University of North Carolina at Charlotte
{cenunner, gssincla, bbkang}@uncc.edu

Abstract

In this study, we advance the understanding of botmaster-owned systems in an advanced botnet, Waledac, through the analysis of file-system and network trace data from the upper-tiers in its architecture. The functionality and existence of these systems has to-date only been postulated as existing knowledge has generally been limited to behavioral observations from hosts infected by bot binaries. We describe our new findings for this botnet relating to botmaster interaction, topological nuances, provided services, and malicious output, providing a more complete view of the botnet infrastructure and insight into the motivations and methods of sophisticated botnet deployment. The exposure of these explicit details of Waledac reveals and clarifies overall trends in the construction of advanced botnets with tiered architectures, both past, such as the Storm botnet which featured a highly similar architecture, and future. Implications of our findings are discussed, addressing how the botnet's auditing activities, authenticated spam dispersion technique, repacking method, and tier utilization affect remediation and challenge current notions of botnet configuration and behavior.

1 Introduction

Recent increases in malware sophistication are largely driven by a desire to generate profit in a thriving underground economy despite advances in malware defense. Botnet architectures, which are no exception to this trend, have become considerably more advanced than their ancestral counterparts, often functioning as elaborate and resilient infrastructures supporting numerous services. In the Waledac and Storm botnets, notable for their longevity and resilience, bot nodes have been used for relaying web content and malicious binaries, providing peer data to other nodes, and participating in fast-flux DNS services.

While the host-level behavior of bot nodes and their malicious activities in these advanced, tiered architectures has been explored in numerous works [3, 13, 11, 10, 4, 9, 12], their complete architectures are poorly understood. Current knowledge is largely limited to the behavior exhibited by bot binaries on infected hosts and information obtained through network probing, which does not reveal the structure, configuration, or behavior of the systems in the upper-tiers deployed and directly controlled by botnet operators.

In this paper we expose the behavior and configuration of systems in the highest tiers of the Waledac architecture. In doing so, we illuminate previously nebulous areas of this infrastructure, allowing one to better understand how these systems are utilized in the network to command, protect, and ultimately, generate profit as a part of the complete architecture. By revealing details relating to the deployment and functionality of botmaster-operated systems in a multi-tier botnet architecture, those in the information security community can better respond to these threats and further their understanding of the motivations and techniques employed to operate a botnet.

The protocol details of the Waledac botnet with regard to communication and encryption are outside the scope of this study and are not included. These details, along with the functionality and behavior of nodes in the lowest two tiers in Waledac, comprised of infected hosts and known as *Repeaters* and *Spammers*, have been accurately described in existing literature [9, 1, 12].

The remainder of this paper is structured as follows: In Section 2, we accurately describe the botmaster-owned components of the Waledac infrastructure based on obtained network traces and file-system data from the systems in the highest tiers of its architecture. Section 3 describes the implications of our findings, documenting previously unknown behavior. A discussion of related work is included in Section 4. We conclude in Section 5.

2 Exploring Waledac's Components

2.1 Overview

Waledac emerged in late 2008 as a possible successor to the Storm botnet. The Waledac botnet on a macroscopic scale can be described as a spam-generating phishing infrastructure with fast-flux functionality. Waledac employs a hierarchical architecture with four tiers. The bottom two layers, the *Repeater* and *Spammer* tiers, are comprised of infected hosts, while the top two layers, the *UTS* and *TSL* tiers, are deployed and managed by the botnet operator(s). This topology is shown in Figure 1.

For the sake of clarity, it is worth noting that the terms TSL and UTS have only arbitrary meaning. The identifier

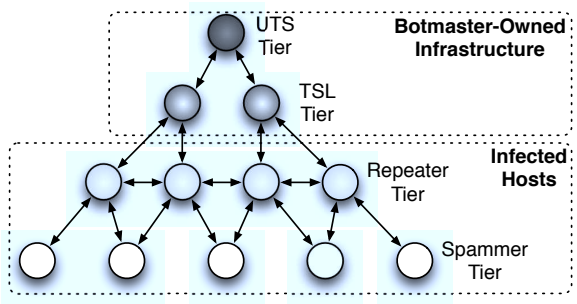

Figure 1: The hierarchical topology of Waledac.

