The Waledac Protocol: The How and Why

Greg Sinclair
*iDefense/
University of North
Carolina at Charlotte
gsinclair@idefense.com
gssincla@uncc.edu*

Chris Nunnery
*University of North
Carolina at Charlotte
cenunner@uncc.edu*

Brent ByungHoon Kang
*University of North
Carolina at Charlotte
bbkang@uncc.edu*

Abstract

Peer to Peer (P2P) botnets are a growing occurrence in the malware community. The Waledac botnet represents a new, more challenging trend in the P2P botnet evolution. The Waledac infrastructure has evolved key aspects of the P2P architecture and devolved others. This evolution/devolution has resulted in a more formidable botnet. As a result, the Waledac botnet is harder to infiltrate and harder to enumerate. This paper explains the various aspects of the Waledac botnet infrastructures to give defenders a better understanding of the botnet in order to protect themselves and others.

1. Introduction

Botnets continue to grow in their effectiveness and robust architecture. Traditional botnets use a single server or a small set of servers as the rallying point for all bots in the botnet. Defenders easily defeat this type of infrastructure by simply blocking these servers, typically referred to as the Command and Control (C&C) servers. In 2007, the Storm botnet demonstrated the effectiveness of decentralizing the C&C infrastructure to protect the botnet's viability. Decentralizing the C&C presents a challenge to defenders who can no longer take out a single set of servers to dismantle a botnet [1], [2].

In January of 2009, we began looking at the emerging threat known as Waledac. Waledac is a botnet that has striking similarities to the Storm botnet while at the same time exhibiting unique refinements that in part make the botnet more robust and in part vulnerable to attack. Through our research of the Waledac botnet, we were able to identify the communication schemes and network topology of the botnet. In late January 2009 [3], we were able to give defenders the necessary information to decipher the Waledac communication allowing the defender community to better understand the Waledac command structure. In addition to providing the necessary information, we released a decryption tool on a private security list to aid defenders in their decryption tasks. Antivirus vendors have used our research to understand the nature of Waledac [4].

In this paper we will provide a concise description of the Waledac architecture from a network topology and communication perspective. The Waledac architecture refines the Storm botnet's architecture to simplify aspects of the topology in order to increase efficiency and security. The revisions made by the Waledac infrastructure do contain vulnerabilities that defenders can exploit as we will explain.

The remaining sections of this paper explain how Waledac operates and defensive techniques which can be deployed by defenders to protect their infrastructure from the Waledac threat. Section 2 explains the architecture of Waledac from a network design and communication perspective. To aid defenders in protecting themselves and others from Waledac, Section 3 provides remediation strategies. This paper concludes in Section 4 by summarizing the previous sections and identifying future work in Section 5.

2. Waledac's Architecture

The Waledac architecture is peer-to-peer (P2P) based much the same way the infamous Storm botnet used P2P to decentralize the command and control (C&C) communication [5]. As we will present throughout this paper, the Waledac botnet has many similarities to the Storm botnet and at the same time has very distinct architecture from other known botnets. This section will analyze the Waledac architecture at the network and binary levels while making comparisons to the Storm botnet as a point of reference where applicable.

In order to understand the interaction between the Waledac bots and the Waledac botnet, the author of this paper employed both dynamic and static analysis of the Waledac bot binaries and their network communication. Since Waledac came to the attention of researchers in late 2008, the binaries produced by the Waledac author(s) have increased to several thousand known samples. These samples, however, are generally repacked variations of a much smaller sample size. Waledac uses an internal versioning system to identify the various core binary versions. As of August 2009, Waledac is currently at version 42. Using applications such as Patch Diff [6] or BinDiff [7], a comparison of the various Waledac versions indicates a very small, almost insignificant, change between most versions. The notable exception is the introduction of a new command ("creds") in version 34 as mentioned in section 2.2 and the increase in the node table and identifier size in version 36. Appendix A illustrates the timeline of the Waledac versions and major evolutionary changes with key versions.

To understand the Waledac architecture on the whole, it is necessary to break the Waledac system into three key areas: botnet hierarchy, botnet communication and infrastructure defense mechanisms. The remainder of these sections will explore each of these areas to give the reader a fuller understanding of the Waledac architecture.

