Binary Analysis and the 1st Spanish Forensic Challenge

IRIS-CERT. RedIRIS

Feb 8 2005. FIRST -TC Paris

Format of the hand-on-class

Requirement:

- Laptop: You need a laptop/computer to connect to the virtual machines and do the exercises.
- Ethernet cable:for the network connection
- SSH client to access to the system

- Knowledge of :
  - Unix & Linux system
  - Incident analysis
  - C programming language

The lab

The laptop contains different vmware machines running Linux that could be used as main system

We are going to setup the wired LAN, with static address to connect to the main system and later to the compromised systems.

Network is 192.168.100.0 , netmask 255.255.255.0

The main system is at 192.168.100.201, choose your IP address from the 192.168.100.100 to 192.168.100.200 range

Username/password are: firstN/firstN, (ex, login first1, password first1

All the binaries and tools are in the “/first/” directory:

/first/handonclass/pres: this presentation

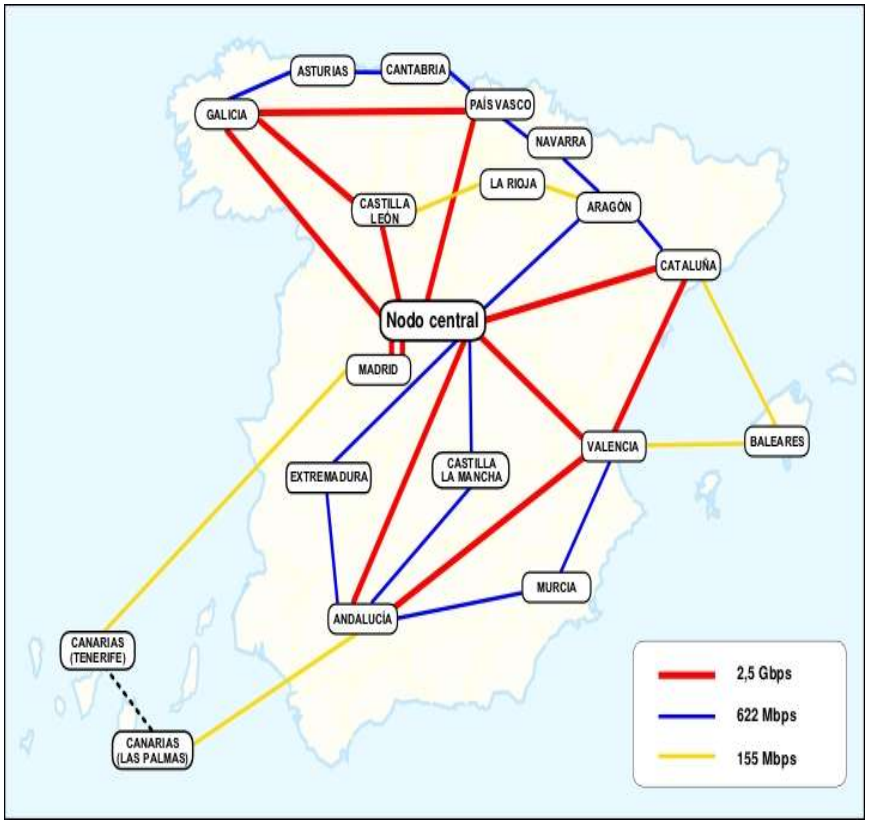
/first/handonclass/exercises

/first/handonclass/tools: some tools

Index

IRIS-CERT: Spanish Academic and Research Network.

- About 250 directly connected Universities and Research Centers.
- Coordination of incidents ,with local staff in each center.
- Don't deal directly (at first steps with the end user)
- We don't support “in site” incident analysis , but provided limited analysis of the binaries discovered in the Universities



Incident cycle

1. The typical users don't know that their system have been compromised, usually we received a complain about scanning/ hacking activity from some host inside our constituency
2. We send to the technical contacts (TC) in the organization the information regarding this incident, asking them to investigate the incident , sometimes we can also send information about the possible tools used.
3. The TC contacts with the end user and sometimes they can send us back some information
4. We analyze the data, usually we don't obtain disk images of the compromised system, instead the user send us binary and logs files.
5. At the end after we finished the analysis and close the incident.

Live forensic

Sometimes we couldn't backup the hard disk data:

- Disks too large for network copy
- Slow network
- Confidential data
- Policy restrictions

Also , we can't reboot the machine and use a forensic boot disk like FiRE

- System can't be shutdown (for example primary DNS server)
- Administrator don't have the required skills.

