

## 00 – Hands on!

The first thing I do as soon as I have some executable files in my hands is to disassemble them.

I know, there are a lot of faster ways to try to discover what is the purpose of the executable (besides, obviously, executing it)... We can run, for example, the Sysinternals command **strings** to expose the Unicode and ASCII strings it contains. But not all the string may be used... It is the case of our malware, which contains a big list of usernames and possible passwords (starting at string – in reverse order - “zimmerman”), which will be never used! Furthermore, the program has the potentiality of storing all strings in an encrypted way, and decrypt them on the fly, according to the value of the preprocessor define `NO_CRYPT` (defined for this compilation).

We can run, maybe, the Microsoft Dependency Walker (**depends.exe**) to see all the functions imported at startup from the system DLLS. But even in this case our program loads the interesting imported functions at runtime with a mechanism `LoadLibrary() / GetProcAddress()`.

We can use **LordPE** to show the sections, the directory entries and other interesting info extracted from the program headers.

But nothing is more exhaustive as disassembling the program. This is a slow practice, but it allows to penetrate the inner mechanisms of the code. In my opinion, nothing is best than **Ida**, for reaching the purpose, because it allows you to recognize the runtime functions and, consequently, the compiler used by the programmer.

A good disassembler is, for the reasons seen above, my first friend for analyzing every kind of programs, not excluding the malwares. I start to annotate on the disassembly list all the variables and subroutines used by the program; after a careful analysis, a good synthesis will complete our job.

But since we are humans and we can do many errated guesses, or simply to shorten the analysis step, the second friend is a good debugger, in order to execute step by step the program and to verify the findings we are proud of. I usually use a rather poor one, Microsoft Visual Studio 6 debugger (**msdev.exe**). It cannot for example put hardware breakpoints, but it is rather robust and it allows to debug nearly everything.

And I forgot a third, very important, friend, as we will see soon: **Google!**

But let's start to enumerate the details about the analysis performed on the current target. We will divide the present document into paragraphs, each one responding to one of the 10 questions created by the challenge staff.

## 01 – Describe your malware lab

Well, this is simply answered: no lab at all! We could try to run the beast under some virtual machine or some esoteric sandbox, and try to see what happens. But this is not the case of a simply executable like this.

The approach I followed, with the help of the three friends seen above, was to disassemble the proposed malware and to start to understand what it is doing, then to execute it step by step. During the trip, to avoid some virulence or to allow a further debugging, I made some patch to the original executable, using a little useful tool: **rta.exe**, by Squidge.

## 02 - What information can you gather about the malware without executing it?

Without executing it, and also without entering in the inner details of the disassembled program, we can gather some interesting info in two successive steps: submitting the program to some antivirus, and googling for what we found.

The Free Avira Antivirus reported: **WORM/Rbot.210944**  
After decompression: **KIT/WebView.1**

Some more detailed information is provided to us, for example, by **VirSCAN.org**:

### File information

File Name : malware.exe  
File Size : 75264 byte  
File Type : PE32 executable for MS Windows (GUI) Intel 80386 32-bit  
MD5 : 59a95f668e1bd00f30fe8c99af675691  
SHA1 : 2d1c8898ccc33c58c552f7a7091b165088c180d5

### Scanner results

Scanner results : 87% Scanner(33/38) found malware!  
Time : 2008/10/12 12:19:48 (CEST)

<u>Scanner</u>	<u>Engine Ver</u>	<u>Sig Ver</u>	<u>Sig Date</u>	<u>Scan result</u>	<u>Time</u>
a-squared	4.0.0.16	2008.10.11	2008-10-11	-	1.422
AhnLab V3	2008.10.11.00	2008.10.11	2008-10-11	Win32/IRCBot.worm.Gen	0.973
AntiVir	7.8.1.34	7.0.7.28	2008-10-11	Worm/Rbot.210944	2.373
Antiy	2.0.18	20081011.1474741	2008-10-11	-	0.119
Arcavir	1.0.5	200810110904	2008-10-11	Trojan.Rbot.Bzf	1.360