2.1. Botnet Hierarchy

The Waledac infrastructure, as seen in Figure 1, is nearly identical in both topology and style to the Storm botnet. Unlike Storm, Waledac does not use Overnet/Kademlia [8] as a communication channel but instead utilizes HTTP communication and a fast-flux based DNS network exclusively. The basic topology of the Waledac botnet consists of many spammer (subnodes) nodes which exist behind a victim's firewall. Spammer nodes communicate exclusively with the next tier of bots known as repeater (supernode) nodes. Repeater nodes, in turn, marshal communication between not only repeater nodes and spammer nodes, but also between the repeater nodes and the next tier of nodes known as the TSL servers (subcontrollers). (TSL is the name of the Windows registry entry that the Waledac binary uses to store a list of servers for this tier. As such, we named the list of these servers as the TSL layer.) The TSL servers proxy data between the repeater nodes and the Upper-Tier Servers (UTS) which in turn communicates with the head-end C&C server. All of the communication between the various layers requires HTTP access and several layers of encoding and decoding.

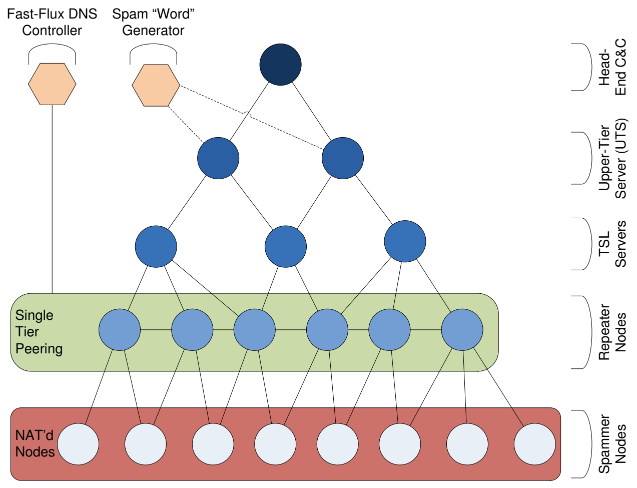

Figure 1: Waledac Hierarchical Topology

Infected victims of Waledac will run the binary in one of two forms: spammer mode or repeater mode. If the infected victim's IP address is a non-private IP address (e.g. not in the 192.168.0.0/16, 172.16.0.0/12 or 10.0.0.0/8 IP space), the Waledac bot will assume the position of a repeater node and will begin to proxy HTTP communication between the tier of bots immediately above it in the infrastructure (TSL servers) and the tier of bots immediately below itself (spammer bots). In addition, the repeater nodes will periodically communicate amongst themselves as explained later in this section.

Unlike repeater nodes, the TSL servers do not change frequently. Repeater nodes obtain the list of TSL servers' IP addresses simply by asking a fellow repeater node for the current TSL server list. This is in stark contrast to Storm which required subcontroller communication in order for supernodes to establish a relationship with the higher tier.

Waledac instantiates itself as a spammer node when the node finds itself unreachable by other nodes. The node makes this determination by the fact the IP address of the victim is a private IP. When operating as a spammer bot, the binary defines the node's primary functionality by the act of culling email addresses from the infected victim's machine and sending out directed spamming campaigns as orchestrated by the higher tier C&C server. In order to receive commands, spam templates, and targets for spamming attacks, the spammer nodes interact with the repeater bots which exist immediately above them in the network hierarchy.

At start up, both spammers and repeaters must establish an encrypted communication channel with the rest of the Waledac botnet. In order to establish the encrypted communication, a Waledac node must first

Comm-Type	Used For	Encoding Transform
1	Node List Updating via .php Page	Base64(AES.key2(XML))
2	Initial Key Exchange	Base64({header}{AES.key2(BZip2(XML))})
3	General Bot Communication	Base64({header}{AES.key0(BZip2(XML))})
4	Repeater Node List Updating using "X-Request-Kind-Code: nodes" HTTP Header	Base64(AES.key1(BZip2(XML)))
5	TSL Server List Updating using "X-Request-Kind-Code: servers" HTTP Header	Base64({header}{RSA/SHA1 Signature}{header}{Timestamp}{Entry Count}{IP:Port Pairs})

Table 1: Waledac Encoding Scheme

locate a neighboring repeater node. Much like Storm, each Waledac binary contains a list of IP addresses to use as a bootstrap list in order to make initial contact with the botnet. To provide additional resiliency, Waledac hardcodes a URL to access in the event a bot is unable to find an active node in the bootstrap list. The domain used for the URL is part of the fast flux network created by the botnet.

