Defina até onde um serviço pode ir

Regras de conduta

POSIX Capabilities é um recurso do kernel que permite limitar o que cada serviço pode fazer, reduzindo as conseqüências de um ataque.

por Marlon Luis Petry

O superusuário *root* pode realizar qualquer operação sem restrições, o que muitas vezes é mais que o necessário para executar um serviço. Isso é preocupante se lembrarmos que todos os dias novos *bugs* são encontrados e novos *exploits* são desenvolvidos, possibilitando um ataque do tipo *buffer-overflow* nesses serviços.

Um dos modos para minimizar os efeitos de ataques desse tipo, nos quais o atacante pode conseguir abrir uma sessão do shell, é executar os serviços utilizando um usuário sem privilégios de administrador. Dessa forma, o acesso conseguido usando determinada falha terá alcance restrito aos privilégios do serviço em questão.

A partir da versão 2.1 do Kernel, surgiu o conceito de "capacidades", ou *POSIX capability*, cujo princípio é dividir os privilégios do root em um conjunto de capacidades [1], cobrindo todos os privilégios do super usuário. Por exemplo, quando vamos alterar a hora do sistema, é necessária a capacidade `CAP_SYS_TIME`. Atribuindo essa capacidade ao comando `date`, qualquer usuário poderá alterar a hora do sistema. Esse recurso entrou no Kernel oficial a partir da versão 2.6.24rc2.

Pré-Requisitos

Para verificar se o kernel está compilado com suporte às POSIX capabilities, execute o comando `zgrep '\(XATTR\|CAPA\)' /proc/config.gz` e verifique a sua saída:

```
CONFIG_EXT2_FS_XATTR=y
CONFIG_EXT3_FS_XATTR=y
CONFIG_EXT4DEV_FS_XATTR=y
CONFIG_REISERFS_FS_XATTR=y
# CONFIG_JFFS2_FS_XATTR is not set
CONFIG_CIFS_XATTR=y
CONFIG_SECURITY_CAPABILITIES=y
CONFIG_SECURITY_FILE_
➥CAPABILITIES=y
```

Caso o resultado seja diferente do mostrado, é necessário habilitar essas opções no kernel e recompilá-lo.

Também é necessário possuir a biblioteca *libcap2*, que pode ser baixada no endereço [2] e compilada da forma tradicional:

```
tar -xzf libcap-2.11.tar.gz
cd libcap-2.11
make
make install
```

Ou, se você for usuário da distribuição *Gentoo*:

```
emerge -av =sys-libs/libcap-2.11
```

Privilégios mínimos

Normalmente, o comando `ping` necessita estar com o bit *SUID* habilitado, tendo como dono o usuário root para que um usuário comum consiga executar o comando. Então, vamos remover o bit SUID e colocar a capacidade mínima necessária para executar o comando.

```
chmod u-s /bin/ping
```

```
ping 127.0.0.1
ping: icmp open socket: Operation
➥not permitted
```

Como podemos verificar, removendo o bit SUID do ping, um usuário comum não pode executá-lo. Para tornar a execução possível, atribuiremos a capacidade *CAP_NET_RAW* ao comando `ping`. Essa capacidade permite que um usuário comum abra conexões de rede do tipo *raw* (bruto):

```
setcap cap_net_raw=ep /bin/ping
```

Agora conseguimos executar o comando `ping` com usuário comum sem que o comando esteja com o SUID habilitado:

```
ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84)
➥bytes of data.
64 bytes from 127.0.0.1: icmp_
➥seq=1 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_
➥seq=2 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_
➥seq=3 ttl=64 time=0.035 ms
--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received,
➥0% packet loss, time 1998ms rtt
➥min/avg/max/mdev =
0.033/0.033/0.035/0.006 ms
```

Para verificar quais capacidades estão definidas, basta utilizar o comando `getcap` como segue:

```
getcap /bin/ping
```

```
/bin/ping = cap_net_raw+ep
```

Necessidades

No artigo "*POSIX file capabilities: Parceling the power of root* "[3], o autor fornece o código-fonte de um módulo chamado `capable_probe`, que nos ajuda a descobrir as necessidades de cada processo. Sempre que houver uma chamada de sistema para a função `cap_capable()`, ele a intercepta e substitui pela função `cr_capable()`, que mostrará a capacidade exigida e qual processo a está requisitando.

Para utilizar esse módulo, use o comando `zgrep '\(PROBE\)' /proc/config.gz` para verificar a compatibilidade do kernel:

```
zgrep '\(PROBE\)' /proc/config.gz
CONFIG_GENERIC_IRQ_PROBE=y
CONFIG_KPROBES=y
CONFIG_KRETPROBES=y
CONFIG_HAVE_KPROBES=y
CONFIG_HAVE_KRETPROBES=y
# CONFIG_NET_DCCPPROBE is not set
# CONFIG_NET_TCPPROBE is not set
# CONFIG_MTD_JEDECPROBE is not set
CONFIG_KPROBES_SANITY_TEST=y
```

Verifique se as opções mostradas correspondem às mostradas. Em seguida, o módulo `capable_discovery` pode ser instalado da maneira tradicional:

```
tar -xjvf capable_discovery.tar.
➥bz2
cd capable_discovery
make
make install
```

Carregue o módulo com o comando `modprobe capable_discovery`. As mensagens mostradas pelo módulo ao usar o `ping` podem ser monitoradas utilizando o comando `tail -f /var/log/messages | grep ping`, que exibirá a saída:

```
Sep  1 21:35:23 localhost
➥capability 21=CAP_SYS_ADMIN for
➥ping
Sep  1 21:35:23 localhost
➥capability 13=CAP_NET_RAW
➥for ping
Sep  1 21:35:23 localhost
➥capability 7=CAP_SETUID for ping
```

No resultado acima podemos ver de quais capacidades o `ping` necessita:

- *21: CAP_SYS_ADMIN*: Não atribuir, corresponde à capacidade de administração do sistema.
- *13: CAP_NET_RAW*: Permite a comunicação de dados brutos pela rede.
- *7: CAP_SETUID*: Necessária para manipulação de arquivos.

