

Linux Privilege Escalation

目次

- ・話すこと、話さないこと

- ・検証環境

- ・各項目について

- Sudoers
- SUID, SGID
- Capability
- Cronjob
- Systemd Timer Unit
- PATH ENV
- NFS no_root_squash
- Wild card injection

- ・有効なツール群

・話すこと

- Linuxの設定ミスを利用した権限昇格の方法
- それを防ぐために
- 権限昇格パスの探索に有効なツール

・話さないこと

- 初期侵害について(今日の説明は一般ユーザーのシェルを獲得している前提)
- 既知の脆弱性を利用した権限昇格

検証環境

Windows

Virtual Box(10.10.10.0/24)

Attacker

Parrot Security
(10.10.10.10/24)



Victim

Ubuntu
(10.10.10.20/24)



Sudoers

Sudoers

root権限を一般ユーザーに委譲？貸出？するための仕組み

特定のユーザーが特定のコマンドを実行するときのみroot権限を与えたり。

デフォルトでは『/etc/sudoers』に設定、要root権限

💀 visudo以外で編集してミスった場合、sudoが使えなくなり死ぬ💀

```
/etc/sudoers:40:9: 構文エラー
abcd aaa
^
次は何でしょうか？
オプション:
e -- sudoers ファイルを再度編集します
x -- sudoers ファイルへの変更を保存せずに終了します
Q -- sudoers ファイルへの変更を保存して終了します (*危険です!*)
次は何でしょうか？ █
```

Sudoers

『sudo -l』コマンドでもカレントユーザーに対する設定を確認できる

ただし、カレントユーザーに対して『NOPASSWD』の権限(後述)が一つも設定されていない場合、パスワードを要求される

Reverse Shell等で初期侵害した場合、パスワードは手に入っていないのでSudoersを利用した権限昇格は期待できない

正しいパスワードを入力してもSudoersに設定がされてないと怒られる

```
john@For-Security-Test
└─ sudo nmap -sN 10.10.10.10
[sudo] john のパスワード:
john は sudoers ファイル内にありません。この事象は記録・報告されます。
```

Sudoers

Tag Spec: 全部で14種類あるらしい

```
root ALL=(ALL:ALL) NOPASSWD: ALL
```

ユーザーは※1

このホストで

このユーザーとして

このグループとして

パスワード無しで

このコマンドを実行できる

※1『%』が頭に付くとグループの指定になります

Sudoers

```
john    ALL=(bob) NOPASSWD: /usr/bin/vim, /bin/less
```

訳

johnは全てのホスト上で、bobとしてパスワード無しでvimとlessを実行できる

Sudoers

ペネトレーションテストで一般ユーザーのシェルを獲得することができた

『sudo -l』コマンドを実行すると下記の出力が得られた

```
john@For-Security-Test
└─ sudo -l
既定値のエントリと照合中 (ユーザー名 john) (ホスト名 ip-10-0-1-249):
!visiblepw, always_set_home, match_group_by_gid, always_query_group_plugin, env_reset,
env_keep="COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS", env_keep+="MAIL PS1 PS2
QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE", env_keep+="LC_COLLATE LC_IDENTIFICATION
LC_MEASUREMENT LC_MESSAGES", env_keep+="LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER
LC_TELEPHONE", env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS _XKB_CHARSET XAUTHORITY",
secure_path="/sbin\:/bin\:/usr/sbin\:/usr/bin

ユーザー john は ip-10-0-1-249 上で コマンドを実行できます
(root) NOPASSWD: /usr/bin/nmap
```

nmapの要root権限スキャンを実行するため？

ここからの権限昇格パス？

Sudoers

実践

nmap

Sudoers

- ・最小権限の法則

ALLを使用するのは基本的に避ける

- ・NOPASSWDは使用しない

許可したコマンドから任意コマンドを実行できると即権限昇格パスに

- ・各ユーザーを/etc/sudoersに記載するのが億劫な場合は、

新規グループを作成し、グループに対してsudoersを設定するのがいいと思う

SUID, SGID

SUID,SGID

パーミッションの一種 Set User ID, Set Group ID

ファイルやディレクトリに設定する権限

ls -l で確認できる

```
parrot@parrot-virtualbox:/  
> ls -l  
合計 40K  
lrwxrwxrwx 1 root root 7 5月 4 2022 bin -> usr/bin  
drwxr-xr-x 1 root root 258 8月 1 09:44 boot  
drwxr-xr-x 17 root root 3.1K 11月 24 10:57 dev  
drwxr-xr-x 1 root root 5.4K 11月 22 13:28 etc  
-rw-r--r-- 1 root root 0 9月 9 17:45 flag  
drwxr-xr-x 1 root root 12 8月 1 09:42 home
```

SUIDがついているとこうなる

```
> ls -l /home/parrot/bash  
-rwsrwxrwx 1 root root 1.2M 10月 28 16:43 /home/parrot/bash
```

SUID,SGID

これが付与されていると...