2.2. Botnet Communication

Once a neighboring node has been located, the bot sends an encrypted HTTP request to the neighboring node using a hardcoded password found in the Waledac binary. The transmission identifies the newly activated node by a 32 digit/16 byte or 40 digit/20 byte hexadecimal ID number (depending on the version of the binary) and sends the neighboring node (a repeater) a copy of the node's internal public certificate. The bot passes this certificate, which the binary generates at startup and is unique to each node, via the repeater through the botnet until it reaches the head-end C&C server. The C&C server uses the certificate to encrypt the current communication key required to interact with the botnet. Until July 20, 2009, this key was constant and the same for all nodes. On July 20, 2009, the head-end C&C server began changing the key at frequent intervals. The head-end C&C server passes the encrypted key back through the botnet to the repeater which relays the encrypted key to the node. Given that the key has been encrypted using the public key of the node, the key is inaccessible to those who intercept the communication. Once the node decrypts the key supplied by the head-end C&C server, any additional communication with the neighboring node uses this new password.

It is important to note that Waledac uses five different forms of communication encoding. As indicated in Table 1, all Waledac communication between nodes employs ASCII based Base64 encoding instead of using binary communication. We identified five types of communication encoding used by Waledac and gave these distinct traffic types "Comm-Type" identifiers of one through five.

One of the features of the HTTP communication that researchers found to be similar to Storm was the use of a very distinct URL format. HTTP communication between spammer nodes and repeater nodes and between repeater nodes and neighboring repeater nodes begins by a node issuing a request that takes the form of:

```
POST /<random>.<htm|png> HTTP/1.0
a=<Base64 Encoded Data
String>\&b=<Base64 Encoded Data
String>
```

The use of "a=" and "b=" with Base64 data is almost identical to the Storm HTTP communication. However, unlike Storm, Waledac uses a variety of different encoding schemes inside the Base64 data. In order to understand the meaning of these POST statements, we reverse engineered the Waledac binary to extract the encoding scheme used. We were able to find that the data contained within the "a=" string is translated into its original form by first decoding the Base64 string into binary. With the binary form of the data recovered, we identified that the first byte represents the type of command contained within the rest of the binary data. Following this are 4 bytes representing the size of the data once the node fully restores the data. After the data size, the remaining data is AES-128-CBC encrypted using one of two keys. The AES data is either encrypted using the static key used for the initial key exchange (known as AES.key2) or with the key recovered by the node after decrypting the response from the head-end C&C server during the key exchange (known as AES.key0). After we decrypted the AES data, the data stream represents a BZip2 compressed stream. Simply decompressing this BZip2 stream reveals plaintext XML formatted data. The "b=" parameter represents the IP address of the node submitting the request encoded in Base64 or the binary representation of the number zero if the IP is not required. Table 1 details this type of communication as comm-type 2 and 3 depending on the type of command. Key exchange, as presented in the example, is comm-type 2 traffic whereas all other command types (as described later) use comm-type 3. The difference between the comm-types, in this case, is the use of different AES keys.

Replies to the encrypted/encoded HTTP request from the repeater (or by proxy, the head-end C&C servers) are encoded in the exact same scheme as the original request with the exception that the reply is contained within the body of the reply and not as an "a=" or "b=" POST parameter. This symmetry allowed us to quickly determine both sides of the communication without additional work.

While repeaters are used primarily to move requests and replies between spammers and the head-end C&C server, repeater nodes also communicate between themselves in order to maintain lists of active nodes. Requests for .php pages and X-Request-Kind-Code pages are handled at the repeater node level and, as seen by experimentation, do not propagate upwards in the node tiers. While Storm nodes used Overnet protocols to provide advanced P2P routing information, Waledac uses a simpler, but surprisingly robust, solution for maintaining node lists.