TSL comes from the Windows registry key Repeaters use to store the list of servers in this tier. The term *UTS* was assigned by the authors to identify both the tier and single server located at the highest point in the hierarchy. The term is derived from the label *Upper Tier Server*. To avoid confusion, when referring to the tier (e.g. the network hierarchy level) the term *UTS tier* is used, while when referring to the Command and Control server of Waledac, the term *UTS* is used alone.

2.2 Analysis Methods

Reconstructing and discerning the functionality of the upper tiers in the Waledac infrastructure was possible due to both file system data from two TSL systems and network traces from the TSL and UTS systems. Servers located within these top two tiers are hosted by third-party hosting providers. The TSL tier is located largely in the Netherlands, Germany, and Russia. Through cooperation with two of the affected hosting providers in the Netherlands, we were able to obtain two of the TSL server images immediately after they were taken offline. We were also provided with network trace data and file system artifacts from the single UTS server also located in the Netherlands. This rare opportunity provided detailed insight into the operation and makeup of the servers in the upper-tiers of Waledac's infrastructure. We also verified the behavior and functionality of nodes in the lower two layers, comprised of infected hosts, by studying network traffic and reverse-engineering bot binaries.

2.3 Infected-Host Systems

The bottom two layers in the Waledac botnet are comprised of *Repeaters* and *Spammers*. Both of these layers are comprised of systems infected with Waledac binaries. Nodes are relegated to either of these layers based on IP addresses; publicly accessible systems become Repeaters. Systems located behind NAT devices are designated as Spammers. Each Repeater node operates as a HTTP proxy as well as a DNS server and facilitates communication between the Spammer tier and the botmaster. Nodes in the spammer tier, which retrieve commands in a *pull*-based scheme, are used to distribute unauthenticated spam and harvest local data such as email addresses and network credentials. As these layers and the communication scheme they use have been correctly described in various literature [1, 9, 12] the paper focuses on the tiers above the Repeater layer, the functionality and very existence of which have only been speculated.

2.4 Botmaster-Owned Infrastructure: TSL

The TSL is the last victim-exposed tier and the first obfuscated tier in the Waledac infrastructure. As such, researchers have come to make assumptions about the intent and functionality of this tier [9]. It is important to understand that there are actually two sides to a TSL server (and the TSL tier on the whole): the *public* side and the *hidden* side.

In terms of visible functionality, the TSL tier is a simple proxy and obfuscation layer. From the perspective of the Repeater tier, the TSL tier does little more than to relay requests from the Repeaters (and by extension, the Spammer tier and neighboring Repeaters) to the next, unknown tier in the botnet. The assertion that the TSL tier is itself only one layer amongst additional upper, hidden tiers has been largely unproven until the research disclosed in this paper. The fact that these servers operate in a coordinated manner indicates that either the servers are in some way self-organizing to share information or they independently report to a central repository and control server. Given the streamlined architecture of the Waledac botnet when compared to its more complex ancestor, Storm, more weight can be given to the centralized control server theory as this model matches more closely with the rest of the botnet's design. We have found that there is indeed an additional tier beyond the TSL tier consisting of a single, controlling server known as the UTS.

The hidden side of the TSL tier, which faces into the higher tiers of the Waledac botnet, contains several surprising characteristics. Whereas the public side of the TSL proxies communication between the Repeater (and subsequently the Spammer) tier to the upper tiers, the private side of the TSL does the inverse by marshaling communication in the opposite direction, from the upper tier back into the Repeater tier. What hasn't been known until now is that the TSL servers within the TSL tier perform additional functions such as targeted spam generation. Before delving into the functionality of the TSL servers, it is worthwhile to understand the construction of these servers.

2.4.1 Deployment and Configuration

At its core, each TSL rides on a pre-furnished Linux operating system image. Upon this image the botmaster installs several standard services and applications such as the Network Time Protocol (`ntp`) daemon, the DNS server BIND, PHP, OpenVPN, BZip2, and the nginx [15] proxy. In order to ease the process of installation, the botmaster deploys many of these services with a pre-generated configuration file common to all TSL servers. These configuration files are not by themselves accurate upon their initial installation, however. The *nginx.conf* and *iptables* files require the botmaster to manually configure the proxy settings and filtering rules for the next higher tier before the TSL can properly marshal communication between the Repeater tier and the next tier (UTS).