SUID → ファイルを所有者の権限で実行する

SGID → ファイルを所有グループの権限で実行する

ディレクトリに付与した場合、子に所有グループが継承される

ここに注目



SUID,SGID

通常、実行可能なスクリプトやバイナリは実行したユーザーの権限で動作するが、SUID,SGIDがあることで...

- ・一般ユーザーが『passwd』コマンドでパスワードの変更ができる

『passwd』は『/etc/shadow』へのアクセスが発生するため、一般ユーザー権限では×

- ・一般ユーザーが『ping』コマンドでICMPパケットの送信ができる

raw packetを操作するためには管理者権限が必要なため、一般ユーザー権限では×

etc.

SUID,SGID

SUID,SGIDは『chmod』コマンドで設定する

SUIDの設定

『chmod +4000 /app/example.sh』 or 『chmod u+s /app/example.sh』

SGIDの設定

『chmod +2000 /app/example.sh』 or 『chmod g+s /app/example.sh』

SUID,SGID

SUID, SGIDが設定されたファイルを検索

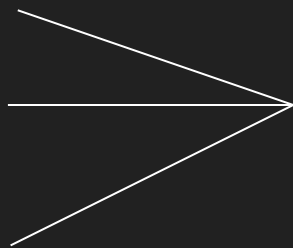
```
find / -user root -perm -4000 -o -perm -2000 -type f 2>/dev/null
```

SUID,SGID

SUID(SGID)が設定されている

所有者(所有グループ)がroot

任意のコマンド実行 or ファイル
の読み書き



root権限への昇格が可能!!

SUID,SGID

一般ユーザーのシェル獲得後、

『find / -perm -4000 -o -perm -2000 -type f 2>/dev/null』(-perm /6000でも可)コマンドを実行すると下記の出力が得られた。

```
/usr/bin/gpasswd  
/usr/bin/chsh  
/usr/bin/cp  
/usr/bin/newgrp  
/usr/bin/fusermount3
```

権限昇格パスは？

Sudoers

实践

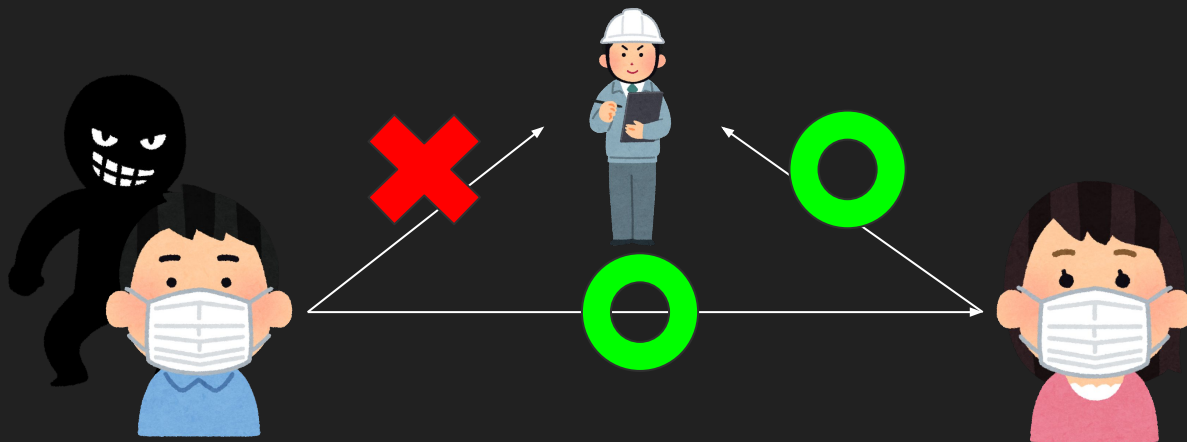
cp

SUID,SGID

- ・任意コマンドが実行できる、任意ファイルの読み書きが可能等のファイルにSUIDを付与しない

rootが所有者でなくとも水平権限移動が可能になる

機能としてコマンドが実行できない場合でもBoFなどの脆弱を利用される可能性もある



Capability

Capability

プロセスの権限をより細分化する仕組み

pingコマンドって

Raw Socket(生パケット)をいじるから

root権限での実行が必要だけど、

root権限をまるごと付与する必要ある？



Capability

実行ファイルに対して特定の機能のみを実行できる権限を付与できる

- ・RawSocket使いたい → CAP_NET_RAW
- ・1024未満ポートでHTTPサーバー立てたい → CAP_NET_BIND_SERVICE

etc...