Each repeater node maintains a list of available IP addresses that represent known and currently active repeater nodes known as the "node table". In order to keep stale IP addresses from taking up valuable room in the finite list of nodes which has a maximum capacity of 500 or 1000 entries (depending on the version of the binary), the binary gives each IP in the node list a timestamp. The use of timestamps allows newer nodes obtained through node-list updates to replace older nodes ensuring that the node list is as fresh as possible. When a node (a client node) uses another node (the server node) as a peer, the client node updates the timestamp entry for the server node to reflect the recent activity associated with the recipient node.

Obviously Waledac does not use repeater nodes exclusively for node updating. The Waledac botnet also uses repeater nodes to move data from the lower tier (spammers) to the upper tiers (TSL servers). Storm used the Overnet protocol to select the next peer to communicate with in order to provide a more wide reaching P2P architecture. Waledac uses random peer selection when choosing a peer with which to communicate. In addition to timestamps, each node assigns itself a 32 or 40 character (16 or 20 bytes, respectively) value which correlates to the internal identifier for a node. These identifiers are associated with the IP address of the node when a node list update occurs. Since Waledac is not using the Storm Distributed Hash Table (DHT) method for peer selection, these identifiers do not serve any tangible purpose for routing, but merely prevent the same node from occurring more than once in a node list.

The use of P2P architecture only truly exists at the repeater tier. Spammer nodes do not share the same P2P functionality due to their limited external connectivity. Similarly, the TSL servers appear to be primarily reverse proxies to the master proxy(s). The choice to restrict the P2P functionality to a single tier does allow the infrastructure of the botnet to make certain optimizations to reduce the overall traffic handling requirements for the higher tiers. The repeater nodes, as stated, handle the node update functionality for both the repeater and the spammer nodes. This autonomous behavior disconnects the overhead of node update traffic from the higher tiers. The repeaters, as it turns out, proxy the communication which must be handled by the head-end C&C server such as update requests, campaign notifications, and the other subset of commands supported by Waledac.

While it may not be immediately obvious, node lists at the spammer and repeater tiers represent the IP addresses of repeaters only. By virtue of the fact that spammer nodes are not Internet accessible, the choice to ignore the identities of spammer nodes provides a layer of obfuscation to the botnet. Researchers can easily identify repeater nodes by constantly querying and walking the repeater node list update mechanism. However, unless the researcher has managed to integrate themselves at a sufficiently high enough tier (typically TSL server tier or higher), the researcher will be unable to identify the vast majority of spammer nodes. If a researcher is able to proxy communication between the TSL server tier and the spammer tier, the researcher will only be able to record a small subset of the overall botnet. Unless researchers can develop a suitable sampling model, the true count of spammer nodes will remain known only to the botmaster. This is an advantage that Waledac has over Storm from a "numbers game" perspective. Storm's use of the Overnet protocol made enumeration of subnodes possible giving researchers the ability to quickly, and somewhat accurately, determine the size of the botnet. To date, no one has been able to concretely identify the size of the entire Waledac botnet.

Waledac's use of XML gives researchers the ability to view the command and control information in a human readable form, which allows researchers to quickly determine the intent of a command. While it is unclear why the author(s) of Waledac chose to use XML, it is possible that by using a standard format the developers could rely on open source packages to process the traffic more easily while at the same time giving the authors considerable flexibility with regards

Command Number	Command Name	Description
0 or 0xFF	getkey	Request used to request AES key1 for additional communication with node
1	first	Identifies the node's OS and the label associated with the binary
2	notify	Request instructions and configuration of spam campaign
3	taskreq	Requests a spam campaign configuration
4	words	Requests the meaning of variables in spam campaigns defined by *taskreq*
5	taskrep	Report campaign details statistics and email statuses
6	httpstats	(Repeater only) Reports internal HTTP access_log
7	emails	Report of all email addresses found on the victim's machine
8	creds	(Unconfirmed) Credentials found on the victim's machine (introduced in v. 34)

Table 2: Waledac Commands

to the data passed. The author(s) of Waledac relied on several open source packages such as TinyXML for the XML parser, OpenSSL for the cryptography and BZip2 for compression. Prior to version 15, which the author(s) introduced into the wild on November 28, 2008, Waledac used a strictly binary form of communication for bot requests while relying on XML for botnet replies.