More time you only got a tar.gz /zip file with suspected binaries found by the user

The solution is perform a analysis of the binaries in order to know how they work.

Light Malware Analysis

When we got the suspicious files, we want to know:

- What information are the Trojan binaries (ls , ps, etc) hidding ?
- What information has the intruder obtain from the compromise (look for sniffer logs)
- Has the system been used to compromise other systems ?
- What others tools had been used // installed ?

Most of the times this is a cyclic process :

- You ask for some information / files
- Users send you it.
- You analyze it and found some possible hidden directories/files
- ask for more information

How to detect rootkits and malware ?

Some tools / techniques that can be used:

- Chkrootkit, from http://www.chkrootkit.org
- Package database management in modern Unix // Linux System and static checksum (MD5, tripwire) of the system binaries
- Trusted files from another similar system
- Looking for suspicious files and directories in the system
- The techniques are the same as several years ago, the CERT/CC document about finding information in compromised system is still valid

Package management system

Most of the modern Unix keep a database of signature for installed binaries:

- Pkg format for Solaris, FreeBSD
- Debian/Gnu Linux maintain a optional MD5 database of installed files
- RedHAT RPM and derived has a MD5 database of each file installed using the package system.

Common intruder (script-kiddies) don't touch this database.

With rpm files:

- Rpm -Va checks the MD5 checksum of all the installed files
- Rpm -qla lists all the files.

This does not works for kernel level rootkits but still normal rootkit are quite frequent

Discrepancies

In most of the Linux system there are different command that provide the same functionality:

- Ls , dir, mc , echo “*” for file listing
- Netstat, lsof , cat of /proc/net/netststat for connections
- Ps , top, pstree, oldps cat /proc/NN/ for information of processes.

Most of the time we can see discrepancies:

- Listing of process with “ps -aex” different (some are not seen) when compared with:
- Direct ls of /proc/ to see the proccess

Some of this techniques can be used to detect kernel level rootkit (sometimes)

Light Binary Analysis

Objective: See what the binaries hides, where are the configuration files and approximately how it works.

- No full disassembly of the code for reversing engineering of code only small preview searching for suspicious lines
- You don't need to know how to write assembly code , only some skills reading the disassembled code
- The C compiled code follow some standards in how to process the information
- Use of opensource (objdump) program , for learning the basic of the analysis , with a small “wrapper” to allow easier analysis.

Linux Binary Files

Most Modern Unix share the same file format for binaries: ELF

Principal features of ELF:

- Allow dynamic linking of code , so the binaries are smaller.
- Builds shared libraries of code , so this library code is shared by different programs.
  - Allow replacement and update without changing the applications
  - The program are smaller than the static ones

As an additional feature, the binaries can be adapted to run in different operating system (Linux, Solaris, *BSD) in the same platform (Intel, SPARC).

Different Classes of ELF programs

You can build for different classes of binaries executables with the ELF format:

- Dynamic linked, smaller because the code of the function is in the libraries and linked to the file at runtime
  - Non stripped : usually the result of compiling a program without the optimization and with debug flags
  - Stripped, : Normally the usual binaries in a Linux system
- Statically linked: Bigger than the dinamic, the file included the code for the system libraries that are used by the program
  - Non strippedstatic compiled binaries with debug information
  - Striped static: Only used for recovery porpoises for the operating system

Why to have statically linked binaries ?

Dynamic linked binaries are very sensible to the changes in libraries (sometimes a small change could not allow them to run

for system recovery (for example some damage in the /libraries directory the static linked binaries can fix the problem it common to have:

- Dump, ext2fs, etc linked statically for system recovery

Also static binaries are fine if you want to share the same binary with different distributions (the same binary can be used with Redhat, Debian,etc) and also with different kernel version if the system calls don't change.

This static binaries are used by intruder:

- Allow sharing the rootkit without rebuilding it in different systems.
- More difficult to analyze
- It's possible to compress and encrypt the binary (see x2 sshd exploit)

Intel ASM introduction

Brief introduction to ASM:

- Registers:
  - General registers : eax , ebx , ecx , edx
  - Segment Registers: CS, SS, DS, ES , FS, GS
  - Program flow and stack registers: EIP, ESP, EBP

- Normal “not optimized” C -compiled code:
  - Eax is used as return argument (the result of calling a function)
  - Low eax (al) is used as function call index when dealing directly with the kernel

Interesting register

EIP : Extended instruction Pointer : Point to the next instruction to be executed by the CPU. Modify directly by the “jump” instruction and call and return instruction.