Capability

定義されているCapabilityの一覧は『/usr/include/linux/capability.h』で確認できる

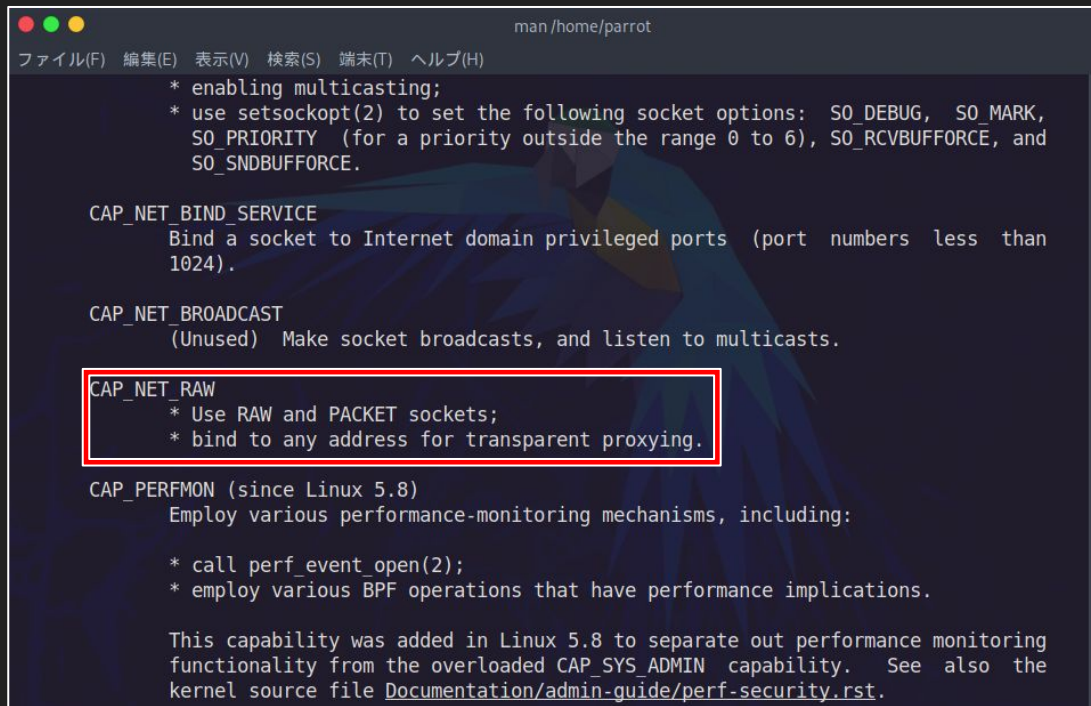
```
parrot@parrot-virtualbox:~  
> cat /usr/include/linux/capability.h | grep "#define CAP_"  
#define CAP_CHOWN 0  
#define CAP_DAC_OVERRIDE 1  
#define CAP_DAC_READ_SEARCH 2  
#define CAP_FOWNER 3  
#define CAP_FSETID 4  
#define CAP_KILL 5  
#define CAP_SETGID 6  
#define CAP_SETUID 7  
#define CAP_SETPCAP 8  
#define CAP_LINUX_IMMUTABLE 9  
#define CAP_NET_BIND_SERVICE 10  
#define CAP_NET_BROADCAST 11  
#define CAP_NET_ADMIN 12  
#define CAP_NET_RAW 13  
#define CAP_IPC_LOCK 14  
#define CAP_IPC_OWNER 15
```

検証用VMのLinuxでは44個定義されている

```
parrot@parrot-virtualbox:~  
> cat /usr/include/linux/capability.h | grep "#define CAP_" | wc -l  
44
```

Capability

各Capabilityがどのように作用するかは『man 7 capabilities』で確認できる



```
man /home/parrot
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)

* enabling multicasting;
* use setsockopt(2) to set the following socket options: SO_DEBUG, SO_MARK,
SO_PRIORITY (for a priority outside the range 0 to 6), SO_RCVBUFFORCE, and
SO_SNDBUFFORCE.

CAP_NET_BIND_SERVICE
Bind a socket to Internet domain privileged ports (port numbers less than
1024).

CAP_NET_BROADCAST
(Unused) Make socket broadcasts, and listen to multicasts.

CAP_NET_RAW
* Use RAW and PACKET sockets;
* bind to any address for transparent proxying.

CAP_PERFMON (since Linux 5.8)
Employ various performance-monitoring mechanisms, including:

* call perf_event_open(2);
* employ various BPF operations that have performance implications.

This capability was added in Linux 5.8 to separate out performance monitoring
functionality from the overloaded CAP_SYS_ADMIN capability. See also the
kernel source file Documentation/admin-guide/perf-security.rst.
```

Capability

Capabilityの設定 → `setcap cap_net_raw=+ep /path/to/file` (要root権限)

Capabilityの削除 → `setcap -r /path/to/file`(要root権限)

Capabilityの確認 → `getcap -r / 2>/dev/null`

```
parrot@parrot-virtualbox:~/Desktop
> getcap -r / 2>/dev/null
/usr/bin/dumpcap cap_net_admin,cap_net_raw=eip
/usr/bin/fping cap_net_raw=ep
/usr/bin/gnome-keyring-daemon cap_ipc_lock=ep
/usr/bin/ping cap_net_raw=ep
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper cap_net_bind_service,cap_net_admin=ep
```

16:33

Capability

File Capability

- ・ Permitted(許可)
- ・ Inheritable(子への継承)
- ・ Effective(実効)

