Malware Authors Don't Learn, and That's Good!

Joan Calvet, Carlton R. Davis
École Polytechnique de Montréal,
Montréal, QC, Canada
{joan.calvet, carlton.davis}@polymtl.ca

Pierre-Marc Bureau
ESET
San Diego, CA, U.S.A.
pbureau@eset.com

Abstract

The Waledac malware first appeared in November 2008, shortly after the Storm botnet became inactive. This malware is currently quite prominent and active. Its main propagation mechanism is via social engineering schemes which entice or trick users into downloading and executing the malware binaries. The Waledac malware differs significantly from the Storm malware. For example, unlike Storm, Waledac utilises strong cryptographic algorithms, such as AES and RSA with 128 and 1024-bit keys, respectively. There are however a number of design and implementation errors and weaknesses in the malware which makes it relatively easy to intercept, analyse and modify and even to replay Waledac's communication traffic. Interestingly, some of these design and implementation errors and weaknesses were also present in the Storm malware.

In this paper, we present the results of our analysis on Waledac. To facilitate our analysis, we captured several versions of the malware binaries and reverse engineered them. We also executed the binaries in secure environments and observed their communication traffic. Our analysis provides valuable insights into the inner working of Waledac malware and the botnet it constitutes. In addition to giving details of the mode of operation of Waledac, we highlight some of the weakness of Waledac, outline some of the differences and similarities between Waledac and Storm, and suggest means by which Waledac botnet can be infiltrated and disrupted.

1 Introduction

Waledac is recent prominent malware that first appeared on the Internet in November 2008, shortly after Storm botnet became inactive. As is the case for the Storm malware, the principal propagation mode for Waledac is social engineering schemes which entice or trick Internet users to download and execute the malware binaries. Additionally, Waledac has also been reported [12] to be downloaded and installed by other malware families, such as Conficker [9].

The functionalities in Waledac have remained fairly constant over time. The main changes are related to the malware protection layer, i.e. the packaging of the binaries, which is aimed at evading anti-malware detection software. The first variants of Waledac binaries were packed using the publicly available UPX packer [4]. This packer is well known and it is used by a number of legitimate software vendors; it offers very limited protection against reverse engineering and detection by anti-malware softwares. In later variants of the malware, the UPX packer was replaced with custom made packers and various anti-debugging and anti-emulation techniques incorporated into the binaries to slow down or thwart reverse engineering analyses. Like most modern malware families, Waledac is spread by waves or campaigns. Before each campaign, the malware packer is modified and apparently tested against known anti-malware software solutions to ensure that, upon release, the majority of the security software will not detect it. There are consequently several versions of Waledac malware whose difference is mainly attributed to the packing characteristics.

In this paper, we present the results of our analysis of Waledac. To facilitate our analysis, we captured several Waledac binaries and unpacked and reverse engineered the binaries. We also executed the binaries in secure environments and observed their mode of operations. The contributions of the paper can be summarised as follows:

1. We provide details of the mode of operation of Waledac. The information we present can be employed in the design of Waledac detection and mitigation schemes.

2. We identify weaknesses in the design and implementation of Waledac malware. These weaknesses can be exploited in Waledac countermeasure schemes.

3. We outline methodologies for two attacks that can be used to disrupt the Waledac botnet.

4. We highlight differences between Storm and Waledac.

The rest of the paper is structured as follows. In Section 2, we review previous works that are related to Waledac, and highlight how our work differs from these works. We present Waledac's technical features, in Section 3. We provide details of Waledac's communication structure and message types, in Section 4. In Section 5, we highlight some of Waledac communication weaknesses and outline methodologies whereby Waledac botnet could be infiltrated and disrupted. We compare and contrast Storm and Waledac, in Section 6. Finally, in Section 7, we summarise our findings and share some ideas for future work.

2 Related work

At the time of writing, we have found only two published works related to Waledac analysis. We indicate below, how our work differs from these previous works.

Lasse [7] presented in his M.Sc. thesis the results of a case study on Waledac. Some of the findings he presented are similarly to ours. However, our work differs from his in the following ways: (i) we provide greater details about the technical characteristics and the inner working of Waledac, (ii) we outline practical schemes for attacking and disrupting the Waledac botnet, and (iii) unlike Lasse's thesis, our work highlights differences between Waledac and Storm.

Wu, Kink and Molenkamp [12] presented a case study of Waledac which gives a time-line of Waledac's activities, and provide details regarding the number of machines that have been infected by Waledac, in selected countries. The emphasis of their work is different from ours, in that it does not involved reverse engineering analysis of the code.