By default, the *ngnix.conf* file, seen in Figure 2, contains a simple set of proxy transformations. The primary function of the proxy transformations is the translation of requests from the public side of the TSL tier to a format acceptable to the higher tiers of the botnet. These transformations focus primarily on ensuring that the request originated from within the Repeater tier of the botnet, as indicated by the user-agent field of the HTTP request containing the string `LMK`. With three exceptions (`/pr/`, `/lm/`, and `/tds/`), the proxy will return a HTTP 404 error code if the user-agent does not contain the `LMK` substring. This effectively weeds out non-Repeater tier originating requests while at the same time preventing additional work for the UTS tier.

```
location /mr.txt {
     proxy_pass http://85.x.x.x/lm/data/hosting/mr.txt;
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP $remote_addr;
}
location /pr/ {
     proxy_pass http://85.x.x.x/lm/data/hosting/partnerka/;
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP $remote_addr;
}
location /tds/ {
     proxy_pass http://{removed}.name/tds/;
     proxy_redirect off;
     proxy_set_header X-Real-IP $remote_addr;
     proxy_set_header User-Agent $http_user_agent;
     proxy_set_header Referer $http_referer;
     proxy_pass_header Client-Host;
}
location / {
     if ($http_user_agent !~ (.+)LMK$) {
          error_page  403 404 500 502 503 504 /404.html;
          return 404;
     }
     proxy_pass http://85.x.x.x/lm/data/hosting/;
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP $remote_addr;
}
location ~ ^/[a-z]*\.(png|htm)$ {
     if ($http_user_agent !~ (.+)LMK$) {
          error_page  403 404 500 502 503 504 /404.html;
          return 404;
     }
     rewrite ^/[a-z]*\.(png|htm)$ /lm/main.php last;
}
location /lm/ {
     if ($http_user_agent !~ (.+)LMK$) {
          return 404;
          error_page  403 404 500 502 503 504  /404.html;

     }
     proxy_pass     http://85.x.x.x/lm/;
     proxy_redirect off;
     proxy_set_header Host $host;
     proxy_set_header X-Real-IP    $remote_addr;
}
```

Figure 2: The default TSL *nginx.conf* configuration for the TSL servers defines the translation of HTTP requests from the public interface of the TSL to the next tier in the hierarchy (UTS). This configuration specifies how Repeater nodes must conform to a specific user-agent in order to pass traffic through the TSL tier.

The three exceptions to the `LMK` rule relate to traffic originating from outside of the Repeater tier. These exceptions establish the fact that what was originally considered a simple proxy tier is actually an entry point for third party access. The exceptions allow third party actors (such as affiliates) to interface with the Waledac botnet in order to facilitate the underground commerce the Waledac botnet generates. The `/pr/` exception allows the botmaster to transfer content between the botmaster-controlled tiers (TSL and UTS) without significant overhead and provides a means for phishing webpages to serve content such as graphics and executables.

The concept of the TSL servers act as proxies to the obfuscated tiers is by no means a new revelation. Common sense and experience have shown that botmasters routinely generate malicious servers using pre-configured scripts and packages. As evident by the creation of the TSL servers, this deployment takes an insignificant amount of time allowing the botmaster to quickly "stand-up" new servers when needed or after a takedown of existing servers.

The fact that the botmaster is actively using the TSL servers for additional functions such as targeted spam generation is until now an unknown fact. After installing and configuring the TSL server to act as a proxy, the botmaster installs PHP, proxychains, BIND and a custom package known as `php_mailer`. The combination of these applications allows the botmaster to send specifically targeted email from the TSL through a series of proxies.

2.4.2 Spamming Models: HQS and LQS

- Low Quality Spam (LQS)

The bulk of Waledac's spam is generated through the Spammer tier. Nodes in this tier receive their spam campaigns from the UTS via the various tiers between them. The end result of this type of spam campaign is bulk spam

with a higher probability of being blacklisted due to the originating IP addresses being dynamically assigned (e.g. residential cable modems or DSL services). The Spammer nodes that ultimately transmit this type of Low Quality Spam (*LQS*) keep detailed statistics on if a particular piece of spam from a particular campaign destined for a particular email address was successfully transmitted. This indicates that the spam sent by the Spammer tier is possibly generated as part of a bulk order from another actor using Waledac as a spam generator. The use of statistics allows the purchasing actor to determine the ultimate number of spam messages delivered and the final bill for the campaign.