Thread Capability

- ・ Permitted
- ・ Inheritable
- ・ Effective
- ・ Ambient(since Linux 4.3)

Capabilityは死ぬほど複雑なので簡単に

カーネルがCapabilityを判定するのに使用する EffectiveのCapabilityが設定されているファイルを探す

Capability

特にわかりやすく注意すべきだと思ってるCapability

- `cap_setuid`
- `cap_dac_override`
- `cap_dac_read_search`

Capability

cap_setuid

自プロセスの実ユーザーIDと実効ユーザーIDを変更できる

ユーザーIDを変更できるということは0を指定することでrootに昇格できる

pythonなどのインタプリタについているとやばい

Capability

cap_dac_override

ファイルの読み出し、書き込み、実行の権限チェックをバイパスする

ほぼroot

Capability

cap_dac_read_search

ファイルの読み出し権限のチェックとディレクトリの読み出しと実行※1 の権限チェックをバイパスする

CTFだとバックアップ用のプログラムについていることが多い？

※1 ディレクトリの実行権限 = ディレクトリ内に移動できる

Capability

实践

python3 (cap_setuid)

Capability

- ・SudoersやSUIDと一緒にとにかく最小権限の原則
- ・Capabilityを付与したファイルから任意にプロセスが生成できたり、コマンドが実行できるような仕様、またはそのような脆弱性が無いことを確認

Cronjob

Cronjob

定期的にコマンドを実行するための仕組み

設定ファイルに記載されたコマンドを設定したインターバルで実行する

Cronjob

パス	オーナー	説明
/etc/crontab	root	メインの設定ファイル
/etc/cron.monthly	root	月次実行ジョブを配置するディレクトリ
/etc/cron.weekly	root	週次実行ジョブを配置するディレクトリ
/etc/cron.daily	root	日次実行ジョブを配置するディレクトリ
/etc/cron.hourly	root	毎時実行ジョブを配置するディレクトリ
→ /var/spool/cron/crontabs/user名	各ユーザー	各ユーザーのジョブ設定ファイル crontab -u USER -e で生成される
/etc/cron.d/*	root	上記以外のジョブ設定ファイルを配置するディレクトリ

Cronjob

設定ファイルの構文

linux manualから抜粋

* ← 全ての値にマッチ

* / 2 ← 2単位時間毎

```
SHELL=/bin/bash
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
MAILTO=root
```

```
HOME=/
```

```
# For details see man 4 crontabs
```

```
# Example of job definition:
```

```
# .----- minute (0 - 59)
```

```
# | .----- hour (0 - 23)
```

```
# | | .----- day of month (1 - 31)
```

```
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
```

```
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR
```

```
sun,mon,tue,wed,thu,fri,sat
```

```
# | | | |
```

```
# * * * * * user-name command to be executed
```

分 時 日 月 曜日 ユーザー名 コマンド

Cronjob

Cronjobに誰でも編集が可能なスクリプトがroot権限で実行されている場合、スクリプトファイルを書き換えることで管理者権限への昇格が可能

```
* * * * * root /tmp/do-something.sh
```



このファイルに書き込み権限があるとまずい

Cronjob

实践

cronjob reverse shell

Cronjob

- ・スクリプトファイルを実行する場合、そのスクリプトファイルの書き込み権限はCronjobの実行ユーザー以外に付与しない
- ・直接的な対策ではないが/etc/cron.allowや/etc/cron.denyを活用し、cronを使用できるユーザーを絞る

cron.allow	cron.deny	実行可能ユーザー
ファイル無し	ファイル無し	全てのユーザー
ファイル有り	無視される	cron.allow記載ユーザーのみ
ファイル無し	ファイル有り	cron.denyに記載が無いユーザーのみ
空ファイル	無視される	無し
ファイル無し	空ファイル	全ユーザー

Cronjob(オマケ)

★PSPY

<https://github.com/DominicBreuker/pspy>

Cronの設定ファイルに読み取り権限がなく、どんなコマンドを実行しているのか確認できない場合によくお世話になるOSS

(/var/spool/cron/crontabs/[USER]ディレクトリに設定ファイルがあると所有者以外では読み取れない)

Watching recursively : [/usr /tmp /etc /home /var /opt] (6)

Watching non-recursively: [] (0)

Printing: processes=true file-system events=false

2018/02/18 21:00:03 Inotify watcher limit: 524288 (/proc/sys/fs/inotify/max_user_watches)

2018/02/18 21:00:03 Inotify watchers set up: Watching 1030 directories - watching now

2018/02/18 21:00:03 CMD: UID=0 PID=9 | cron -f

2018/02/18 21:00:03 CMD: UID=0 PID=7 | sudo cron -f

2018/02/18 21:00:03 CMD: UID=1000 PID=14 | pspy

2018/02/18 21:00:03 CMD: UID=1000 PID=1 | /bin/bash /entrypoint.sh

2018/02/18 21:01:01 CMD: UID=0 PID=20 | CRON -f

2018/02/18 21:01:01 CMD: UID=0 PID=21 | CRON -f

2018/02/18 21:01:01 CMD: UID=0 PID=22 | python3 /root/scripts/password_reset.py

2018/02/18 21:01:01 CMD: UID=0 PID=25 |

2018/02/18 21:01:01 CMD: UID=??? PID=24 | ???