3 Waledac technical features

Waledac malware binaries are coded in C++. Our analysis of the compiled code suggests that the malware codes were compiled with Microsoft Visual C++ compiler. The size of the binary files are over 860 KB; this is larger than the average size of malware binaries. When Waledac malware executes for the first time, it modifies the Windows registry by adding an entry to the `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run` key. This ensures that the malware code is executed every time the machine starts.

Our reverse engineering analysis revealed that Waledac malware has the following features.

Mail engine. The core binaries of Waledac all contain a SMTP engine which is capable of communicating with a SMTP server and sending emails. Thus, it appears that the main objective of Waledac is to send spam messages.

Strong cryptography. Waledac binaries are statically linked with a compiled version of the OpenSSL library that has `OpenSSL 0.9.8e 23 Feb 2007` as its version tag. The malware also appears to utilise the CryptoLib library for cryptographic services. The binaries are hardcoded with two 128-bit AES keys and a X.509 certificate with a 1024-bit RSA public key; an additional RSA public key and a private key is generated at runtime when the binaries first execute. We give details regarding the usage of these keys in Section 4.

Basic HTTP proxying. Waledac binaries contain code to proxy HTTP traffic. The malware can handle two types of HTTP traffic: the control messages to the command and control (C&C) servers and the "normal" HTTP traffic to or from other Waledac peers.

Double fast flux DNS. Every Waledac domain name is attached to a set of infected machines (which are in fact only proxys for the real Waledac servers) and the DNS servers for these domain names are also interchanged on different machines. This technique is often utilised by malware to hide the true identities of proxies and servers [11].

Peer-to-peer communications. Waledac uses a fairly simple custom-made peer-to-peer (P2P) communication protocol that involves locally maintained peer-lists which are constantly updated.

Email addresses harvest. Certain versions of the Waledac malware have a module which scans the hard drive of an infected machine in attempts to harvest email addresses. A list of file extensions is embedded within the binary (example, *.avi,.mp3,.mp4 and .7z*) and the malware scans every file whose extension is not listed in the list, to find email addresses. To decrease the likehood of the scanning activities being detected, the malware utilises the following simple load balancing mechanism: it records the time it takes to read 200,000 bytes, if this time is greater than a hardcoded value, it waits for a specified time period before scanning other bytes.

Password theft. Waledac uses the WinPCap library, which is downloaded on a newly infected computer, to sniff all the network traffic, in attempts to find HTTP, FTP and SMTP passwords. As indicated in Section 4.2, the malware then sends the information it harvests to the C&C servers.

4 Waledac communication infrastructure

Our analysis of Waledac malware reveals that Waledac botnet involves at least 4 layers (see Figure 1). In the description below, we label the layers based on the role they play.

Spammers. These are the worker machines: they do the basic jobs such as sending spams and performing DoS attacks. They are typically Microsoft Windows machines with private IP addresses. They do not know the identity information of other spammers: they are only aware of Repeaters that are in their peer lists. Spammers constantly query the

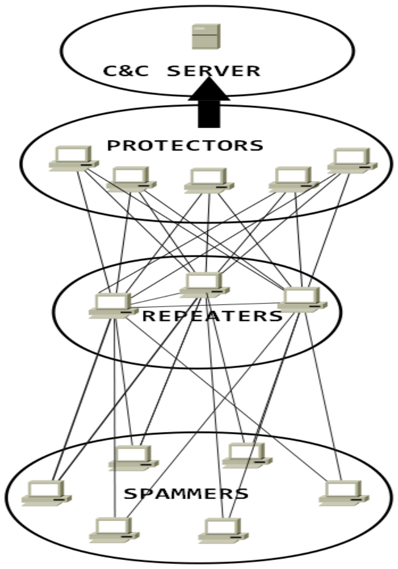

Figure 1. Architecture of the Waledac botnet

Repeaters that are in their peer lists to get orders for jobs they are required to perform.

Repeaters. These are machines with public IP addresses. Their purpose is to relay the C&C traffic between the spammers and the Protector layer. They also serve as HTTP proxies for the Waledac websites, and as DNS servers for the Waledac double fast flux system. Repeaters communicate with Protectors and other Repeaters but they cannot contact Spammers directly. If the Windows Firewall is running on an infected machine which is designated as a Repeater, the malware modifies the firewall setting to ensure that it can perform its Repeater roles without hindrance.

Protectors. These are Linux servers running different versions of nginx, a light HTTP proxy. Through our analysis, we have identified 5 machines in this layer. Interestingly, the identities of these machines have not changed over the pass several months. The location of these machines are as follows: two are located in Germany, one in USA, one in Russia and one in Netherlands. The IP addresses of these Protectors are owned by classic web hosters; this suggests that bullet-proof hosting may be involved. Initially, it appeared as if these machines were the actual C&C servers. However, during our analysis, we observed that the field *Date* of their HTTP responses have the same value at the same moment; which implies that the five servers (which are located in two different continents) are either time synchronised or they act as proxies for another server. We assume that it is the latter.