Authentium	5.1.1	200810100520	2008-10-10	W32/Ircbot.1!Generic (Possible)	1.088
AVAST!	3.0.1	081011-0	2008-10-11	Win32:Rbot-CSN [Trj]	0.025
AVG	7.5.52.442	270.8.0/1720	2008-10-11	IRC/BackDoor.SdBot4.FOV	1.633
BitDefender	7.60825.18666117.21234		2008-10-12	Generic.Sdbot.8CA4036D	3.149
CA (VET)	9.0.0.143	31.6.6141	2008-10-10	Win32/Rbot!generic worm.	7.544
ClamAV	0.94	8414	2008-10-12	Trojan.Mybot-1445	0.064
Comodo	2.11	2.0.0.674	2008-10-12	-	0.431
CP Secure	1.1.0.715	2008.10.12	2008-10-12	BackDoor.W32.SdBot.ji	6.150
Dr.Web	4.44.0.9170	2008.10.12	2008-10-12	BackDoor.IRC.Sdbot.origin	3.450
ewido	4.0.0.2	2008.10.12	2008-10-12	Backdoor.Rbot	2.994
F-Prot	4.4.4.56	20081011	2008-10-11	W32/Ircbot.1!Generic	1.097
F-Secure	5.51.6100	2008.10.11.01	2008-10-11	Backdoor.Win32.Rbot.bzf [AVP]	3.696
Fortinet	2.81-3.113	9.636	2008-10-12	W32/RBot.5	0.183
Ikarus	T3.1.01.34	2008.10.12.71627	2008-10-12	Backdoor.Win32.Rbot.bzf	3.789
JiangMin	11.0.706	2008.10.12	2008-10-12	Backdoor/Agobot.Gen.f	1.259
Kaspersky	5.5.10	2008.10.12	2008-10-12	Backdoor.Win32.Rbot.bzf	0.175
KingSoft	2008.9.8.18	2008.10.12.15	2008-10-12	Win32.Hack.RBotT.a.83968	0.643
McAfee	5.3.00	5403	2008-10-10	W32/Sdbot.worm	2.320
Microsoft	1.4005	2008.10.12	2008-10-12	Backdoor:Win32/Rbot.gen	4.616
mks_vir	2.01	2008.10.11	2008-10-11	-	2.812
Norman	5.93.01	5.93.00	2008-10-10	W32/Spybot.DCJP	5.155
nProtect	2008-10-10.00	2229401	2008-10-10	Generic.Sdbot.8CA4036D	4.496
Panda	9.05.01	2008.10.10	2008-10-10	Generic Malware	2.197
Quick Heal	9.50	2008.10.11	2008-10-11	Backdoor.Rbot.bzf	2.490
Rising	20.0	20.65.40.00	2008-10-10	-	0.919
Sophos	2.79.0	4.34	2008-	W32/Rbot-Fam	2.068

Sunbelt	3.1.1716.1	2303	10-12 2008- 10-11	Backdoor.Rbot	0.459
Symantec	1.3.0.24	20081011.003	2008- 10-11	W32.Spybot.Worm	0.099
The Hacker	6.3.1.0	v00108	2008- 10-11	Backdoor/Rbot.bzf	0.459
Trend Micro	8.700-1004	5.594.24	2008- 10-12	Mal_MLWR-5	0.042
VBA32	3.12.8.6	20081011.2103	2008- 10-11	Backdoor.Win32.Rbot.bzf	1.241
ViRobot	20081010	2008.10.10	2008- 10-10	Backdoor.Win32.IRCBot.75 264.E	0.402
VirusBuster4	5.11.10	10.89.14/634234	2008- 10-11	Worm.Rbot.ACZA	1.022

Other precious informations, even if not always exact, can be found searching here and there in the forums all over the net. As an example, let's see in **elhacker.net**:

[skapunky](#)  
Electronik  
Colaborador

Desconectado

Mensajes: 1.238

[www.killtrojan.es](http://www.killtrojan.es)

[Re: Malware Challenge \[2008\]](#)

« Respuesta #2 en: Ayer a las 13:24 »

- Crea el siguiente archivo: C:\WINDOWS\Winsec32.exe y le crea un proceso con "CreateProcess.
- Crea un único mutex como "RasPbFile"
- Crea las siguientes llaves en el registro para autoejecutarse:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run "" = Winsec32.exe
```

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices "" = Winsec32.exe
```

```
HKEY_CURRENT_USER\Software\Microsoft\OLE "" = Winsec32.exe
```

- Por alguna razón el propio virus recoge la siguiente información:

```
Directorio de windows.  
Hora del sistema.  
Nombre de la computadora.
```