When nodes in the Waledac bot communicate, the structure of the communication is symmetric. In other words, both the request and the reply contain common elements which tie the two sides of the conversation together. As a result, requests and replies both have a standard format as follows:

Request Format:

```
<lm><t>{command name}</t><v>{version of
node}</v><i>{hash ID of
node}</i><r>{repeater status: 0 or
1}</r><props><p n=...>{property fields
here}</props>{additional attributes
specific to the command type}</lm>
```

Reply Format:

```
<lm><v>{version of node}</v><t>{command
name}</t><props>// {additional properties
of the reply, if any}</props>{ additional
attributes specific to the command
type}</lm>
```

We identified 9 unique commands in the Waledac binary. While exploring the minutiae of each command would be space prohibitive, Table 2 lists each of the commands and their basic purpose while [3] provides significant detail of each command and its related format. As detailed previously, nodes identify request commands by a single 8-bit character in the POST request, but within the XML plaintext the binary gives each command an ASCII identifier. Each of these requests has a reciprocal reply. Replies use the same names as their request counterparts but differ slightly in their XML attributes. During our experimentation, we found that the head-end C&C ignores the 8-bit identifier in favor of the textual representation of the command. This means that if one gave the <t> field the value 'taskreq' but the 8-bit identifier was set to 1, the head-end C&C would reply with a taskreq response.

Waledac nodes use repeaters to move requests and replies between the various tiers of the topology. In order to reduce congestion at the higher tiers (TSL servers and higher), the repeaters have been programmed to handle many external requests associated with Waledac's propagation such as webpage serving and responding to fast-flux DNS queries.

From the HTTP perspective, repeaters act as a reverse proxy between the lower nodes (spammer nodes and adjacent repeater nodes) and the next tier, the TSL servers. The repeater node will immediately pass to the TSL server tier any command request (commands 0 through 8) from a lower node and marshal back the response received. Request for node list updates are not proxied but rather handled by the repeater itself resulting in a reduction in network traffic from unnecessary administrative overhead.

Waledac, like Storm before it, spreads through emails indicating that a news story has broken or that a loved one has sent the victim an e-card. These emails invariably point to a URL that is part of the Waledac fast-flux network. As a result, when a victim opens the link sent to them in an email in order to retrieve the story or e-card, the victim is interacting directly with a repeater node.

Repeaters do not handle HTTP traffic exclusively. When the upper tier of the botnet gives a repeater node the instruction to operate as a DNS server to support the Waledac fast-flux network, the repeater node will respond to DNS queries from both lower nodes and external IP addresses not associated with the botnet.

The DNS responses will return one or more repeater node IP addresses. Researchers can use this fact to get a rough enumeration of the repeater node tier, but may be less precise than walking the node tables via peer update requests.

The DNS configuration used by Waledac to support the botnet's fast-flux configuration is the result of coordination between the head-end C&C server and an external name server. Each repeater is capable of acting as a DNS server in order to hand out IP addresses for the botnet. In order for the repeater to understand which IP addresses to hand out for which domain, the head-end C&C server will reply to a repeater's 'notify' (command number 2) request with configuration information. The head-end C&C server will return a <dnszones> tag indicating which domain name or domain names the repeater will be responsible for serving. The tag also indicates which IP addresses to associate with the domain name or domain names.

In order for the repeater to act as an authorized name server, the top-tier C&C server will update an external domain registrar with the IP address of each repeater node acting as a DNS server. This event must occur after the upper tier of the botnet has given the repeater the DNS configuration information. It was observed on several occasions that the domain registrar's name server (NS) records were updated frequently which is characteristic of a fast-flux DNS configuration.

2.3. Waledac's Infrastructure Defenses

The linchpin in the Waledac topology is the connection between the repeater nodes and the TSL servers. It is this interface between the two tiers that researchers and defenders could exploit to cripple the Waledac infrastructure by severing the link between the lower tiers and their C&C masters. As a result, the author(s) of Waledac paid special attention to how nodes update the TSL IP list.