- High Quality Spam (HQS)

As *LQS* is likely to be impeded by blacklists, the botmaster has developed a solution to this particular problem by relying on legitimate email accounts and their corresponding SMTP servers. The botmaster, as part of the initial deployment of a TSL server, installs a custom PHP application called `php_mailer`. This application is a simple bulk mailer that is coupled with an open source package known as proxychains. Equipped with a collection of validated SMTP login credentials, the botmaster generates between 100 and 300 instances of `php_mailer`. The bulk mailer connects to a cloud of proxy servers via SOCKS5. These proxies in turn connect to the specified SMTP server via TCP port 25. The `php_mailer` application uses valid login credentials to authenticate with the SMTP server before sending multiple spam emails from the victim's account. The result of this attack is a spam campaign with a higher probability of success, resulting in a High Quality Spam (*HQS*) campaign.

The source of the SMTP credentials is presently unknown however, it is within reason to assume the Waledac bots, which now actively monitor an infected machine's network traffic for SMTP, HTTP and FTP credentials, are the source of this intelligence. The `php_mailer` application contains two small PHP scripts which download both a list of current SOCKS5 proxies and validated SMTP server credentials. These scripts access the TSL tier using the `wget` application. The location of the two lists contains the path `/pr/` resulting in the TSL tier forwarding the request to the UTS tier. Therefore, while the request is destined for the TSL tier it is actually the UTS tier that contains the required information.

The source of the targeted email addresses for the HQS campaign is obtained from a download from UTS by the TSL server in question. The download takes the form of `http://Neighboring TSL Server IP/ pr/ short name.gz`. This URL is itself interesting by the fact that the *nginx* proxy will retrieve the information from the *partnerka* [1] directory indicating, again, that the spam is part of a paid service to third party underground actors. As explained in the next section, one possible source for the information at the UTS level is the spam clearinghouse called *spamit.com*. This fact further adds credence to the theory of Waledac's spam as a service model. Figure 3 illustrates the process of generating *HQS*.

Unlike *LQS* campaigns, the botmaster puts significant effort into *HQS* campaigns by not only using compromised SMTP accounts to circumvent blacklists, but by testing the campaigns before they are delivered. Contained within the `php_mailer` directory is a list of target email addresses for *HQS* campaigns. Before the target list is engaged, the botmaster uses a list labeled "test" to run the first batch of the campaign. The test file contains four email addresses each with the same username but with a differing domain name (*yahoo.com, hotmail.com, mail.ru, and gmail.com*). The four email addresses are repeated several times in the test file ensuring that the same spam email is sent to each of the email accounts multiple times. In this way the botmaster can determine if the current spam campaign will face any blocks at the email provider level. If the spam emails will end up in the spam or junk folder of the email provider, the botmaster can adjust the spam message and run the test again. Once the spam emails have been successfully delivered to the inbox of each of the four test email accounts the botmaster can safely assume the spam will not be blocked by the recipients' providers and the main *HQS* campaign may commence. This same technique along with the same test email addresses were used by the Storm botnet. This is another example of the strong relationship between Storm and Waledac.

When running 100 to 300 `php_mailer` instances, a significant amount of DNS queries must occur to locate the appropriate SMTP server for each victim. The sheer quantity of the *MX* and *A* records generated by this process can raise flags when using an ISP's DNS servers. To avoid this situation or perhaps to increase the throughput for DNS queries, the botmaster manually installs the BIND server on the TSL server during the initialization process. The DNS server is configured as a simple caching DNS server that uses the root DNS servers instead of the ISP's DNS servers to handle *MX* and *A* record location. Evidence of this behavior was found in the `named.run` file of the TSL.