2018/02/18 21:01:01 CMD: UID=0 PID=23 | /bin/sh -c /bin/echo -e "KI5PZQ2ZPWQXJKEL\nKI5PZQ2ZPWQ

2018/02/18 21:01:01 CMD: UID=0 PID=26 | /usr/sbin/sendmail -i -FCronDaemon -B8BITMIME -oem roc

2018/02/18 21:01:01 CMD: UID=101 PID=27 |

2018/02/18 21:01:01 CMD: UID=8 PID=28 | /usr/sbin/exim4 -Mc 1enW4z-00000Q-Mk

Systemd Timer Unit

Systemd Timer Unit

Cronの後発ジョブスケジューラ

モノトニックタイマー → 『システム起動後10分後』のように特定のイベントからの経過時間で発動することができるタイマー

リアルタイムタイマー → Cron同様定期的に発動するタイマー

Systemd Timer Unit

基本的に権限昇格パスはCronと一緒なのでデモは割愛

ただし追加で設定不備があれば自由自在かも

- ・『systemctl』系のコマンドがroot権限で実行できる
- ・『/etc/systemd/system/』以下に書き込み権限がある

自分でサービス作って起動等...

PATH環境変数

PATH環境変数

コマンドを実行するときに絶対パスを解決する仕組み

『ls』とか『cat』の実体は『/bin』とか『/usr/bin』ディレクトリにあったりする
シェルがPATH環境変数に設定されているパスから検索してくれている

現在設定されているPATHは『echo \$PATH』で確認できる

PATH環境変数

環境変数の追加はbashの場合『PATH=\$PATH:追加したいパス』で実行できる

・『PATH=\$PATH:追加したいパス』



・『PATH=追加したいパス』



PATH環境変数

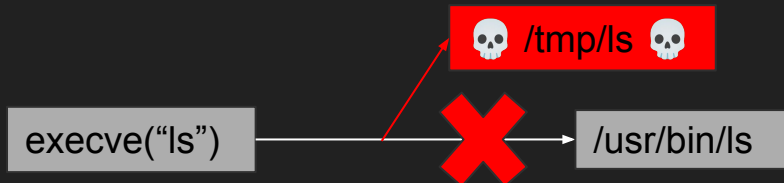
PATHの解決はPATHの先頭のディレクトリから行われる

```
[parrot@parrot-virtualbox]~  
$ echo $PATH  
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/usr/share/games:/usr/local/sbin  
:/usr/sbin:/sbin:/home/parrot/.local/bin:/snap/bin:/home/parrot/.local/bin:/snap/bin:/usr  
/sandbox/:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/usr/share/games:/usr/  
local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

/usr/local/bin/lsと/usr/bin/lsというコマンドが存在し、PATH環境変数が上記のような場合、『ls』は/usr/local/bin/lsに解決される

PATH環境変数

実行ファイル内で呼ばれるシステムコールで、
実行コマンドが絶対パスで記述されていない場合、
PATHの先頭に『/tmp』などworld writableなディレクトリを追加し、
そこに同名のスクリプトファイルや実行ファイルを配置することで
実行するシェルスクリプトを操作できる



PATH環境変数

SUIDが設定されている/opt/monitorというELFファイルを発見

実行するとなにやら情報が表示されるだけのプログラムのようなが...

PATH環境変数

実践

PATH環境変数

PATH環境変数

- ・シェルスクリプトや実行ファイル(のシステムコールやライブラリ関数)に記載するコマンドは絶対パスで記述する(根本的解決)
- ・シェルスクリプトや実行ファイルの権限を確認(SUID, SGID, Sudoers...)

NFS no_root_squash

NFS no_root_squash

NFSとは...

NFSとは、主にUNIX系OSで利用される分散ファイルシステム、および、そのための通信規約(プロトコル)

簡単に言えばネットワーク上に配置できる共有フォルダ

NFS no_root_squash

今回はnfs-kernel-serverを検証環境として使用

設定ファイル → /etc/exports (defaultでworld readable)

初期シェルを取っていればどのユーザーでも設定の確認が可能

```
# /etc/exports: the access control list for filesystems which may be exported
#                to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes      hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs5       gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
/srv/pe_share     10.10.10.0/24(rw,no_root_squash)
/srv/not_pe_share 10.10.10.0/24(rw)
```

NFS no_root_squash

外部からNFSの共有を確認するにはnmapのスク립トが便利

```
nmap --script=nfs-showmount {TARGET_IP}
```

metasploitもあります

```
scanner/nfs/nfsmount
```

NFS no_root_squash

/etc/exportsの構文は下記


share-name hostname or Network(options)