Samba sem root

Antes de fazermos o samba rodar com um usuário comum, temos que descobrir quais são as capacidades exigidas pelo serviço. Para descobrir essas capacidades, usaremos novamente o módulo `capable_discovery`:

```
# modprobe capable_discovery
# tail -f /var/log/messages |grep
➥smbd
```

Em outro terminal, inicie o servidor Samba:

```
# smbd -d
```

No primeiro terminal, podemos verificar as capacidade exigidas pelo Samba:

```
Sep  3 13:59:51 localhost
➥capability 21=CAP_SYS_ADMIN for
➥smbd
Sep  3 13:59:51 localhost
➥capability 7=CAP_SETUID for smbd
Sep  3 13:59:52 localhost
➥capability 24=CAP_SYS_RESOURCE
➥for smbd
Sep  3 13:59:51 localhost
➥capability 6=CAP_SETGID for smbd
Sep  3 13:59:52 localhost
➥capability 10=CAP_NET_BIND_SERVICE
➥for smbd
```

Repetimos o mesmo procedimento para verificar quais são os requisitos do `nmbd`, obtendo como resposta:

```
Sep  3 14:00:03 localhost
➥capability 21=CAP_SYS_ADMIN for
➥nmbd
Sep  3 14:00:03 localhost
➥capability 10=CAP_NET_BIND_
➥SERVICE
➥for nmbd
```

Por fim, o módulo pode ser descarregado com `rmmod capable_discovery`. É interessante remover o módulo logo após o uso, pois todos os processos que estão rodando sempre estão verificando as capacidades necessárias, haja vista que o módulo utiliza a função `printk` para mostrar as capacidades e isso causa um grande aumento dos arquivos de log.

Prosseguimos criando um usuário e grupo específicos para rodar o Samba:

```
# groupadd samba
# adduser samba -g samba
```

Há diretórios que pertencem ao Samba que também precisam ter dono e grupo alterados:

```
chown samba:samba -R /var/run/samba/
chown samba:samba -R /var/log/
➥samba/
chown samba:samba -R /etc/samba/
chown samba:samba -R /var/cache/
➥samba/
chown samba:samba -R /var/lib/
➥samba/
```

Para evitar que outros usuários possam tentar executar os comandos, alteramos dono e permissão de execução:

```
chown samba:samba /usr/sbin/nmbd
chown samba:samba /usr/sbin/smbd
chmod g-x,o-x /usr/sbin/smbd
chmod g-x,o-x /usr/sbin/nmbd
chmod u+s /usr/sbin/nmbd /usr/
➥sbin/smbd
ls -la /usr/sbin/*bd
-rwsr--r-- 1 samba samba 1061664
➥Mar 10 22:54 /usr/sbin/nmbd
-rwsr--r-- 1 samba samba 3633876
➥Mar 10 22:54 /usr/sbin/smbd
```

Finalmente, as capacidades necessárias são atribuídas:

```
setcap  cap_net_bind_service,cap_
➥sys_resource=ep /usr/sbin/smbd
setcap  cap_net_bind_service=ep /
➥usr/sbin/nmbd
```

Certifique-se de interromper o serviço e reinicie-o usando o usuário *samba*:

```
# su - samba
$ /usr/sbin/smbd -D
$ /usr/sbin/nmbd -D
```

O comando `ps -uax | grep samba` mostrará o serviços em execução com o usuário samba:

```
samba     9535  0.0  0.2   8876
➥2196 ?        Ss   15:30   0:00 /
➥usr/sbin/smbd -D
samba     9536  0.0  0.0   8876
➥948 ?         S    15:30   0:00 /
➥usr/sbin/smbd -D
```

Neste momento, o samba já funciona com usuário comum utilizando as capacidades que atribuímos aos binários do serviço.

Cada um no seu quadrado

As vantagens de rodar um serviço com alcance restrito são inegáveis, mas todo o procedimento de descobrir as capacidades mínimas e atribuí-las aos serviços não é tarefa das mais triviais. Porém, como toda inovação, não tardará a ser incorporada como recurso padrão nas distribuições mais populares. ■

Mais informações

[1] Lista Capacidades: http://www.gentoo.org/proj/en/hardened/capabilities.xml

[2] Download libcap2: http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.6/libcap-2.11.tar.gz

[3] Linux Capabilities: making them work. http://ols.fedoraproject.org/OLS/Reprints-2008/hallyn-reprint.pdf

[4] Módulo personalizado: http://petryx.blogrs.com.br/capable_discovery.tar.bz2

[5] POSIX File Capabilities: http://www.friedhoff.org/posixfilecapsold.html

[6] Introduction to Linux Capabilities and ACL's: http://www.securityfocus.com/infocus/1400

[7] Secure programmer: Minimizing privileges: http://www.ibm.com/developerworks/linux/library/l-sppriv.html

[8] Linux kernel capabilities FAQ: ftp://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt

[9] POSIX file capabilities: Parceling the power of root: http://www.ibm.com/developerworks/library/l-posixcap.html

Sobre o autor

Marlon Luis Petry é bacharel em Ciência da Computação graduado pela Unicruz e trabalha com servidores Linux e consultoria para pequenos provedores de Internet, além de manter o blog http://petryx.blogrs.com.br