2.5 Botmaster-Owned Infrastructure: UTS

For the most part, despite its additional functionality, the TSL tier is largely a buffer between the infected machine tiers (Repeater and Spammer layers) and the upper tiers of the botnet. With regards to the botnet communication,

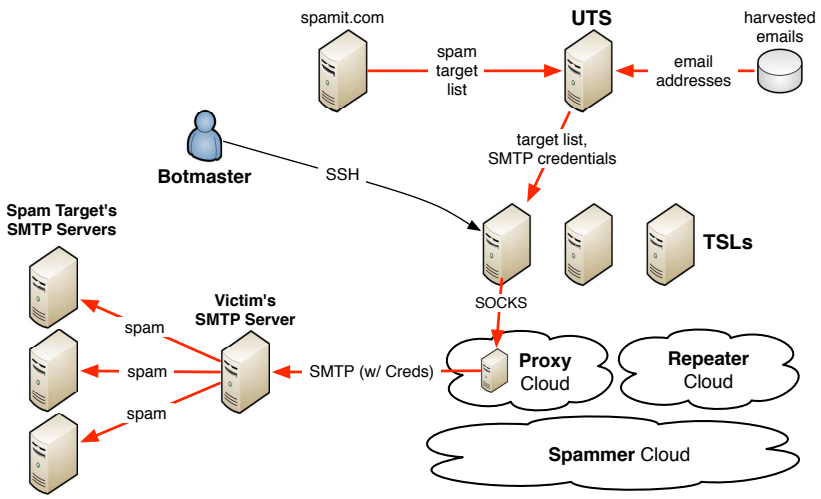

Figure 3: Waledac supports the unique ability to send spam using stolen email credentials. This provides a high delivery success rate. We refer to this as *High Quality Spam* (HQS). In the *Low Quality Spam* (LQS) campaigns, nodes in the Spammer tiers send the more common type of spam described in Section 2.4.2.

the TSL tier relies on communication with the Upper-Tier Server (UTS) tier in order to pass instructions to the infected machine tiers. As with the TSL servers, we were able to obtain file system artifacts from the UTS server giving unprecedented insight into the Command and Control infrastructure of the botnet.

2.5.1 Configuration

From the network traffic alone it is possible to determine the operating system the UTS tier is based upon due to its periodic requests to the `yum` repository to look for updated packages. In these requests, XML request containing the operating system and platform can be found. The UTS server under observation sent a standard HTTP request made up of the string `GET /pub/centos/5.3/os/x86_64/repodata/repomd.xml`. This indicates that like the TSL servers, the UTS servers are based on CentOS 5.3 running on a 64-bit platform.

The core of the control software uses PHP. Like other botnets such as Zeus and Capricinus, the use of PHP gives the botmaster the ability to use a variety of server platforms without the need to reconfigure or recode the botnet's master control system. With the availability of hosting services offering multiple OS platforms, the trend of using largely OS independent control software will continue.

2.5.2 Role in Spam Dissemination

While the TSL may be responsible for the production of HQS and the Spammer tier responsible for LQS, ultimately the UTS tier is responsible for acquiring the information to place in the spam campaigns. Evidence of this behavior is found in a series of requests to the spam warehouse website at *spamit.com* [7]. The Spamit system is a known clearinghouse for so-called Canadian Pharmacy websites. On multiple occasions in a very short period of time (less than one hour) the UTS used the `wget` application to query the Spamit website for new domains to enter into the current spam campaign. The UTS queries the spamit.com server using a simple HTTP GET request that takes the form of `GET /export.php?aid={affliateID}&mode=personal&design=blue&secure={hash token}`.

The request generates a simple list of domain names such as *http://offerled.com*, *http://toldtool.com* and *http://hourshine.com* with each domain name separated by a newline break. This information is disseminated downward into the botnet depending on which type of campaign is currently being produced by the botnet (HQS or LQS). This use of *spamit.com* shows that the Waledac botnet is undoubtedly part of an affiliate network where money is given for revenue generated by spam campaigns. This revenue source may account for some portion of the funding required to support the TSL tier and UTS tier hosting costs.

2.5.3 Third-Party Repacking Service

While Spamit and Rogue A/V vendors may be a source of income, Waledac relies on a specialized pay service for its daily operation. Waledac does not employ rootkits

in order to hide from antivirus applications, but rather it uses a constantly changing set of packed binaries to avoid signature detection. There are approximately 50 known versions of Waledac in the wild, but there are over 3200 different binaries for these 50 versions [14]. The Waledac binary is routinely repacked resulting in the large number of binaries each with a unique MD5 hash (or signature). The frequency at which these binaries are repacked is exceedingly high and requires automation. From the UTS network traffic, we observed the UTS employing a third party service provider at *crypt.j-roger.com* and *cservice.j-roger.com* to repack Waledac binaries. In order to repack a binary, the UTS server sends a POST request to one of the two URLs `crypt.j-roger.com/api/apicrypt2/{16 hexadecimal digit hash}` or `cservice.j-roger.com/api/apicrypt2/{16 hexadecimal digit hash}`. Contained within the POST is an action form detailing the specifics of the repacking request along with the binary to pack in a modified version of Base64.