- Se intenta conectar al siguiente hosting remoto:

```
testirc1.sh1xy2bg.NET
```

- Mantiene comunicación con un servidor IRC donde genera como muestra el siguiente trafico:

Citar

```
NICK USA[XP]5483063  
USER lseyljt 0 0 :USA[XP]5483063  
USERHOST USA[XP]5483063  
MODE USA[XP]5483063 +i-x+s  
JOIN #challenge happy12  
NOTICE USA[XP]5483063 :.VERSION http://www.W32-gen.us (-National Virus Site-).  
NOTICE #challenge :USA[XP]5483063 has just versioned me.  
PRIVMSG #challenge :RealmBoT (irc.p.l.g) .... Status: Ready. Bot Uptime: 0d 0h 0m.  
PRIVMSG #challenge :RealmBoT (irc.p.l.g) .... Bot ID: 1.  
PRIVMSG #challenge :RealmBoT (portscan.p.l.g) .... Exploit Statistics: VNC: 0, Total: 0 in 0d 0h 0m.  
PRIVMSG #challenge :RealmBoT (irc.p.l.g) .... Uptime: 0d 0h 35m.  
PRIVMSG #challenge :[REALMBOT] : Failed to start scan, port is invalid.  
NICK USA[XP]9177980  
USER iiturtci 0 0 :USA[XP]9177980  
USERHOST USA[XP]9177980  
MODE USA[XP]9177980 +i-x+s  
NICK USA[XP]8779631  
USER iiturtci 0 0 :USA[XP]8779631  
USERHOST USA[XP]8779631  
MODE USA[XP]8779631 +i-x+s  
NICK USA[XP]7743805  
USER vlluxig 0 0 :USA[XP]7743805  
USERHOST USA[XP]7743805  
MODE USA[XP]7743805 +i-x+s
```

Ale, así lo dejo que si no no quedará nada para analizar . Un saludo.

## 03 - Is the malware packed? If so, how did you determine what it was?

But before disassembling the malware, a little preliminary step is mandatory: to find out the compression algorithm, which in the present target is present.

This can be inferred by the ZIP itself: the compressed dimension (70K) is more or less similar to the extracted one (73K).

Even in this case Google resulted to be the best tool! Before opening the executable with some hex editor or scanning it with some packer-protection finder, a hint was provided on the great forum on [www.woodmann.net](http://www.woodmann.net) (see Acknowledgements below): replace all the occurrences ABC with UPX.

In fact, if we look inside the file, with a little knowledge of the UPX formats (it is open source, so everyone can gather it!), we find that there are four occurrences of ABC; three are used as section names (ABC0, ABC1, ABC2), and the fourth one is used as signature (ABC!) of the special header, containing the file data, at the end of PE header (address 0x3E0). We replace them with UPX and voila! **Upx.exe -d** (I tried release 3.03) will automatically do our decompression job.

---

Now we can start to annotate the disassembly listing (see included ascii file **malware.asm**). After the activity of some hours of playing with disassembly and debugger, the third friend, google, gave once again the best results: it discovered the presence on the net of the whole sources, structured in a project for Microsoft Visual Studio 6.0. This allowed a faster and more exact analysis of the program, even if some care was taken in order to detect possible differences between the sources and the binary. Some little differences were found, as we will see. Furthermore, the sources are structured in a modular way, with some preprocessor **ifdef / ifndef** directives; we must see what modules were held out of the binaries. For example, the RFI module is not included (NO\_RFI defined) and we will not take it into consideration.

At a first glance I thought that the original language was 'C', compiled with Microsoft Compiler, but after the sources were located, the language resulted to be **C++**.

In fact, there are no classes definitions at all, and the only extensions used against the C language are: (a) the possibility to define new local variables in every place of the code flow; (b) make use of functions with omitted parameters (the default values are in any case pushed by the compiler). Both extensions cannot be detected in the resulting assembly language.

The most important file in the source tree is **crxbot.cpp**, followed by **scanner.cpp**, **vnc.cpp** and **base.cpp**. I have included all of them to this submission, along with **cfg.h** (containing some global definitions) with plenty of comments and a different indentation (for me the indentation is very useful to help to understand the code).

The original code is in some places difficult to read, very poorly commented, and presents some bugs, as we will see.

Some unusual programming technique can be found, as in the case of functions

returning a whole structure, not just a pointer to the structure. Since the value returned by a function is always in register EAX, it could be interesting to see how the Microsoft Compiler solves this case.