C&C server. We believe that the servers in the Protector layer act as proxies for the server(s) in this uppermost layer.

A newly infected machine is either designated as a Spammer or as a Repeater, depending on whether it has a public IP address or not. We have discovered that the malware binaries accept arguments to force the role: *-r* for Repeater and *-s* for Spammer. We have not observed this “role forcing” in the wild though; consequently, we conclude that it is perhaps a debug feature.

4.1 Peer-to-peer protocol

Waledac utilises a simple P2P protocol where each bot in the botnet keeps what we termed as a Repeater-list (*RList*) which contains the identification information of Repeaters that the given bot is allowed to communicate with. In addition to a RList, a Repeater also keep a Protector-list which contains identity information of Protectors. The RLists and the Protector-lists are updated as indicated below.

4.1.1 Update of Repeater-lists

All Waledac binaries come with a hardcoded list that contains approximately 200 Repeaters. The content of the list is different for each binary we examined. The RList is an XML file which is stored in a registry key. The file is compressed using the bz2 algorithm, then encrypted using a 128-bit AES key K_1, which is hardcoded in all Waledac binaries. The file has the following form:

```
<lm>
<localtime>1244053204</localtime>
<nodes>
<node ip="a.b.c.d" port="80" time="124..204">
469abea004710c1ac0022489cef03183
</node>
<node ip="e.f.g.h" port="80" time="124..532">
691775154c03424d9f12c17fdf4b640b
</node>
...
</nodes>
</lm>
```

Each entry in the file contains an IP address, a port number (80), a global UNIX timestamp and a random 16-byte ID, which is an unique identifier of the Repeater whose identification information is contained in the entry. RLists are ordered by timestamps: the entry with the most recent timestamp is in the first position. Waledac uses TinyXML (a C++ XML parser), which is embedded in the malware binary, to process the XML file. RLists are updated in the following two ways:

1. **Constant sharing with other peers**. All the bots regularly contact Repeaters that are in their Rlist, to get updates. The choice of Repeaters to contact is chosen randomly. The Repeaters that are selected are sent a list containing the identity information of 100 Repeaters. This list is extracted from the given bot’s RList. When a bot B receives an RList extract from

another bot S, B extracts the identity information of 100 Repeaters from its RList, randomly and sends the sublist to the bot. If B is a Repeater it only extracts 99 and puts its identity information at the first position of the sublist before sending the sublist to S. This provides a mean for bots to propagate their identity information in the botnet. The bot B uses the information it received from S to update its RList as follows. First, B computes a timestamp *TSCurrent*, and then, uses it to assign new timestamps NewTS_i for the entries in the sublist it received from S, using the formula

$$\mathit{NewTS}_i = \mathit{TSCurrent} - (\mathit{UpdateTS} - \mathit{TS}_i)$$

where *UPdateTS* is the global timestamp on the sublist received from S, and TS_i is the current value in the timestamp field of the i-th entry. The entries with the newly assigned timestamps are then inserted in the RList at the appropriated positions (recall that the RLists are sorted by timestamps). Also, it should be noted that the maximum number of entries in a RList is 500; entries that are in excess of 500 are discarded.

2. **Connection to a website**. All the Waledac related domain names have a special page (for the domain we observed, it is linked as *index.php*) which contains an encrypted version of an RList, with approximately 180 Repeaters. The encryption is done with a 128-bit AES key K_2 which is hardcoded in each binary. This list is automatically updated every 10 minutes; we believe it contains the Repeaters that most recently contacted the C&C servers. When a bot makes 10 consecutive failed attempts to contact Repeaters, it attempts to connect to one of the Waledac websites and gets updates. The URLs for a list of Waledac websites are hardcoded in each binary. These websites have very good availability, owing to the double fast flux DNS techniques the botnet employs. This mechanism therefore provides an effective mean through which old versions of the Waledac malware can be updated. The updates the bots receive have the same form as the RList. When a bot receives an update, it assigns new timestamps to the entries in the update using the formula above, and it inserts the entries in its RList. It then prunes the RList by removing entries that are at position 501 or greater.