On average, the packing service at *j-roger.com* returned a repacked binary in 4 seconds. This allows the UTS to repack multiple binaries in a very short period of time. During a two hour period, Waledac was observed requesting (and receiving) 157 binaries through the *j-roger.com* service. When the service returns the binary to the UTS, the server uses a similar format as the request.

2.5.4 Auditing Activities

The Waledac botnet is open to observation as this paper and others related to this topic have shown [8, 9, 1, 12]. The botnet has limited protection from poisoning attacks at the Repeater tier. To monitor and prevent such attacks, the botmaster uses the UTS as a self-auditing component to ensure that only legitimate Waledac bots are introducing traffic into the botnet. Simulating the behavior of a Waledac Repeater node is possible given the open XML format the botnet uses for communication. Provided that the simulated Repeater node properly handles the encryption and compression required to transmit the XML through the botnet, it is a trivial matter to construct a simulated Repeater node that appears to be a legitimate Repeater node. The Waledac botmaster has developed creative solutions to determine simulated (illegitimate) Repeater nodes.

The first test performed by a UTS server when auditing a node is known as the Executable Request Proxy (*ERP*) test. When developing a simulated node, it is conceivable that the researcher would prevent the node from being used to propagate Waledac or other malicious nodes. As such, the node would drop any request for an executable by an outside (victim) entity. The *ERP* test plays against this fact by having the UTS issue a request for a specific file named *readme.exe*. The UTS will directly contact the node under audit with the URL `/readme.exe`. A real node will pass this request to the TSL server which will in turn pass the request to the UTS server. Therefore, it is possible for the UTS to track from start to finish the request and reply for *readme.exe*. The contents of readme.exe consist of two bytes which simulate the DOS header of a PE/COFF file, the letters `MZ`. A variation of the *ERP* test is also performed randomly when the UTS requests *readme.txt* instead of *readme.exe*. The reply to this variation of the *ERP* test is the string `Hello`. During a two hour period, the observed UTS server issued 597 *ERP* tests.

The second test performed by a UTS server focuses on the DNS component of a Repeater node. Since a simulated Repeater node would not necessarily need to participate in the DNS portion of the Waledac fast-flux network, it is conceivable that researchers would simply ignore DNS requests. To test for this possibility, the botmaster introduced a new domain into the Waledac fast-flux configuration named *hellohello123.com* in August of 2009. The domain currently does not have an associated name server and as such cannot be resolved though the *.com* Top Level Domain (TLD). The Domain Response (*DR*) test uses the fast-flux network configuration in order to determine the validity of the audited node. The UTS issues a DNS lookup for *hellohello123.com* by querying the node under review. Since *hellohello123.com* is part of the fast-flux network configuration, a valid repeater would return one of the predefined IP addresses from the configuration data. A simulated repeater would potentially fail this test by either returning invalid information or not responding at all. Therefore the *DR* test can identify invalid Repeater nodes based solely on their response to a specific, non-resolvable domain query. The UTS issued 693 *DR* tests during a two hour period of observation.

3 Implications

In this section, we delineate the implications from our analysis of the complete Waledac infrastructure. Specifically, we describe aspects of the botnet infrastructure which contradict current notions of how advanced botnets function.

3.1 Multi-Service Tiers

The body of research on the Storm botnet and early studies into the function and structure of Waledac concluded that some layers in the hierarchy were used exclusively as proxies to occlude the location of nodes owned by the botnet operator. While this is partly true, we have found that

the TSL systems, in a tier previously described as a "proxy layer," provide additional services. In addition to relaying traffic between the UTS and Repeater tiers, the TSL systems are also used in the High Quality Spam (*HQS*) campaigns. Botnet operators log in directly to the TSL systems via `ssh` to start instances of `php_mailer`. This finding also demonstrates that botnet operators use more than one tier to distribute commands, as the UTS tier is also used.