ESP: Extended Stack Pointer: Point to the global Stack (area of memory used to interchange data between functions (pushing the arguments of functions in it).

EBP: Extended Base Pointer: Used in “C” programs as the base pointer for storing local variables in the stack .

Assembly instructions:

Some of the common instructions:

- Mov : Move information between registers and memory or between registers.
- Lea: move information from memory to register
- Push/pop : add or delete information to the special memory zone called “stack”
- Call execute a function and return back to the next instruction
- Jump (conditional or not) allow to go to other parts of code depending of some conditions.
- Cmp , test: compare values and test if meets some condition
- Arithmetic : (add , sub) and logic (not, xor, and) between registers

Subrutine Call

We are going to focus in the functions (subrutines) calls in order to analyze the flow of the programs:

Allways is the same operation:

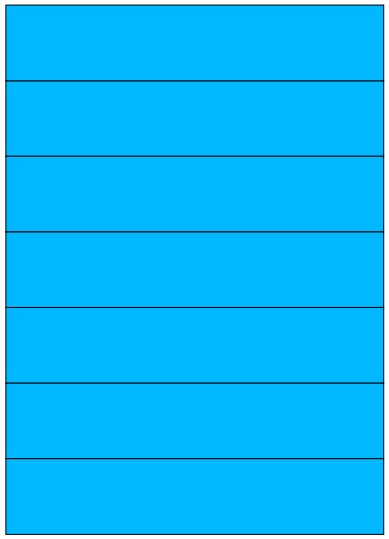
- The program push (add) to the stack the arguments of the function , starting from RIGHT to left
- A call to the address (placing the return address also in the stack is made)
- After the return the stack is “incresed” so the values are removed from the stack

Stack and function call

If we have the following code:

```
int add ( a,b) {
 int a,b ;
return (a+b) }

main () {
 int c ;
c=add (10,20) ;
}
```



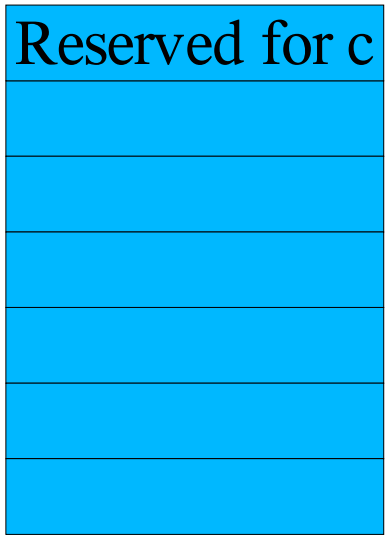
A simplified version of how the stack works:

Stack and function call

If we have the following code:

```
int add ( a,b) {
 int a,b ;
return (a+b) }

main () {
 int c ;
c=add (10,20) ;
}
```



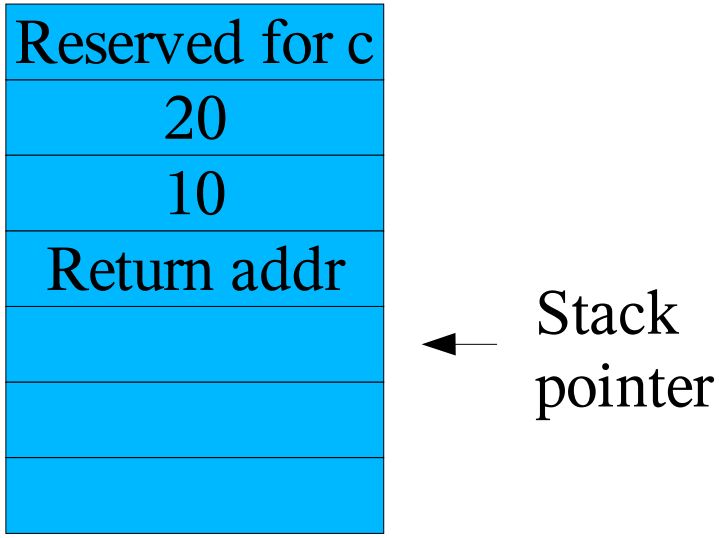
A simplified version of how the stack works:

Stack and function call

If we have the following code:

```
int add ( a,b) {
 int a,b ;
return (a+b) }

main () {
 int c ;
c=add (10,20) ;
}
```