```
/srv/pe_share          10.10.10.0/24(rw,no_root_squash)
/srv/not_pe_share      10.10.10.0/24(rw)
```

NFS no_root_squash

💀 rwとno_root_squashの組み合わせが危険 💀

NFS no_root_squash

**Red Hat**
Customer Portal

Products & ServicesToolsSecurityCommunity

Support cases will be inaccessible Oct. 29 7:00-7:30am ET for planned maintenance. To create cases during this time, use techsupport@redhat.com or if critical, call Red Hat.

Products & Services > Product Documentation > Red Hat Enterprise Linux > 7 > セキュリティーガイド > 4.3.7.4. no_root_squash オプションを使用しないでください

4.2. Root アクセスの制御 >

4.3. サービスのセキュア化 ^

4.3.1. サービスへのリスク

4.3.2. サービスの識別と設定

4.3.3. 安全でないサービス

4.3.4. rpcbind のセキュア化 >

4.3.5. rpc.mountd のセキュア化 >

4.3.6. NIS のセキュア化 >

4.3.7. NFS のセキュア化 ^

4.3.7.1. ネットワークの注意深いプランニング

4.3.7.2. NFS マウントオプションのセキュア化 >

4.3.7.3. 構文エラーに注意

4.3.7.4. no_root_squash オプションを使用しないでください

4.3.7.5. NFS ファイルシステムの設定

Red Hat Training

A Red Hat training course is available for [Red Hat Enterprise Linux](#) >

4.3.7.4. no_root_squash オプションを使用しないでください

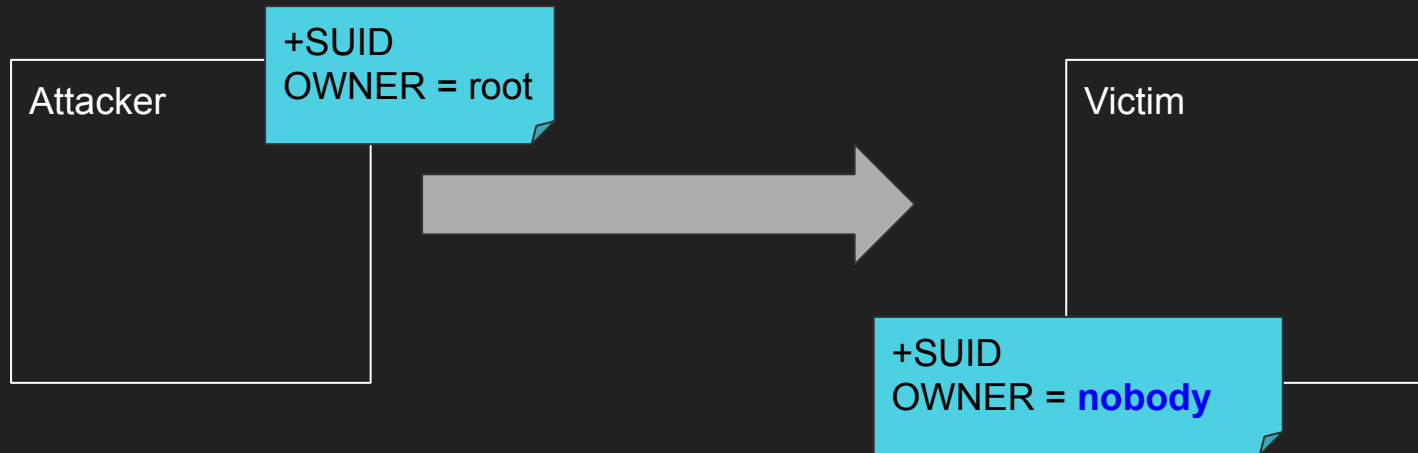
デフォルトでは、NFS 共有は、root ユーザーを非特権ユーザーアカウントである **nfsnobody** ユーザーに変更します。これにより、root で作成されたすべてのファイルの所有者が **nfsnobody** に変更され、setuid ビットが設定されたプログラムのアップロードができなくなります。

no_root_squash を使用すると、リモート root ユーザーは、共有ファイルシステム上の任意のファイルを変更し、他のユーザーが誤って実行するようにアプリケーションをトロイの木馬に感染したままにすることができます。

NFS no_root_squash

no_root_squashとはなんなのか

まずno_root_squash無しのデフォルトの挙動(all_squash,root_squash...)



NFS no_root_squash

検証結果

root_squashを設定したNFSを/tmp/not_peにマウントし、rootでファイル作成