Waledac nodes will blindly accept any valid IP given to them for repeater nodes. The same is not true for TSL servers. Using comm-type 5 communication, a repeater node will ask another repeater node or one of the TSL servers for the current list of TSL servers. While the rest of the communication traffic in the Waledac infrastructure uses encryption, surprisingly the TSL update traffic does not. Instead the list contains two key defenses: a timestamp and a cryptographic signature.

The timestamp is arguably less of a security mechanism than a simple sanity check. The intent of the timestamp is to prevent old TSL IP updates from influencing newer TSL IP updates. This fact, however, provides protection from a replay attack against the TSL IP list. For instance, if a defender were to manage to take down all of the current TSL servers, conceivably the Waledac controller(s) would send a new list of IP addresses to the Waledac botnet in order to maintain the botnet's communication path. A defender who attempted to disable the botnet by injecting the old, now defunct, IP list back into the Waledac infrastructure would fail given that the timestamp would be older than the newer list. Therefore, the timestamp prevents a replay attack from succeeding.

The second, and most significant, defense found in the TSL is the use of a RSA cryptographic signature. Conceivably, researchers could easily avoid the replay attack defense mentioned previously by adjusting the timestamp to a date in the future. To prevent such an event, the author(s) of Waledac sign TSL IP updates using a RSA signature. The signature uses a private key held by the attacker(s) to sign the entire payload of the TSL update (excluding the signature portion, of course). Each Waledac binary contains a copy of the public key in order to verify that unauthorized entities have not altered the contents of the TSL update. If Waledac detects that the signature does not match the calculated signature value, the binary discards the TSL IP update. The use of a public/private key pair to sign the TSL update prevents defenders from inserting themselves into the TSL tier as well as prevents defenders from disrupting the upper tier communication.

As of August 2009, the author(s) of Waledac have used this defense functionality as an effective means to split the Waledac botnet. In late July 2009, the Waledac author(s) released a new binary (version 36) containing a new TSL signing certificate. As the population of new binaries increases, any new TSL updates sent by the author(s) will apply to either the old binaries (version before 36) or the newer binaries. This effectively separates the Waledac botnet at the repeater tier into the "old" and "new" botnets by simply altering the signing certificate of the TSL.

3. Defenses against Waledac

The use of fast-flux DNS makes IP blocking impractical as well. Despite these facts, we found several techniques to allow defenders to locate Waledac infections on their networks.

3.1. Detection of Waledac on Local Host

Throughout the course of our research, we found detection was simplistic. Unlike Storm which had the functionality to remain somewhat stealthy on an infected system, Waledac does not attempt to hide itself from the victim. The location where the binary was originally downloaded is the location where the binary will remain after each reboot of the victim's machine. Processes found running from binaries located on a victim's desktop or My Documents directory (the typical locations where defenders have found Waledac binaries) should raise immediate suspicion.

Over the last several months, Waledac has been instructing both spammer and repeater nodes to download and install Rogue A/V applications. These scareware applications immediately alert victims to their presence, another way in which Waledac does not remain stealthy after infection.

3.2. Enumeration of Infection Hosts

From a network perspective, defenders are encouraged to stay informed of the current Waledac domains used as part of the fast-flux network. Blocking these domains will help remediate some of the functionality of infected hosts, but this solution only mitigates a small portion of the Waledac topology.

Waledac's repeater node update system relies on HTTP traffic that contains the string "X-Request-Kind-Code: nodes" in clear-text. IDS/IPS systems should identify any network traffic containing this string and, for IPS systems, terminate the connection. This, in combination with blocking the fast-flux domains, can severely reduce the impact of Waledac infections.

3.3. Remediation of the Waledac Botnet

The Waledac infrastructure is resilient to disruption, but by no means is immune to a comprehensive attack. The Waledac infrastructure relies on three critical pieces in order to remain cohesive: TSL servers, repeater node tables and the fast-flux network. After looking at the Waledac botnet infrastructure in great detail, we have determined a probable attack against the botnet that will result in the collapse of the Waledac infrastructure.