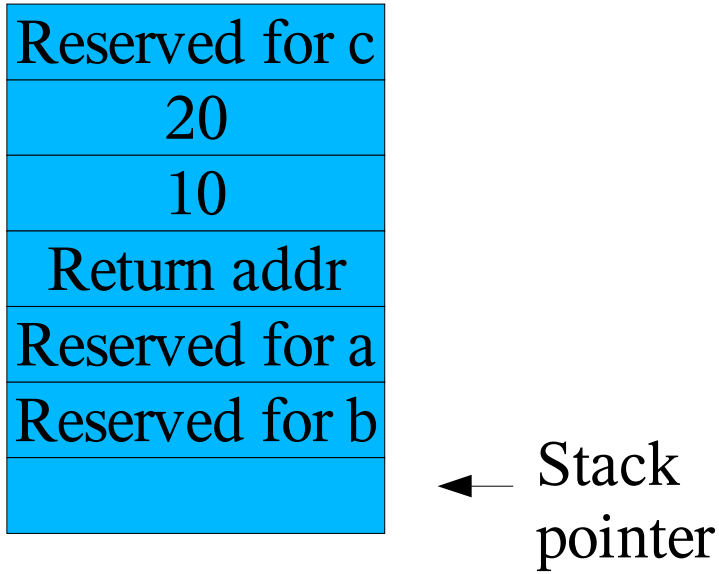
A simplified version of how the stack works:

Stack and function call

If we have the following code:

```
int add ( a,b) {
 int a,b ;
return (a+b) }

main () {
 int c ;
c=add (10,20) ;
}
```



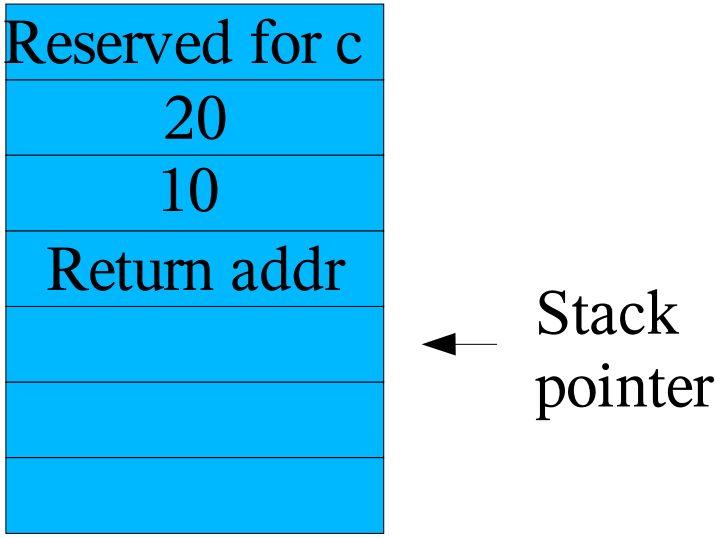
A simplified version of how the stack works:

Stack and function call

If we have the following code:

```
int add ( a,b) {
 int a,b ;
return (a+b) }

main () {
 int c ;
 c=add (10,20) ;
}
```



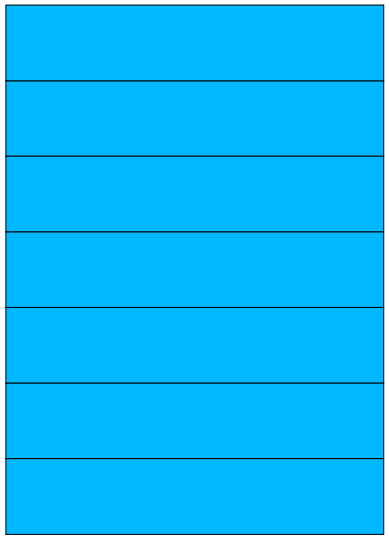
A simplified version of how the stack works:

Stack and function call

If we have the following code:

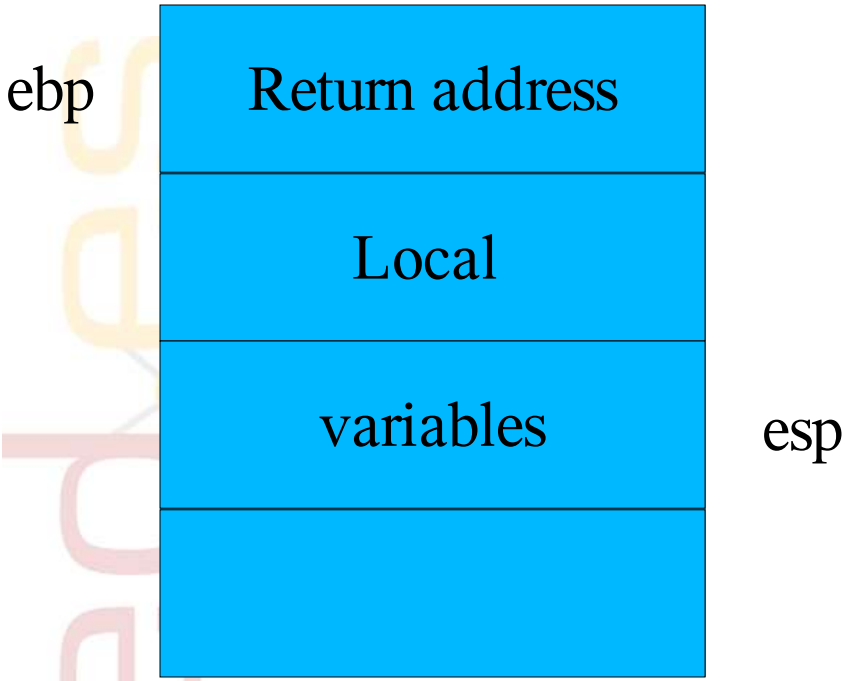
```
int add ( a,b) {
 int a,b ;
return (a+b) }

main () {
 int c ;
c=add (10,20) ;
}
```



A simplified version of how the stack works:

Stack frames



Inside a function ebs is used to keep track of the current stack value.

- At the start of the function ebp is pushed (stored in the stack)
- Ebp keep the esp value
- Esp is modified by local variables and calls to other functions
- At the end esp is loaded with the value of ebp
- Ebp is recovered (pop) from the stack
- The function return

Stack frames

Example function:

```
push %ebp
mov %esp, %ebp
sub -20, %esp
.......
......
 mov %ebp , %esp
 pop %ebp
retn
```

Inside a function ebs is used to keep track of the current stack value.

- At the start of the function ebp is pushed (stored in the stack)
- Ebp keep the esp value
- Esp is modified by local variables and calls to other functions
- At the end esp is loaded with the value of ebp
- Ebp is recovered (pop) from the stack
- The function return

Some ASM examples

How does a loop look in ASM ?

Typical loop:

```
for (n=10 ; n!=0 ; n--) { code}
```

```
Start:  mov $10,  4(%ebp)
loop: cmp 4(%ebp),0
        jg code
       jmp end_loop
code:   loop code.
        .........
        dec 4(%ebp)
       jmp loop

end_loop:
```

ASM example II

Normal condition;

```
if (a !=0) then {do1}
   else { do2}
```

(sometimes)

```
if (c=fopen(.....) !=0) ...
```

```
Cmp $0, -4(%ebp)
jne do1
do2:
.....
do1: code of d1
```

Depending on the optimization and the condition the condition can be reversed.

Asmdump,

The basic disassembler is shipped with most development tools, part of the "binutils" package: objedump.

We are going to use a small perl wrapper around this program that provide some additional help:

- Add comments and references to the strings found in the assembly text.
- Replace the address of the function with their name, for easy reading of the code.

This small script is part of another program called LDASM, but asmdump works in text mode only , browsing the output code in your favorite editor.

Note about ASM syntaxis

AT&T	Intel
Used mainly in the Unix World	Mostly used in the Windows world
Register are prefixed with “%” and values with $	No prefixes for register of value references
The first operand is the source and the second the destination	First operand is the destination and first one is the source
The base register is enclosed by “(“ and “)”	The base register is enclosed by “[“ and “]”
Suffix for operand sizes (l= long, w=world, b= byte)	Additional directives for use with memory operand byte ptr…
Indirect addressing take form of %segment:disp(base, index,scale)	Indirect addressing takes the form of segment:[base+index*scal+disp]

(Gnu)libc : Standard C library

- GnuLibC provides the most common functions for C (and other languages) interfaces: printf, fopen, .. all this functions are implemented in the glibc library.ç
- Some of the functions are translated as call to the operating system (read, fopen), but other are user space related (printf, scanf, etc).
- You can use the “ldd “ command in linux to list the dinamic linked libraries used by a program.
- We are dealing with dynamic linked binaries so all this function are “linked” at run time in the binary, so in the disassembled source code we will not see the code for the function only the call to it.

Searching for the main function

Searching for the main function.