```
[root@parrot-virtualbox]~[/home/parrot]
#id
uid=0(root) gid=0(root) groups=0(root)
[root@parrot-virtualbox]~[/home/parrot]
#touch /tmp/not_pe/test
[root@parrot-virtualbox]~[/home/parrot]
#ls -la /tmp/not_pe/test
-rw-r--r-- 1 nobody nogroup 0 10月 28 17:27 /tmp/not_pe/test
```

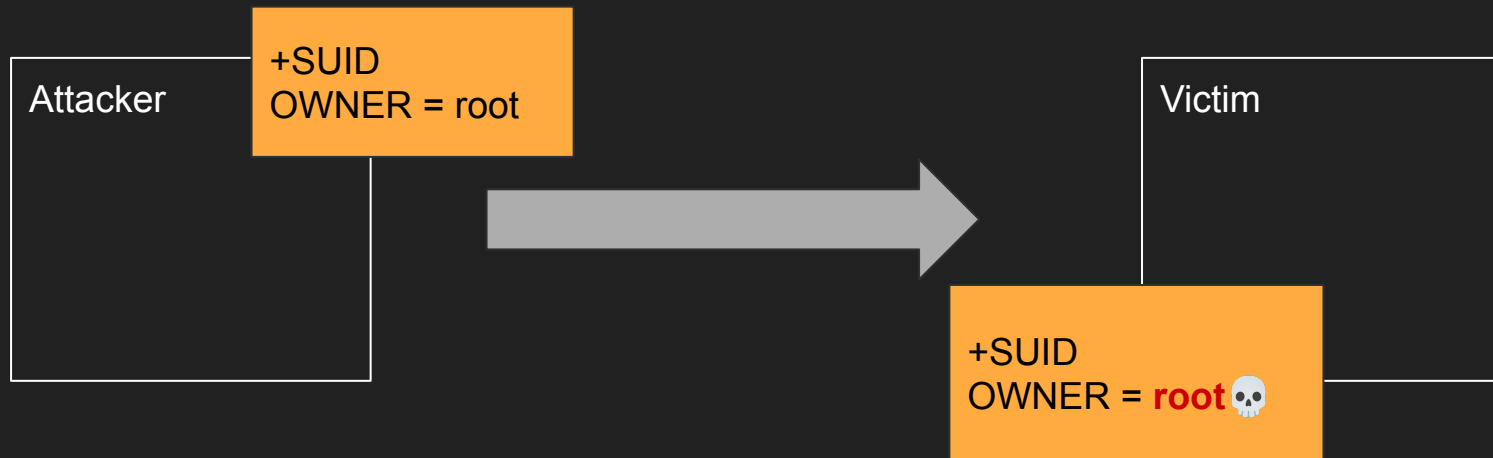

NFS no_root_squash

rootが所有者のファイルのmvは失敗するっぽい

```
[root@parrot-virtualbox]~# id
uid=0(root) gid=0(root) groups=0(root)
[root@parrot-virtualbox]~# touch ./suid.test
[root@parrot-virtualbox]~# chmod 4777 ./suid.test
[root@parrot-virtualbox]~# mv ./suid.test /mnt/not_pe/
mv: '/mnt/not_pe/suid.test' の所有者の保護に失敗しました: 許可されていない操作です
```

NFS no_root_squash

no_root_squashの挙動



NFS no_root_squash

検証結果

no_root_squashを設定したNFSを/tmp/peにマウントし、rootでファイル作成

```
[root@parrot-virtualbox]-[/home/parrot]
└─ #touch ./suid.test
[root@parrot-virtualbox]-[/home/parrot]
└─ #chmod 4777 ./suid.test
[root@parrot-virtualbox]-[/home/parrot]
└─ #mv ./suid.test /mnt/pe/
[root@parrot-virtualbox]-[/home/parrot]
└─ #ls -l /mnt/pe
合計 0
-rw-r--r-- 1 root root 0 10月 28 13:45 nfs-test.txt
-rwsrwxrwx 1 root root 0 11月 25 09:23 suid.test
-rw-r--r-- 1 root root 0 10月 28 17:31 test
```

NFS no_root_squash

实践

/bin/bash

ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)

startup files are read.

Bash attempts to determine when it is being run with its standard input connected to a network connection, as when executed by the remote shell daemon, usually rshd, or the secure shell daemon sshd. If bash determines it is being run in this fashion, it reads and executes commands from ~/.bashrc and ~/.bashrc, if these files exist and are readable. It will not do this if invoked as sh. The `--norc` option may be used to inhibit this behavior, and the `--rcfile` option may be used to force another file to be read, but neither rshd nor sshd generally invoke the shell with those options or allow them to be specified.

If the shell is started with the effective user (group) id not equal to the real user (group) id, and the `-p` option is not supplied, no startup files are read, shell functions are not inherited from the environment, the `SHELLOPTS`, `BASHOPTS`, `CDPATH`, and `GLOBIGNORE` variables, if they appear in the environment, are ignored, and the effective user id is set to the real user id. If the `-p` option is supplied at invocation, the startup behavior is the same, but the effective user id is not reset.

DEFINITIONS

The following definitions are used throughout the rest of this document. Manual page bash(1) line 193 (press h for help or q to quit)

NFS no_root_squash

- ・NFSの共有には『no_root_squash』を設定しない

オプションを指定しない場合はデフォルトで『root_squash』が設定される

- ・書き込み権限が必要無い場合は『rw(read, write)』ではなく『ro(read only)』を指定する

Wild card injection

(勝手に呼んでるだけ)

Wild card injection

- ・ ワイルドカード

→ bashなどのシェルでコマンドを実行する際に『*』を記述する事で
シェル がコマンドを展開してくれる機能

例:ファイル名末尾が『.txt』のファイルを一括で削除したい場合