Take for example

DKSPKB DiskSpace(LPCSTR Drive)

where DKSPKB is a structure defined as a collection of three ascii buffers. At calling time the compiler will push on stack 2 arguments instead of 1:

arg\_0: pointer to the returned struct!

arg\_4: Drive

The value returned in EAX is at this point no more significative.

## 04 - Describe the malware's behavior

The first thing the malware does is to dynamically load all required DLLs and solve the exported functions that will be called afterwards: see *LoadDLLs()*. This is performed maybe to hide those DLL from a statical analysis of the executable, but also to allow the loading of the malware itself even if some DLL is missing.

A malware has two main actions: first, it inquinate the victim, installing itself in the host computer and running every time the computer is booted; second, it inquinate the net, propagating through it and performing harmful attacks against other computers, mainly servers. As a bonus, a third action may be that of stealing files and even passwords, digited on the fly, from the infected computer. All the three actions are unfortunately perpetrated by the infected computer, as we will see.

The first two actions are kept well separated in our program. Part of the first one (copying itself in a secure directory) is executed from a father process. Then a child process is created which re-executes all the same code. But it understands to be a child from the location it was runned from: the Windows system directory.

A named mutex is created, with the name "1" (the "bot id"): this is used from the child to wait until the father dies, and to disallow two concurrent executions of the program.

### 4a: What files does it drop?

The father copy itself with the name **winsec32.exe** in the windows directory (generally [c:\windows](#)). A flag, *rndfilename*, allows it to mutate its name in a random way, but it is not enabled in the present binary. The original name was in **cfg.h: nodkrn23.exe**

The file times are brought back and made the same as the Explorer executable in order to not awake suspects.

The file attributes are set as FILE\_ATTRIBUTE\_HIDDEN + FILE\_ATTRIBUTE\_SYSTEM + FILE\_ATTRIBUTE\_READONLY.

## 4b: What registry keys does it create and/or modify? How does it auto-start?

If a flag, *AutoStart*, is enabled, and this is our case, some registry entries are added, in order to allow the automatic execution of the malware every time Windows is booted.

The same entry

"Microsoft Svchost local services" = <installed\_child\_exename>

is added in each of the three following keys:

HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run

HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices

HKEY\_CURRENT\_USER\Software\Microsoft\OLE

---

Some initializations are kept out of the proposed binaries (NO\_SECSYSTEM, NO\_REGISTRY, NO\_IDENT defined in phase of compilation) but some other important initializations are yet performed. Let's see.

(1) the program keeps an array *threads[]* of different processes created during its activity, excluding the installation in the victim computer, that we have already seen. Every slot of this array is dedicated to a new process; every process may assume different personality. We will refer to it as 'thread\_id'. For example the 'thread\_id' of the main process (the child, since it was already installed by the father) is 0 (MAIN\_THREAD), and it is now time to fill its slot, which will be found at index 0, using the function *addthread()*

(2) a log buffer is maintained by the program and it is here initialized with its first entry: "RealmBoT (irc.p.l.g) .᠒᠒. Bot started."

(3) through the function *sethttp()*, set the variable *szHTTPHost* to "<http://www.Nivdav.net/Winsec32.exe>". That is the target of the VNC exploit (see below). In brief, it is the URL of an executable which will be executed without the consciousness of the computer user. Unfortunately, it is no more available on the net for analysis, but probably is no more than a copy of this malware itself, judging from its name. This make no much sense, since two concurrently copies cannot run on the same computer (see the mutex above). The original sources had, as target, the URL "<http://ciudad.latinol.com/reptilex2/new.exe>". This is yet downloadable: it is a little Visual Basic programs that install a new malware, named MDM.EXE, aimed at opening a great number of ports on the infected computer.

(4) some more configuration variables are set here on the fly, and this is maybe the greatest difference between the sources, where they are initialized statically, and the binary:

*server[]*, the name of the IRC server we will try to connect to, is set to

"**testirc1.sh1xy2bg.NET**"; the original server name was

"**nonna.sytes.net**"; none of them is existing anymore

*channel[]*, the IRC channel to join to, is set to "**#challenge**"; the original channel was "**#vml**"

*chanpass[]*, the channel password, is set to "**happy12**"; the original one was "**pass**"

Other static replacements for global variables are found in the comments of the file **cfg.h**, included with this submission.