4.1.2 Update of Protectors-lists

The Repeaters need to have a list of the Protectors so that they can relay C&C traffic to and from the Protectors to the other bots. Waledac binaries that are designated for Repeater bots therefore contain a base64 encoded Protector-list, which is signed with the RSA private key that is associated with the public key embedded in the binary. Each entry in the list contains a UNIX timestamp and the IP address of a Protector. As is the case for RLists, Protector-lists are also sorted by timestamps. Repeaters keep their Protector-list updated by exchanging Protector information regularly. A Repeater update process consists of two phases: in the first phase, the Repeater updates its RList; and in the second phase, it updates its Protector-list. The two types of update messages are distinguishable by a custom field called `X-Request-Kind-Code` in the HTTP header, which takes the value *"nodes"* for the Repeaters update messages and *"servers"* for Protectors update messages.

4.2 Message types

Waledac bots communicate using HTTP messages. The data field of the messages are bz2 compressed, encrypted with a 128-bit AES key, and base64 encoded. The messages have the following format:

```
POST /jyl.png HTTP/1.1
Referer: Mozilla
Accept: */*
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla
nHost: A.B.C.D
Content-Length: Y
Cache-Control: no-cache

a=BwAAC3Jrvsqwur_.....jBJlP7cpO
NrERG-c7uHr-&b=AAAAAA
```

The image file name (jyl.png) appears to be randomly selected; the `Host` field contains the IP address of the destination bot. The data field is the block behind the `a` parameter; and the block behind `b` parameter is the base64 encoded IP address of the sender of the message. Actually, if the sender is a Spammer, we observed that the value of this field is always `AAAAAA`. When a Repeater receives a message from a Spammer, the Repeater adds the field `Client-Host` to the message, fills it with the IP address of the Spammer, and forward the message to a Protector. Here is a brief description of the format of the data field for different message types.

getkey: This is the first message a bot sends when it initiates a dialog with another bot. The message is encrypted with AES key K_2. The plain text message has the following format:

```
<lm>
<t>getkey</t>
<v>34</v>
<i>004aee5a614fc47617439d64bb42c01d</i>
<r>0</r>
<props>
<p n="cert">
-----BEGIN CERTIFICATE-----
MIIBvjCCASegAwIBAgIBADANBgkqhk...
```

```
...BAwAqe1u3s68csfHQOSicrtvvfJMXt
-----END CERTIFICATE-----
</p>
</props>
</lm>
```

The <v> tag contains the version of the Waledac binary and <i> contains the ID of the bot that sends the message. The <r> parameter is set to 0 if the bot sending the message is a Spammer, and to 1 if it is a Repeater. The message contains the X.509 certificate with a 1024-bit RSA public key, which the bot (the sender) generated the first time it ran. The certificate is set to expire one year after it is generated. The response of the C&C server has the following form:

```
<lm>
<v>34</v>
<t>getkey</t>
<props>
<p n="key">UvqIPaS.....TT5X5hfJNNLoac=
</p>
</props>
</lm>
```

The `key` contains a 128-bit AES key K_3 that is encrypted and base64 encoded. The encryption is done with the 1024-bit RSA public key sent in the getkey message. It appears that K_3 was meant to be a session key since all messages that follow the getkey message are encrypted with it. There appears to have been a programming error on the part of Waledac malware developers, as *the session key K_3 is always the same*.

For the next three message types, we only describe the <props> field because the other fields are the same as for other message types.

first: This message is send just once by a newly infected machine. The likely purpose of this message is to provide a count of the number of Waledac bots. The <props> field of this message has the following format:

```
...
<props>
<p n="label">mirabella_site</p>
<p n="winver">5.1.2600</p>
</props>
...
```

The variable `label` indicates the origin of the infection. Some of the values we have seen for this variable includes: *tty*, *tty2msn*, *twist*, *birdie3* and *ub*. Wu *et al.* [12] suggest that the *twist* value indicates that the Waledac binary was downloaded by the Conficker malware [10]. The <winver> field contains the infected host's Microsoft Windows OS version.

The response to the *first* message is an acknowledge from the server; it is essentially an empty *first* message. When a bot receives the response, it creates a registry key named `FWDone` with value set to *true*, indicating the `first` message was sent and needs not to be sent it again.

notify: This message contains some timing information.

```
...
<props>
<p n="label">mirabella_site</p>
<p n="time_init">Tue May 05 20:28:35 2009</p>
<p n="time_now">Tue May 05 20:39:24 2009</p>
<p n="time_sys">Tue May 05 20:39:24 2009</p>
<p n="time_ticks">7097225</p>
</props>
...
```

The content of the response depends on the role of the bot. We indicate below, the relevant fields of the response message, starting with the portion of the message that is present irrespective of the role of the bot.

```
...
<props>
<p n="ptr">abcd.domain.com</p>
<p n="ip">A.B.C.D</p>
<p n="dns_ip">E.F.G.H</p>
<p n="smtp_ip">I.J.K.L</p>
<p n="http_cache_timeout">3600</p>
 <p n="sender_threads">13</p>
<p n="sender_queue">2000</p>
<p n="short_logs">true</p>
<p n="commands"><![CDATA[
341|download|http://abcd.com/win.jpg
341|downloadexe|http://xyz.com/n1.exe]]>
</p>
</props>
...
```

The value of the relevant fields are indicated below.