The first step in dismantling the Waledac infrastructure requires injecting one or more fake ("sinkhole") Waledac nodes into the Waledac botnet at the repeater tier. These fake nodes would operate exactly like real Waledac nodes with the exception that any command request would result in the node returning a no-operation response to halt the malicious activity of the requesting bot and any request for a node list would contain a full list of IP/hash combinations that point to the fake node(s). Each hash in the supplied node list must be unique while the IPs may be identical. It is important to use hash values not currently present in the botnet. Using hash values that exist in the botnet will only reinforce the node entry associated with that hash since the Waledac binary inspects only the hash value when updating the timestamp of an entry, not the IP, when a hash exists in the node's local node table.

Once the defender has redirected the majority of the existing Waledac spammer and repeater nodes to communicate with the fake node(s), the defender can then, and only then, attack the fast-flux component of the network. As mentioned previously, the fast-flux domains hardcoded in each binary provide the nodes with a back-up method for obtaining valid node entries if the node's current node table were to expire. By systematically removing all of the domain names from the .com TLD zone, nodes will be unable to find valid nodes once the defender shuts down the fake node(s).

Finally, the last component of the Waledac botnet remediation plan requires a coordinated shutdown of all current TSL servers. If the servers are not shutdown by the ISPs at roughly the same time, the Waledac author(s) can simply issue a new TSL IP update to any nodes the defender has not corralled using the fake node(s).

Once these three attacks have concluded, the defender can shut down the fake node(s). After a period of time, the entries for the fake node(s) will expire from the remaining Waledac nodes' node tables due to the fake node(s) being unresponsive. After the nodes' node table empties, the nodes will attempt to contact the fast-flux domains in order to obtain a new peer list. With the fast-flux domains no longer active, the nodes will be unable to find valid repeater peers. Consequentially, repeater nodes will be unable to contact the TSLs or neighboring repeaters in order to obtain new TSL entries effectively cutting off the repeaters from the botnet.

The end result of this attack is the collapse of the Waledac infrastructure. We believe that the order and completeness of the attacks is critical to a successful shutdown of the Waledac infrastructure. If a defender were to only attack the node tables and TSL servers, the Waledac author(s) could recover the botnet by assuming the fast-flux domains and issuing new node table entries. Likewise, if the defender were to take-down the TSL servers before disabling the Waledac author(s)' access to the botnet, the author(s) could simply inject a new Waledac repeater node into the botnet to issue new TSL IP updates. It is imperative that the Waledac author(s) lose access to the repeater and spammer tiers prior to attacking the fast-flux or TSL components in order to prevent the author(s) from obtaining a foothold on the botnet which would allow them to defeat the attack.

4. Conclusions

In this paper we presented an analysis of the newly discovered Waledac botnet with respect to its topology, traffic characteristics, and command and control scheme. Throughout this analysis, we defined a definite protocol and architecture for Waledac, and we made comparisons to the former Storm botnet, which possesses a strikingly similar multi-tiered architecture and has exhibited similar behavior in its self-propagation campaigns through social-engineering enhanced spam emails as well as its role delegation to participating bots. The architectural metamorphosis between these two botnets is that of refinement and simplification. Once perhaps unnecessarily complex, the author(s) of Waledac have stripped the architecture of the Storm botnet of its Overnet-based protocol and given a single tier peer layer within its hierarchical form in the Waledac botnet. As often seen in initially over-engineered technologies, eventual optimization occurs. The goal in malware design is, of course, to achieve profitability while maintaining defenses at a level equal to or above the mitigation capabilities of the computer security knowledge base.

Whether or not the Storm and Waledac botnets are related is of little importance to the gained knowledge from their comparison and the malware trends they suggest. Critically, we can observe the hybrid hierarchical and peer-based networks common among the two networks. The mere fact that attackers replicated this architecture in the wild implies a limited number of possibilities: the developers of Waledac were familiar with nearly all of Storm's topological and behavioral characteristics and made a conscious decision to emulate them, the botnets share a common developer, or the architectures are similar by pure coincidence. From our analysis, we note a proclivity for botnets of this breed to employ bot nodes in aggressive and highly resilient fast-flux DNS functionality. We expect this usage to become widespread even among more moderately-engineered bot networks. Further, we can expect attackers to utilize more advanced private and public cryptographic schemes in botnets given the uses of this technology in the architecture discussed in this paper.