- With the non-stripped binaries you will found a “main function declared in the disassembled output, this is were the programs starts.
- For stripped binaries:
  - One important function provided by glibc is the “__libc_start_main” , that is used to initialize the program and call the main function .
  - This is a dinamic function so is allways imported in normal stripped binaries.
  - The first argument to this function is the address in which the main program start

ASM: information hidding I

The forensic tool “strings” is very useful but:

- Show only strings of more than four characters (by default) so if the strings has less that this number is not displayed.
- Some rootkit could split longs strings in small pieces to hide the configuration files.
- What happened if the rootkit uses some kind of encryption of the strings ?

- You can search for pattern in the code (similar instructions repeated instruction
- Also calls to “sprintf” to join small strings in one big strings.

ASM information hidding:

Some intruder are using the binary compressor (UPX) to reduce the size of binaries and also hide strings.

- The files with the strings “This file has been compresed by UPX..” are UPX compressed.
  - Can be used as rootkit test:: no normal distribution ships compressed upx binaries.
- UPX (In linux) wrote the uncompressed file in the /tmp directory , run and after running delete it, so you can recover the files from this directory.
- You can also use UPX to decompress the file, but some files are modified to not been recognized as compressed files by upx.
- You always trace the files with a debugger (gdb) or use memory dumps to analyze the binary.

Exercise 1 : Small code analysis

Objective: Test the skills explained about source code disassembly with some C code examples.

- The example include are some easy to understand C program code.
- Copy to your home directory the source code files, compile them and after disassembly them view the assembly code with an editor.

So it's your turn:

Exercise 2.

Linux binaries from two rootkis:

- Linux-1 : Analyze some linux trojan files, trying to reverse engineering how they works.

- Linux-2: Try to detect the hidden strings in linux rootkit, the strings are hidden so they are not directly show with the strings command.

Binaries from other platforms

- Sparc architecture is different from intel.
  - Register are used to exchange arguments in function calls.
  - Require alignment of instructions so some “nop” (nulls) are inserted in the code.
  - Optimization used to change the order of some instructions
- Two examples
  - Sun-1: Try to find the configuration files in a solaris rootkit
  - Sun-2: Same, but the binary has the configuration file hidden

The 1st Forensic Challenge in Spanish

Organized last year by RedIRIS, with collaboration from:

- UNAM-CERT (México)
- CAIS/RNP (Brasil)
- SANS
- Spanish Security Experts

Prices donated from some companies:

- Encase (Guidance Software)
- SANS documentation

Forensic analysis of a compromised Linux Machine (Honeypot)

- No special intrussion, simple one.

Small diffusion security lists in Spanish

Participants must provide:

- ❑ Anonymous “executive” report describing the incident.
- ❑ Technical report (60 pages max.) analyzing the incident.

It was very important that the report describes not only what the intruder has done, but also how the investigator has found it,

Describing the tools used in the analysis .

Correlating information from different sources.

Objective:

- ❑ Promote computer forensic “awareness” in our constituencies
- ❑ Provide a “learn by example” repository of forensic analysis in Spanish that could be useful to users that want to learn about computer forensic.

At the end: 14 participants send their reports that are published in the web pages

High quality of all the reports

Quality of reports

- **All reports obtained more than 5 points (if use a scale of 10 point for the best papers, the "fewer" was 6,2 points).**

Most users provided graph and statistics of the intrusion

Different techniques and tools used:

- Sleutkit
- TCT
- But also :
  - Home made tools
  - Repositories of MD5 files
- BSD accounting from the compromised system
- SWAP memory cumps

2nd Forensic Challenge in Spanish

Promoted this year by UNAM-CERT (México), wth some help from RedIRIS, CAIS, RNP,

Initial phase:

- Published in Security list and also with some press information
- Ask people to register to know the number of people interested in the challenge

The Challenge started on 1st of Feb, with the download of the image.

- More than 900 users have been registered:
  - If only 5% of users presents a report it would be about 50 new forensic reports.

The result of the challenge would be announced in April , with a official price award in the UNAM security conference (May)

Our Opinion about the challenge:

- Very interesting to promote some techniques (computer forensic in your constituency)
- Reports are in people's mother tongue, so they reach more people that other information written in English.
- Provide a good example library that can be used by other user that want to learn about this matter.

So....

for the next TC:

- Why not organize a forensic challenge inside TC first members ?

An small example

As an small example for you:

- There are different machines at Ipaddress: XXX.XXX.,XXX,X compromised with different rookits.
- Connect to them (root password if first) and try to find the rootkit and how they works.
- Use the tools that are now available, don't start with the copy of the filesystem :-)