#### 4c: What network connections does it create?

Finally a loop is entered in which connections are attempted with the above mentioned IRC server, on standard IRC port 6667.

First we check the Internet connection state, allowing up to six retries at intervals of 30 seconds if none is found.

Then, if the Internet connection is ok, an *irc\_connect()* function is entered.

Every *irc\_connect()* is similarly an endless loop in each step of which a new socket is created and connected, and a bot-handler function *irc\_receiveloop()* is called.

In my opinion the *irc\_connect()* can return only with a value of 2, in case that a QUIT condition is detected. The outer loop, at a return of 2 from *irc\_connect()*, breaks, and the malware is exited. Strangely enough, the outer loop handles also return codes different by 2, in which cases it sleeps 3 seconds and cycle again. This seems to me a dead code that can never happen.

#### 05 - What type of command and control server does the malware use?

*irc\_connect()* is the main function used by the initial program and all the clones/spies it will eventually create in the course of its life.

As we have said, it start an endless loop in which a new socket will be created and connected, and subsequently operated upon, thanks to the functions *irc\_receiveloop()* -> *irc\_parseline()*, according to the Internet Relay Chat protocol.

The IRC protocol is a little outdated today, after the boom of more modern socialization systems like facebook, but it is yet used and many servers can be found on the net. The protocol specifications were established about 20 years ago and can freely downloaded as RFC1459, the document I have studied to understand the actions taken by the malware. All the command and answers are ascii strings.

An amusing way to try it is to download a Windows IRC server like **bircd.exe** (less than 150K), edit the file **ircd.conf** in order to replace the 90 ping freq with 0 (otherwise we will be forced to continuously answer to server PING requests), and run it without need of installation. It will appear as a little icon in the taskbar tray. Now we can send simple commands to our local server with a generic TCP client. It is enough to run, from a command prompt, "**telnet localhost 6667**".

The first two commands we must send are always USER and NICK or viceversa, for example "USER foo 0 0 foo" and "NICK foo", followed by an answer to the first PING the server will send us. If the server sends "PING :12345678" we will answer PONG 12345678. At this point the connection is done and the server will send us some initial messages.

The job of the *irc\_receiveloop()*, before calling *irc\_parseline()*, is exactly that of handling the initial connection with the remote IRC server, following the simple steps seen above.

Subsequently it enters a loop, one loop step for each line received from the server: every single line is passed to the `irc_parseline()` function.

It is time to examine the `irc_parseline()` function. It first splits the line into separate tokens to be stored in the array `a[]` and in the array `parameters[]`. The latter holds 256 elements, one for each byte value, which will be set if the corresponding letter is passed as an option. For example, the option “-a” will set the element `parameter[0x61]`.

The remainder of the function (about 2400 lines of code in my restyling) implements a malicious IRC Bot; a Bot, because it acts as a robot which can take automatic decisions following some lines received from the server; a malicious bot, because it extends the set of commands understood by the server, implementing some evil ones.

The way of extending the commands is simple and smart: it is the same as the one that regular clients use to communicate with each other, PRIVMSG (to send private messages) and NOTICE (private messages without echo). The contents, or message, makes sense to the bot, which will take it as an encoded command.

Another way of making commands available to the bot for being parsed is to send them as a topic string, if the flag `topiccmd` is enabled (and it is in our case). This way, as soon as a user successfully JOINS a channel, the IRC server sends the channel's topic with a RPL\_TOPIC answer. The RPL\_TOPIC answer is successively parsed by the bot as a sort of starting command. In the current malware, the string `topics[]` is set at the first channel joining as “.asc vnc 75 0 0 -r -b”, which means Advanced Scan for VNC Exploit with random IP address, as we will see below.



Let's see first the automatic, not-malicious actions:

(1) It can send a PONG in response to a server PING. At first PING it tries to join and start the new #challenge channel, setting as TOPIC the string “.asc vnc 75 0 0 -r -b”, seen above.

(2) at the welcome messages (answers 0001-005) run a command USERHOST to get info on the host and try to join the created channel

(3) at reply RPL\_USERHOST scan the string got to store the host name

(4) it can change a nickname, randomly chosen, if the server returned ERR\_NICKNAMEINUSE

(5) it can try to rejoin a channel if someone sent a KICK command against him. Furthermore it updates an array of users, `masters[]`, currently made of only two entries, containing the names of the users enabled to give commands and entered with the extended command LOGIN (see below).