The trend of using layers of the botnet for numerous functions has been widely documented for the Repeater tier, but the inclusion of the TSL layer as a multi-service tier shows this trend persists throughout the botnet. Our findings show that modern, advanced botnet architecture are immensely capable, and serve as full infrastructures, not mere collections of systems loosely coupled together. Nodes in the network have specific roles and responsibilities, and are leveraged to their fullest potential for profitability. Rarely are systems used for a single purpose.

By understanding that multiple tiers may engage in previously unknown activities (such as proxies engaging in spam generation), it is now possible to re-evaluate enumeration techniques to look for overlap between these multiple tiers. Moveover, finding nodes that behave outside of the normal functionality for their destinated tier can lead to the discovery of more advanced services offered by a particular botnet.

3.2 Authenticated Spam

Stock et al. described Waledac's ability to harvest email account credentials [12], but we have further discovered that stolen email credentials are used by the botnet for spam dissemination, as seen in Figure 3. We consider authenticated spam distribution a deviation from conventional botnet behavior [5].

Unlike traditional spam dissemination techniques which rely on open mail relays or use bot nodes themselves to send mail directly, the Waledac infrastructure allows the operator to distribute mail using *SMTP-AUTH*. Members of the proxy-cloud which receive email credentials from the TSL layer effectively log in to an SMTP server to send emails. Mail distributed in this manner is more likely to be successfully delivered as it could evade blacklists. In Waledac, mail distributed using this technique is limited and requires initiation from the botnet operator. Waledac also distributes spam using the more traditional unauthenticated method in mass quantities, but these spam campaigns are more autonomous.

While still a relatively uncommon practice, the use of authenicated spam can lead to spam filtering problems in the near future. Botnets that automate authenicated spam will defeat spam filtering based on dynamic IP addresses. Although the Waledac botnet requires botmaster intervention to initiate an authenticated spam campaign, it would be exceedingly easy to automate the process in order to farm these spam campaigns to worker nodes.

3.3 Node Auditing

In peer-to-peer systems, node *auditing* or *vetting* keeps untrusted or malicious nodes from stressing or polluting indexing systems, wasting bandwidth, and eavesdropping in the network. In Storm, evidence of node auditing has not been disclosed, though aggressively crawling the botnet was known to trigger defensive denial of service attacks [2]. Beyond the careful control of peer lists, where active nodes were not excised from routing tables when new nodes announced their presence, Storm accepted illegitimate peers willingly. This resulted in many successful infiltrations by several research groups [5, 4]. A misconception about Waledac has been that node-auditing does not occur. As we discussed in Section 2.5, we have found that the UTS system routinely audits the Repeater layer.

It should be noted that the botnet itself features a blacklist containing untrusted and illegitimate nodes in the Repeater tier. This blacklisting ability provides the ability to block nodes engaging in aggressive or disruptive system activity on the network and those that fail to pass *ERP* and *DR* tests. With botnets such as Waledac employing blacklisting techniques, node-vetting becomes a frightening prospect which can interfere with attempts to measure and directly combat these threats.

Node auditing is a simple process that blends in with normal botnet network chatter. This makes node auditing difficult to detect, but not impossible. When more large scale botnets begin to employ a similar technique, it will become increasingly more important to properly understand every aspect of the bots that make up the botnet in order to infiltrate the network to gain intelligence or perform disruptive actions.

4 Related Work

In an exploration of the history of botnets and emerging peer-to-peer architectures, Grizzard et al. presented a case study on an early version of the Storm botnet [3]. Stewart, who presented the first exploration of Storm's intricacies [11] furthered the understanding of this botnet by later exposing its hierarchical nature [10]. Numerous research groups have proposed possible mitigation strategies and infiltration techniques for modern, formidable botnets [5, 9]. Enumeration accuracy and completeness has also been explored [4, 6].

The communication protocol of the Waledac botnet was first documented by Sinclair et al. [8, 9]. Population estimates and a monitoring methodology for this botnet were

pursued by Stock et al. [12]. These works discussed the protocols in the lower Repeater and Spammer tiers, where a researcher can observe network traffic from running bot samples. None of these works have exposed the architectural and implementation details of the tiers higher in the botnet, which are owned by the operators of the botnet.