- `ptr`: Indicates the domain name that is assigned to the IP address of the bot. It will appear in the header of the spam messages send by the bot.
- `ip`: Shows the IP address of the bot.
- `dns_ip`: Indicates the IP address of a DNS server the bot will use to, for example, determine the IP addresses of target SMTP servers.
- `smtp_ip`: This is the IP address of a SMTP server that will be used as a test server. This is often the address of a Google server.
- some technical parameters, e.g. `sender_threads`, that represents the number of threads the mail process will use when sending spams. We observed values which ranged from 13 to 50.
- `commands`: This contains commands to be executed on the bot. The commands have the following form: `ID|action|URL`, where:

- `ID` is a number associated with the command and it is stored in a registry key named `LastCommandID`. When a bot receives a command, it only executes it if the ID received is strictly greater than the value in the registry key.
- `action` can take several values: *update, download, downloadR, downloadS, updateexe, downloadexe, downloadrexe, downloadsexe*. We observed that for the *download* command, the URL often points to a JPG image which contains a hidden executable. The hidden executable is a modified installer for the WinPCap library, which is used to steal passwords, as indicated in Section 3.
- `URL` points to the target of the action.

For a *Repeater*, the rest of the response to a notify message has the following form:

```
<dns_zones>
<zone>^.*abc\.com$</zone>
...
<zone>^.*xyz\.com$</zone>
</dns_zones>
<dns_hosts>
<host>A.B.C.D</host>
...
<host>W.X.Y.Z</host>
</dns_hosts>
<socks5>
<allow max_conn="100">E.F.G.H</allow>
...
<allow max_conn="100">R.S.T.U</allow>
</socks5>
<dos></dos>
<filter>
<deny>I.J.K.L</deny>
...
<deny>M.N.O.P</deny>
</filter>
```

The important tags and their role are indicated below.

- `zone`: These are the domain names which will be used in the spam messages. As indicated in Section 3, the domain names are maintained with the double fast flux DNS technique.
- `host`: This field contains the IP addresses of Repeaters which also play the role of HTTP proxy for the visitors to the `zone` domain names.
- `socks5`: These are Linux servers running nginx, they appears to be "motherships" for the double fast flux DNS technique. During our experimentation, we observed that these Linux servers also sent DNS queries on behalf of the `zone` domain names.
- `deny`: This tag contains thousands of IP addresses (perhaps about 5,000). Requests for HTTP forwarding that originated from any of these IP addresses are denied.

When a Repeater receives the response to a *notify* message, it tests the connectivity with all the hosts in the `host` field to ensure that they are "alive". Then in the future, when it receives a DNS query for one of the `zone` domain names, the Repeater responds to the query with one of the "live" hosts.

For a *Spammer*, the rest of response to a notify message is only the closing of all the tags, unless the Spammer is being requested to perform a DoS attack.

emails: This message is send regularly by both Repeaters and Spammers, it contains all the emails addresses harvested from infected machines.

taskreq: This request is send by Spammers only if they have contacted the SMTP server whose IP address is contained in the `smtp_ip` tag of the response to a notify message. If the Spammer failed to contact the SMTP server, until it receives an IP address of a SMTP server it is able to contact, its main purpose is to harvest email addresses and steal relevant information from the infected machine. The response to a taskreq message has the following form:

```
...
<tasks>
<task id="4">
<body>UmVjZWl2ZWQ6IChxbWFpbCAlXl...
....2ZWQdawdwadhcm1hX2xpbmtzXiUvXiU
</body>
<a>abcd@abcd.com</a>
...
<a>wxyz@wxyz.com</a>
<w>charset</w>
<w>domains</w>
<w>mynames</w>
...
<w>outver.6</w>
<w>pharma_links</w>
</task>
</tasks>
<words>
<w name="domains" time="1241467326"/>
<w name="names" time="1241467203"/>
<w name="charset" time="1233919243"/>
...
<w name="mynames" time="1233919245"/>
</words>
...
```

This response contains the following parts:

- `body`: The template for the *body* tag is base64 encoded. An example of the plain text form is indicated below.

```
From: "%^Fmynames^% %^Fsurnames^%"
<%^Fnames^%@%^Fdomains^%>
To: <%^0^%>
Subject: %^Fpharma^%
```

```
Date: %^D^%
MIME-Version: 1.0
Content-Type: text/plain;
format=flowed;
charset="%^Fcharset^%";
reply-type=original
Content-Transfer-Encoding: 7bit
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express
         %^C7%^Foutver.5^%^%

%^J%^Fpharma^% http://%^P%^R2-6^%:..
.%^Fpharma_links^%/^%
```

This template contains many parameters, e.g. *my-names, surnames, pharma*, which a bot will fill before sending spams.