(6) the `masters[]` array is updated even if NICK/PART/QUIT commands are sent by one of the masters.

## **06 - What commands are present within the malware and what do they do?**

Finally we must take into account the extended commands. Every extended command starts with a prefix, initially defined as '.' and eventually changed on the fly. Unfortunately no documentation was found in Internet, so all the following information had to be extracted from the program sources.

An alias substitution and variable substitution is initially performed. The recognized variables are **\$N-** (token N and followings), **\$N** (token N), **\$me** (nick), **\$user** (user),

**\$chan, \$rndnick** (for random nick generation), **\$server, \$chr(N)**

And now the commands interpreted and executed by the bot, each with its arguments... For time reasons, I have to be extremely concise... For the details, the reader should have a look at the comments in the included sources.

### **LOGIN / L password**

This is the first command to be sent to authenticate ourselves, and enter in the *masters[]* array. The password is checked and must be equal to **“gemp123”**. The host is checked and must be equal, in the current binary, to **“\*@legalize.it”**  
All the following commands require LOGIN authentication, or to be defined as channel topic.

### **LOGOUT / LO [masters\_slot]**

Logout the user at given index in *masters[]*, the current user by default

### **RNDNICK / RN [-p]**

Random nick change.  
'-p' option instructs to add prefix in *rndnick()*

### **VERSIONSHIP / VER**

Return the current version of the bot: **"Crxbot Alias REalmbot -by Lindem-"**

### **SECURE / SEC**

Secure the computer shares: check two registry entries and remove all the invisible shares (terminated by '\$').  
Software\Microsoft\OLE\EnableDCOM must be set to N, and  
SYSTEM\CurrentControlSet\Control\Lsa\restrictanonymous must be set to 1

### **LOCKDOWN.OFF / LD.OFF**

Unsecure the computer shares:  
Software\Microsoft\OLE\EnableDCOM must be set to Y, and  
SYSTEM\CurrentControlSet\Control\Lsa\restrictanonymous must be set to 0.  
Then add 4 predefined shares: IPC\$, ADMIN\$, C\$, D\$, and all the other partition names, with the trailing '\$', to make the share invisible.

### **SECURE.STOP threadid**

stop SECURE thread above

### **REDIRECT / DAEMON.RD lport dest destport**

Start REDIRECT thread. In my opinion there is a little bug here. The thread generates two children, and both are receiving from the same socket...

### **PROXY.REDIRECT.OFF threadid**

Stop REDIRECT thread

### **SYNFLOOD / DDOS.ACK / DDOS.RANDOM ip port seconds\_length**

start DDOS thread. It fill the target TCP/IP stack with SYN packets. The target will try to send SYN and wait ACK, but none of these actions will be successful.

### **DDOS.OFF threadid**

stop DDOS thread

### **SYN.OFF threadid**

stop SYN thread (who start it ???)

### **UDPFLOOD / DDOS.UDPF / U host packets\_num packets\_size delay [port, rand if missing]**

start UDP thread. This time the flood is performed with UDP packets.

### **UDP.OFF threadid**

stop UDP thread

### **STATS / ST**

List the number of VNC/RFI exploits attempted

### **REBOOT**

do a system shutdown

### **THREADS / THREADS.L [sub]**

start LIST thread: list threads and eventually sub-threads

### **NETINFO / NI**

netinfo about the IRC host name

### **SUPERSYN ip port loops**

start SUPERSYN thread: open 400 connections all at once

### **SYSINFO / SYS**

run sysinfo()

### **REMOVE / RM**

disinstall the current bot

### **OPENCMD / CMD1**

start RCMD thread if not already running  
remote commands must be sent through CMD

### **CLOSECMD threadid**

stop RCMD thread

### **CMD remote\_command**

send a command to an invisible command prompt, using pipes mechanism, and get its answer. RCMD must have been previously started

### **FLUSHARP / FARP**

flush the ARP

### **FLUSHDNS / UTIL.FDNS**

use dnsapi.dll

### **CURRENTIP / CIP [threadid, SCAN\_THREAD by default]**

get the current IP

### **WEB.ON / HTTPD.ON [-d to disable] [port [dir]]**

start an HTTP server thread (limited to clause GET)