5 Conclusions

Many of the information security community's correct notions about Waledac and other botnets of its caliber are due to the ability to dissect binaries and observe bot malware during execution. The tiered architecture of this botnet naturally obfuscates the behaviors, purposes, and very existence of nodes in upper tiers. Our unique insight was possible due to network traces and file system data obtained from these systems deployed by the operators of the botnet.

In this study, we described the deployment specifics of the botmaster-owned nodes in the Waledac infrastructure, system purposes and behaviors, and botmaster interaction. We disclosed how Waledac functions as a robust and complete infrastructure, providing numerous services throughout its tiers. We also discussed Waledac's technique for sending authenticated spam, which challenges the notion that botnets only send unsolicited email directly or via open mail relays. Waledac's novel approach to identify illegitimate Repeater nodes using a form of self-auditing was also discovered. The third-party binary repacking service Waledac employs was also discussed, which has not been documented in prior studies of advanced botnet behavior.

It is the intent of this paper to advance the understanding of sophisticated botnet architectures, such as those used by Waledac and Storm, to broaden the understanding of their deployment and operations, and to provide the information security community with knowledge necessary to defend against these advanced threats. Given the information disclosed in this study and the alarming growth rate of architectural and behavioral sophistication it implies, the information security community should respond accordingly.

6 Acknowledgements

C. Nunnery and B. Kang are in part supported by a grant from ETRI (B551179-09-01-00) and a GAANN fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators and do not necessarily reflect the views of their employers or funding sponsors.

References

[1] J. Calvet, C. Davis, and P M. Bureau. Malware Authors Don't Learn, and That's Good. In *MALWARE 2009: The Fourth Annual Conference on Malicious and Unwanted Software*, October 2009.

[2] G. Keizer. 'We're Not Scared' of Storm, Say Researchers. `http://tinyurl.com/ygyyrc2`, October 2007.

[3] J. Grizzard, V.Sharma, C. Nunnery, B. Kang, and D. Dagon. Peer-to-Peer Botnets: Overview and Case Study. In *First Usenix Workshop on Hot Topics in Understanding Botnets*, April 2007.

[4] B. Kang, E. Chan-Tin, C. Lee, J. Tyra, H. J. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon, and Y. Kim. Towards Complete Node Enumeration in a Peer-to-Peer Botnet. In *ASIACCS 2009: ACM Symposium on Information, Computer and Communication Security*, March 2009.

[5] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. Spamalytics: An Empirical Analysis of Spam Marketing Conversion. In *CCS 2008: ACM Conference on Computer and Communications Security*, pages 3–14, New York, NY, USA, 2008. ACM.

[6] C. Kanich, K. Levchenko, B. Enright, G. Voelker, and S. Savage. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. In *Proceedings of the First USENIX Workshop on Large Scale Exploits and Emergent Threats*. USENIX Association, April 2008.

[7] D. Samosseiko. The Partnerka - What is it, and why should you care? In *Virus Bulletin Conference*, September 2009.

[8] G. Sinclair. Blog Post: Waledac's Communication Protocol. `http://tinyurl.com/y9u4v98`.

[9] G. Sinclair, C. Nunnery, and B. Kang. The Waledac Protocol: The How and Why. In *MALWARE 2009: The Fourth Annual Conference on Malicious and Unwanted Software*, October 2009.

[10] J. Stewart. Protocols and Encryption of The Storm Botnet. `http://tinyurl.com/yhgwlod`.

[11] J. Stewart. Storm worm DDoS Attack. `http://tinyurl.com/686ugd`, February 2007.

[12] B. Stock, M. Engelberth, F. Freiling, and T. Holz. Walowdac - Analysis of a Peer-to-Peer Botnet. In *EC2ND 2009: European Conference on Computer Network Defense*, November 2009.

[13] S. Stover, D. Dittrich, J. Hernandez, and S. Deitrich. Analysis of the Storm and Nugache Trojans - P2P is Here. *Login*, 32(6), December 2007.

[14] Sudosecure. Waledac Tracker. `http://www.sudosecure.net/waledac/bmd5updatecycle.php`.

[15] I. Sysoev. nginx. `http://nginx.net/`.