- `a`: This tag contains the target addresses.
- `w` and `name=`: The `w` tags contains the parameters of the *taskreq* response template; the `w` and `name=` labels contain the timestamps of the parameters. The Spammer will save the values of these parameters if the timestamps are more recent than those of the previously saved values.
- `id`: The `id` parameter of the `task` tag serves to identify different spam campaigns. In one taskreq response, a Spammer can be instructed to participate in several campaigns, each of them having its own `body` and targets.

Once a Spammer receives the complete response to a *taskreq* message, it begins sending spam.

taskrep: When a spammer completes it's spamming task, the Spammer sends a report that contains all the email addresses it spammed, indicating also which email adddresses are functional and which are not. The report is encapsulated in a *taskrep* message. The message has the following format:

```
<reports>
<rep id="4" rcpt="YXJub...l4LmNvbQ==">
RVJS</rep>
<rep id="4" rcpt="bi5uY...UBtZWRpY==">
T0s=</rep>
...
</reports>
```

The `rcpt` parameter contains a target email address; `RVJS` and `T0s=` are base64 encoding of "ERR" and "OK", respectively.

httpstats: When a Repeater forwards a HTTP request for the Waledac web servers, the Repeater logs relevant information about the visitor and encapsulate the logs in *httpstats* messages and sends the messages to the top servers at regular intervals. An example of a *httpstats* message is indicated below.

```
<http_stats>
<stat time="1242929383" ip="a.b.c.d">
<![CDATA[GET /index.php HTTP/1.1  Mozilla]]>
</stat>
...
<stat time="1242929397" ip="w.x.y.z">
<![CDATA[GET /index.php HTTP/1.0  Mozilla]]>
</stat>
</http_stats>
```

creds: When a bot captures passwords and login information on a network, the bot sends the information in a form directly usable, such as, *ftp://login:password@ftp.vulnerableHost.com*, to a top level server.

5 Communication weaknesses

Some of the weaknesses of Waledac communication schemes we present in this section have already been identified by Lasse in his M.Sc. thesis [7]; however, we provide greater details on how the weaknesses can be exploited. We commence by describing a man-in-the-middle attack we launched against Waledec botnet, which allowed us to read the plain text form of Waledac communications and inject relevant messages in the communication traffic. Next, we outline how Sybil attacks can be launched against Waledac.

5.1 Man-in-the-middle attacks

As indicated previously, Waledac uses three 128-bit AES keys. K_1 and K_2 are hardcoded in the binary, and K_3 is the "session" key that always the same value! We found the value for all three keys; therefore it was easy for us to decrypt the Waledac communication traffic. It should be noted that there is a Waledac decoder available on the Internet [8] that allows one to compute the plain text form of an encrypted message. However, owing to our knowledge of the internals of Waledac, we were able to code a fake Repeater in Python that performs two of the main functions of a real Repeater, namely the following:

- **Send and receive update messages**: As indicated in Section 4.1.1, bots update their RLists by sending and receiving update messages that contain identification information for 100 Repeaters. Our fake Repeater succeeded in pushing update messages —with its own address in first position— to real Waledac bots. This allows the identification information of our fake Repeater to be propagated in the botnet.

- **Proxying the C&C traffic**: We succeeded in using our fake Repeater to proxy C&C traffic to the Protectors; and we were also able to log the plain text form of the traffic "on the fly".

With our fake Repeater bot, we were able to easily monitor the Waledac botnet. Consequently, each time the Waledac C&C servers launched a new order, our fake Repeater informed us and we are able to quickly analyse the order. For example, when bots are instructed to download new binaries, as the bots download the new binaries using the Repeaters as proxy for the Waledac webserver, and as the Repeaters send HTTP reports, our fake Repeater is asked to proxy some of the reports to the Protectors. This provides us with instant access to a wealth of "sensitive" Waledac information.

In addition to spying on the botnet and gathering relevant information, another possible use of the man-in-the-middle attack we illustrated above, is to inject traffic into the botnet which can be used to alert the owner of the machine that the machine is infected. It is conceivable that the bots could also be instructed to download code to disinfect them or just to deactivate the malware code.