```
parrot@parrot-virtualbox:~/D/test
> ls
合計 0
-rw-r--r-- 1 parrot parrot 0 11月  1 13:35 1.txt
-rw-r--r-- 1 parrot parrot 0 11月  1 13:35 2.txt
-rw-r--r-- 1 parrot parrot 0 11月  1 13:35 3.csv

parrot@parrot-virtualbox:~/D/test
> rm *.txt

parrot@parrot-virtualbox:~/D/test
> ls
合計 0
-rw-r--r-- 1 parrot parrot 0 11月  1 13:35 3.csv
```


Wild card injection

実際にはどんなコマンドが発行されているか

bashでは『set -x』コマンドを事前に実行しておくと実際に実行されるコマンドを確認できる

```
[x]-[parrot@parrot-virtualbox]-[~/Desktop/test]  
$rm *.txt  
+ rm 1.txt 2.txt  
++ [[ 0 != 0 ]]
```

実際に実行されるのはワイルドカードを展開したコマンド

Wild card injection

シェルスクリプトなどで『*』が使用されている場合、コマンドに任意のオプションを付与することが可能な場合がある。

そのコマンドがroot権限で実行されていると権限昇格に繋がる

Wild card injection

今回は例としてtarコマンドを使用したシナリオ

- ・下記のようなシェルスクリプトがroot権限で実行できる

```
PrivEsc@ubuntu-VirtualBox:~/pe/wildcard$  
sudo -l  
既定値のエントリと照合中 (ユーザー名 PrivEsc) (ホスト名  
ubuntu-VirtualBox):  
env_reset, mail_badpass,  
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin  
n\:/snap/bin,  
use_pty  
  
ユーザー PrivEsc は ubuntu-VirtualBox 上で  
コマンドを実行できます  
(ALL : ALL) NOPASSWD: /usr/bin/nmap  
(ALL : ALL) NOPASSWD: /usr/bin/tar cvf /tmp/backup.tgz *
```

Wild card injection

『--checkpoint=n』 → nファイル毎に進捗を表示(デフォルトは10)

『--checkpoint-action=ACTION』 → チェックポイント毎にACTIONを実行

Informative output

--checkpoint[=N]

Display progress messages every Nth record (default 10).

--checkpoint-action=ACTION

Run ACTION on each checkpoint.

(tarのLinuxマニュアルから抜粋)

ワイルドカードを展開させたときにファイル名をコマンドライン引数として処理させるため
『--checkpoint=1』と『--checkpoint-action=exec=/bin/bash』というファイルを作成する

Wild card injection

TIPS

『touch --checkpoint=1』とすると『--checkpoint=1』がオプションと認識され、そんなオプション無いよといわれるので『touch -- --checkpoint=1』とすると一つの『--』でオプションの終了を知らせることができる

Wild card injection

実践

tar

Wild card injection

・できるだけワイルドカードを使用せず、使用する場合は『*』でファイルを指定するのではなく、『/var/www/html/*』のようにディレクトリを指定するとオプションとして認識されなくなる

他にもあるよ

コマンド	オプション	内容
chown	--reference=FILE	所有者の変更
chmod	--reference=FILE	パーミッションの変更
rsync	--rsh=COMMAND --rsync-path=PROGRAM	任意コマンド実行

有効なツール群

有効なツール群

- ・LinPEAS → 存在するユーザーや権限設定の不備を見つけてくれる

<https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS>

- ・LinEnum → LinPEASとほぼ一緒 個人的にはLinPEASのほうが出力が見易い

<https://github.com/rebootuser/LinEnum>

- ・PSPY → 定期的に生成されるプロセスをroot権限無しで確認できる

<https://github.com/DominicBreuker/pspy>

- ・Nmap → おなじみ。今回の例だとNFSの列挙に使える

<https://github.com/nmap/nmap>

- ・GTFobins → 各コマンドに与えられたSUIDやSudo等の条件から権限昇格パスがあるか検索できる

<https://gtfobins.github.io/>

終わり

- Sudoers → 最小権限の原則
- SUID, SGID → 最小権限の原則
- Capability → 最小権限の原則
- Cronjob → 実行するスクリプトファイルの書き込み権限はCronjobの実行ユーザー以外に付与しない
- Systemd Timer Unit → 『/etc/systemd/system/』以下に書き込み権限を与えない。
『systemctl daemon-reload』をrootで実行させない。
- PATH ENV → スクリプトに記載するパスは絶対パスにする。
- NFS no_root_squash → no_root_squashは使用しない。
- Wild card injection → パスの指定に『*』は使わず、『/var/www/html/*』のように指定する。