### **WEB.OFF threadid**

stop HTTP server thread

### **FTPD.ON / D.FTPD.ON**

start a FTP server thread, if not already started, which only knows how to send the current program

### **FTPD.OFF thread\_id**

stop FTP server thread

**KILLTHREADS / KILLT "all"/list\_of\_slots**

**KILLPROCESS / KPC [procname]**

kill process name

**PROCKILLID / PKID pid**

kill process id

**DELETE / DEL filename**

use API DeleteFile()

**COM.FL / LIST directory**

**MIRC.COMD command**

send a command to mIRC using a shared memory protocol. After filling the shared memory, send to mIRC a message WM\_USER+200, so that it can peek the memory and can leave its answer in the same memory

**COM.O / OPEN target**

execute a shell open verb on target (file or object)

**COM.RF / READFILE filename**

**COM.E / EXECUTE do\_show command**

CreateProcess

**COM.MV / RENAME srcfile dstfile**

**COM.GC / GETCLIP**

echo the text found in clipboard

**COM.PS / PROC.ON [full]**

start PROC thread if not already running: list all the processes running in the system

**COM.PS.OFF / PROC.OFF threadid**

stop PROC thread

**COM.UP / UPTIME [uptime]**

**COM.DRV / DRIVEINFO [x:\]**

**COM.DLL / TESTDLLS**

check if all required DLLs have been dynamically loaded

**KEYLOG.ON/CMD.KL.ON [pay/normal [channel]]**

start KEYLOG thread if not already running. This is a powerful way to intercept all the keys that the user digits on the keyboard. If "pay" is requested, special attention is made to windows which have a caption related to some pay sites, listed in an apposite array.

Little bug: the polled keys are 92 instead of 95

**STOP [thread\_slot]**

stop KEYLOG thread

**NET START [servicename]**

**NET STOP servicename**

**NET PAUSE servicename**

**NET CONTINUE servicename**

**NET DELETE servicename**

**NET SHARE [-d] [name [path]]**  
add or delete or list a share

**NET USER [-d] [name [pass]]**  
add or delete or list

**NET SEND msg**

**CLONE.MAKE/CLONE.START host port chan [chanpass]**  
start a CLONE thread (a bot without commands interpreter)

**CLONE.RA / RAWCLONE thread\_slot string**  
raw send to socket of passed clone

**CLONE.MODE/CLONE.M thread\_slot new\_mode**  
send MODE to socket of passed clone

**CLONE.NICK/CLONE.NI thread\_slot new\_nick**  
change NICK on a clone

**CLONE.JOIN/CLONE.J thread\_slot channel chanpass**

**CLONE.PART/CLONE.P thread\_slot channel**

**CLONE.OFF threadid**  
stop CLONE thread

**CLONE.QUIT / CLONE.Q thread\_slot**  
force the clone to send a QUIT, close its sock and kill it

**CLONE.RNDNICK / CLONE.RN thread\_slot**  
force the clone to randomly change its nickname

**CLONE.PRIVMSG / CLONE.PM thread\_slot dest msg**

**CLONE.ACTION / CLONE.AC thread\_slot receiver message**  
force a clone to send extended IRC command ACTION

**IRC.REPEAT / IRC.RP times\_to\_repeat command**  
force repeat return value and replace line to repeat

**IRC.DE / DELAY seconds command**  
replace line and reparse it after a sleep

**UPDATE [-e] url new\_botid [expectedcrc filelen]**  
start UPDATE thread

**DOWNLOAD / DL [-e] url dest [do\_run [expectedcrc [filelen]]]**  
start DOWNLOAD thread (same code as UPLOAD thread)

### **ADVSCAN/ASC [-a/r/b] method/port threads delay minutes [ip] [#msgchan]**

IP may contain 'x' for randomness

start SCAN thread if not already running

This is the most interesting attack, called VNC exploit. It allows to bypass the authentication mechanism used by a VNC server and execute all the commands you want.

To verify this I downloaded the RFB (remote frame buffer) specifications, a server VNC (winvnc4.exe) – tried release 4.1.1 from RealVNC – and a generic TCP client which, differently from telnet, is able to send and receive also hex codes (TCP Port Toolkit).