This study further progresses the understanding of these advanced networks by exposing the Waledac protocol.

5. Future Work

For our future work on the Waledac infrastructure, we have begun the process of working with various ISPs and fellow researchers in an effort to obtain more information about the TSL, UTS and head-end server configurations.

Additionally, we are working with various organizations in the malware community, including fellow academic researchers, to implement our Waledac remediation plan outlined in section 3.3.

6. References

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

[2] S. Stover, D. Dittrich, J. Hernandez, and S. Deitrich, "Analysis of the Storm and Nugache Trojans - P2P is Here," *Login;*, vol. 32, no. 6, Dec. 2007.

[3] G. Sinclair. (2009, Jan.) http://www.nnl-labs.com. [Online]. http://www.nnl-labs.com

[4] J. Baltazar, J. Costoya, and R. Flores. (2009) Infiltrating WALEDAC Botnet's Covert Operations. [Online]. http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/infiltrating_the_waledac_botnet_v2.pdf

[5] J. Stewart. (2008) Protocols and Encryption of the Storm Bot. http://www.blackhat.com/presentations/bh-usa-08/Stewart/BH_US_08_Stewart_Protocols_of_the_Storm.pdf.

[6] Tenable. Tenable Network Security. [Online]. http://cgi.tenablesecurity.com/tenable/patchdiff.php

[7] Zyanmics. BinDiff. [Online]. http://www.zynamics.com/bindiff.html

[8] E. Michelangeli. KadC - P2P library. [Online]. http://kadc.sourceforge.net/

7. Acknowledgments

We would like to acknowledge Joan Calvet's work in obtaining the most recent Waledac binaries.

A. Waledac Evolutionary Timetable

Waledac was original developed in December 2007. Since its original release on December 25, 2007, Waledac has released over 40 different versions (excluding repacks of the base binaries for each version). This section provides a timeline of major milestones in the evolution of Waledac since its original release along with known version release dates.

December 25, 2007 – Waledac author(s) release version 0 of the binary. This version uses limited XML communication. Requests take a binary form whereas replies contain XML.

January 9, 2008 – Version 2 released

January 26, 2008 – Version 6 released. This represents a period of rapid development and refinement of the Waledac binary.

February 27, 2008 – Version 7 released.

March 12, 2008 – Version 10 released.

March 28, 2008 – Version 11 released.

April 9, 2008 – Version 12 released.

April 30, 2008 – Version 13 released.

September 2008 – Storm is shutdown. Waledac development slows considerably.

November 8, 2008 – Version 14 released. This is the last version to use binary only requests.

November 26, 2008 – Version 15 released. This is the first version to use XML communication with full headers. Version 15 represents the "modern" version of Waledac.

November 28, 2008 – Version 16 released.

December 5, 2008 – Version 18 released.

December 8, 2008 – Version 19 released.

December 17, 2008 – Version 21 released.

December 18, 2008 – Version 22 released.

December 21, 2008 – Version 23 released.

December 23, 2008 – Version 25 released.

December 23, 2008 – Version 26 released four hours after version 25 release.

December 30, 2008 – Version 27 released.

December 31, 2008 – Version 28 released. This is the first version to garner the attention of the malware community at large.

January 5, 2009 – Version 30 released. The packer protecting the Waledac binary changes from the easily recoverable UPX packer to a custom packer. This is the first version to experience a rapid regeneration of the same version with different packers.

January 14, 2009 – Version 31 released.

January 29, 2009 – Version 32 released.

July 20, 2009 – AES.key0 changes for the first time. Key changes on 10 minute intervals.

July 20, 2009 – Waledac author(s) begin using six TSL servers. Prior to this the Waledac author(s) used five servers.

July 22, 2009 – AES.key0 rotations slow to roughly one to two hour intervals.

July 25, 2009 – Version 36 released. The Waledac botnet begins to split in two: those nodes running binaries prior to version 36 and those that run binary version 36.

July 30, 2009 – Version 39 released. The hash size increases from 32 bytes to 40 bytes. The size of the node table increases from 500 entries to 1000 entries.

July 31, 2009 – Version 40 released.