5.2 Vulnerability to Sybil attacks

Waledac bots are identified by a 16-byte ID and an IP address. The IP address does not have to be unique. Consequently, thousands of sybils (fake nodes) with unique IDs can be generated from a single machine and used to infiltrate the botnet.

A methodology through which Sybil attacks could be mounted against Waledac is as follows. One could run the sybils as *super Repeaters* which can infiltrate the botnet by means of the man-in-the-middle attack we outlined above. The aim is to get the identity information of the super Repeaters into as many RLists as possible. The process can be facilitated relatively easy owing to design flaws in the Waledac P2P protocol. For example, we indicated in Section 4.1.1 that bots update their RList by sending and receiving update messages which contain identity information of subsets of 100 Repeaters. We discovered that when a bot receives an update message, it does not check to determine the number of records that are in the update message. It is therefore possible that when the super Repeater gains access to the botnet, it can send updates messages that contain identity information of 500 other super-Repeaters, with the timestamps of the entries selected such that they are guaranteed to be more recently than the timespam of any entry in the RLists of the bots. Note that 500 is chosen because that is maximum number of entries that are allowed in a RList. Consider the sample message below.

```
<lm>
<localtime>600</localtime>
<nodes>
<node ip="ourIPaddress" port="80"time="599">
00000000000000000000000000000001</node>
<node ip="ourIPaddress" port="80" time="599">
00000000000000000000000000000002</node>
...
<node ip="ourIPaddress" port="80" time="599">
00000000000000000000000000000500</node>
</nodes>
</lm>
```

When a bot receives this message, it is likely that all the entries in the bot's RList will be replaced by the entries in the update message, since the difference of 1 between the local timestamp (599) and the global one (600) ensures that sybil entries will be assigned more recent timestamps than the entries that are currently in the bot's *RList*, as described in Section 4.1.1.

The extent to which a bot can be controlled and be isolated from other bots when the bot receives such an update message from a sybil, depends on the type of bot.

- *If the bot is a Repeater*, after it receives the message from the sybil, there is a race condition situation, since the Repeater's identity information is most likely to also be in other bots' RLists. Consequently, other bots will likely send the Repeater update messages, whose entries can replace some of the sybil entries in the Repeater's RList. To maximize the chance that a Repeater will eventually be isolated from other bots after it received update messages from sybils, the sybils need to continue to send the Repeater update messages at short time intervals, for a given period of time, i.e. until the Repeater's identity information is flushed from all the other bots' RLists.

- *If the bot is a Spammer*, the result of the Sybil attack is more effective: since Spammers cannot be contacted directly, when a Spammer receives an update from a sybil, and the entries in its RList are consequently replaced by identity information of sybils, the Spammer will be completely isolated from other bots. It should be noted though, that in order to infiltrate a Spammer's RList, the sybils first need to infiltrate the RLists of Repeaters whose identity information are in the RList of the Spammer.

6 Storm and Waledac, Same Operation?

The Storm botnet is one of the most studied and well known botnets. The Storm malware (also known as Nuwar, Peacomm, and Zhelatin) infected tens of thousands of computers on the Internet [3, 5]. Storm became inactive in October 2008 after the California-based ISP which apparently hosted most of Storm's principal bot servers was shutdown [6]. Storm's main propagation scheme was through links to malicious files embedded in emails. It used themes like Valentine's day, Christmas greetings cards, video codecs and news about nuclear attacks [1].

It is hard to establish a link between Waledac and other malware families (such as Storm), by only examining the malwares binary files. On the other hand, if we take a high level view of the events, we quickly realize that the following characteristics are very similar in both:

1. The spreading mechanisms show a common prevalence of social engineering techniques and downloading by other malware families, probably on a pay-per-install business model. Moreover, the social engineering schemes which are used to entice users into downloading and executing the malicious files, uses almost the same themes.

2. Both Storm and Waledac use P2P communication to distribute information in the botnet.

3. Both malware families use fast flux infrastructure to spread new binaries and attract victims.

4. Both Storm and Waledac adopted information theft operations, but not immediately after their initial release on the Internet.

On the other hand, we have noted the following differences between the Storm and the Waledac family:

1. Storm used a separate spam engine, while Waledac always carries an embedded engine to send spam.

2. Storm used an existing C library implementing the well known Kademlia P2P protocol, while Waledac created and used its own P2P protocol.

3. The cryptographic library found in Waledac is very robust and it is compiled from publicly available code; while Storm's cryptographic functionalities were home made.

4. Multiple variants of Storm had a system driver with rootkit capabilities to hide its presence on an infected machine. To the best of our knowledge, Waledac does not have such functionalities.