### **CHGHTTP target**

change HTTP target in VNC exploit

### **SCANSTOP threadid**

stop SCAN thread

### **PINGFLOOD / DDOS.PINGF / P host packets\_num packets\_size delay**

start PING thread

### **PING.OFF threadid**

stop PING thread

### **HTTPCON / UTIL.HCON host port method url [referer]**

### **FTP.UPLOAD host cmd1 cmd2 cmd3 file\_to\_upload**

### **IRC.GH / GETHOST host [command]**

replace "line" with the passed command

without command is like NETINFO

### **IRC.AA / ADDALIAS alias command**

here a bug contained in the original sources (see function *addalias()*) is corrected in the present binary: look at the sources!

### **IRC.PM / PRIVMSG receiver message**

### **IRC.AC / ACTION receiver message**

send extended IRC command ACTION

### **IRC.CY / CYCLE delay\_seconds channel chanpass**

part and join again a channel

### **IRC.M / MODE new\_mode**

### **IRC.DI / DIE**

Terminate the current process

### **IRC.LG / LOG [entries\_no | search\_substring]**

start LOG thread

### **LOG.OFF threadid**

stop LOG thread

### **CLEARLOG / CLG**

clear all the log entries

**IRC.V / VISIT url [referer]**

Explore the given url

**IRC.R / RECONNECT**

return special code 0, in order to reconnect soon

**IRC.D / DISCONNECT**

return special code -1, in order to reconnect after a sleep

**IRC.Q / QUIT [quit message]**

return special code -2

**IRC.S / STATUS**

return the current status (always ready) and the bot uptime

**IRC.I / ID**

return the bot id "1"

**IRC.AL / ALIASES**

list recognized aliases

**IRC.WHO**

dump masters[] slots

**IRC.NICK/IRC.N [new\_nickname]****IRC.J / JOIN channel chanpass****IRC.PT / PART [channel]****IRC.RA / RAW string****IRC.PR / PREFIX new\_prefix\_char**

change the commands prefix

**IRC.SE / SETSERVER server**

replace the current server (passed arg "server" points to irc.host -> server[])

**IRC.DN / DNS address/name**

lookup param returning name/address

## *07 - How would you classify this malware?*

Well, potentially this malware is highly dangerous, because it can propagate itself across the net and it can perform heavy attacks on many computers, and steal sensible data, especially if it is driven by expert "masters".

But, fortunately, the IRC server which allows the communication between the bot and a "master" does not exist, or is dead, and this greatly reduces, or nullifies, the harm that can be done to the Internet.

Nevertheless, its capacity of installing and auto-running itself on the victim computer is very annoying, and for this reason this piece of software cannot be taken too lightly.

### *08 - What do you think the purpose of this malware is?*

Exactly like all similar malwares and as we already said, this program has multiple purposes: infecting the computer on which it runs, infecting and propagating on the net, attacking some big servers in complete anonymity, stealing data and passwords.

## 09 - Is it possible to find the malware's source code? If so, how did you do it?

Many configuration variables, as we saw, has been changed; but fortunately the version string, "**Crxbot Alias REalmbot -by Lindem-**", has not been altered.

This keywords have been used by me, with different combinations, as a good starting point to locate info – and possibly the sources – of the malware we are talking about

At least three places were storing the sources, exactly in the same version, when I looked for them:

*rapidshare.com/files/149970577/Crx-realmbot.VNC\_RFI.rar.html*  
*darksun.ws/download/Bots/*  
*lulzcoderz.info/Bots/*

The RAR file has the name **Crx-realmbot.VNC.exploit.and.RFI-(rfi.not.tested).rar**

## 10 - How would you write a custom detection and removal tool to determine if the malware is present on the system and remove it?

Since the malware does not interfere with system files and not modify them, but its infection is limited just to one file dropped and three registry entries added, the job is not so difficult.

To detect if the malware is present on the system, we could find it on disk by name ("winsec32.exe"), by size, or even by date (the same as explorer.exe), or we could find it through the auto-start registry entries (there are specialized utilities for this purpose, like *Startup Control Center*).

To remove it, it is sufficient to delete it from the hard disk, and reboot the system. For a complete removal, also the three new entries in the registry should be removed.

## 99 – Acknowledgements

A first acknowledgement is due to **evaluator**, for having showed out the trick of the ABC replaced by UPX.

A second acknowledgement is due to **Kayaker**, for having pointed out the contest.

A third acknowledgement is due to **Woodmann**, for hosting the bulletin board.

The thread can be found at the link

<http://www.woodmann.com/forum/showthread.php?p=77357>