5. Waledac's architecture includes more countermeasures for protecting the C&C and proxy servers, such as double vs. single fast flux and an extra layer of protector proxies.

7 Conclusions and future work

By looking at the differences and similarities between the Waledac and Storm malware families, it seems obvious that their operation is very similar while the malware files are different. This observation leads us to believe that these two malware operations may have been sponsored and organized by the same individuals. On the other hand, the difference in programming language, the similarity in the design flaws and errors in the code of both malware, and the different approaches to similar problems, leads us to believe that the programmers were different. Furthermore, it appears that the programmers did not study Storm's code or learned from Storm's design and implementation flaws, since Waledac's codes contains errors similar to those that appeared in Storm's codes.

Nonetheless, the C&C architecture of Waledac does show some signs of increased sophistication and care about stealth and redundancy. Such signs include the additional layers of protection in the network architecture and the use of double vs. single fast flux DNS. These changes might have indeed been triggered by lessons learned from the Storm botnet and its demise.

The use of custom-made P2P network instead of Kademlia also raises an interesting point. The design and implementation errors mentioned illustrate that designing and implementing good P2P protocols is not a trivial thing. In fact we observe a similar phenomenon to that found for proprietary cryptograpy: the software industry (including Waledac programmers) has mostly learned that correctly designing and implementing such protocols is best left to experts and should not be attempted in-house.

This realisation is even more significant in the light of what we discovered in previous work [2]: it would seem that structured P2P protocols such as Kademlia offer the best balance of robustness and efficiency vs. resilience against attack. Either the botnet operators have not read our papers (!) and were not conscious of this fact, or, their performance objectives have shifted and are different than what we had anticipated. For example, a botnet operator out to make a quick profit might not be so concerned about long-term resilience of the network, since they know that ultimately their very visible botnet will be found out, studied to death and eventually shut down because of its visibility. In that case, they might have consciously traded survivability against attacks for shorter-term goals such as efficiency, i.e. the ability to quickly reach and give commands to as many bots as possible. In that sense, the Waledac P2P protocol should not be completely discarded and considered as a mere failed attempt at building a "proprietary" P2P protocol. Once the obvious implementation errors fixed, e.g. trusting foreign timestamps blindly, the protocol shows characteristics that resemble other unstructured P2P protocols such as Gnutella. This might have been a coincidence or not.

On the other hand, this same tradeoff of resilience vs. efficiency might not be of interest for botnets used for espionnage, subversive activities or other political or longer-term economical agendas. Hence, we contend that independently of the immediate evolution of the more visible and common crime-oriented botnets, we should continue to comparatively study the various P2P protocols, in order to try to determine which of their characteristics affect botnet resilience and efficiency. This knowledge will prepare us for the time when botnet operators and their programmers do finally start to learn... Furthermore, it will put us in a position to start understanding which attacks against botnets work best against this ultimate, optimised botnet threat.

References

[1] P.-M. Bureau. Les changements climatiques et les logiciels malicieux. *MISC Magazine*, June 2008.

[2] C. Davis, S. Neville, J. Fernandez, J.-M. Robert, and J. McHugh. Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures? In *Proceedings of the* 13^{th} *European Symposium on Research in Computer Security (ESORICS'08)*, October 2008.

[3] D. Fisher. Storm, nugache lead dangerous new botnet barrage. SearchSecurity.com, December 2007.

[4] M. F.X.J., Oberhumer, L. Molnr, and J. F. Reiser. Ultimate packer for executables. `http://upx.sourceforge.net`, 2008.

[5] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proceedings of the* 1^{st} *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*, April 2008.

[6] B. Krebs. Atrivo shutdown hastened demise of storm worm. `http://voices.washingtonpost.com/securityfix/2008/10`, October 2008.

[7] T. B. Lasse. Peer-to-peer botnets: A case study on waledac. M.Sc. thesis, April 2009.

[8] F. Leder and T. Werner. Waledac traffic decoder. `http://oslo.cs.uni-bonn.de`.

[9] I. Macalintal. DOWNAD/Conficker watch: New variant in the mix? `http://blog.trendmicro.com/downadconficker-watch-new-variant-in-the-mix%`, April 2009.

[10] P. Porras, H. Saidi, and V. Yagneswaran. A foray into conficker's logic and rendezvous points. In *Proceedings of the* 2^{nd} *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*, April 2009.

[11] The Honeynet Project. Know your enemy: Fast-flux service networks. `http://www.honeynet.org/papers/ff`.

[12] S. Wu, T. Zink, and S. Molenkamp. Where is Waledac? *Virus Bulletin*, pages S1–5, June 2009.