



DEVELOPING MALICIOUS KERNEL DRIVERS

Friday, April 21, 2023

Parallels: go.tij.me/virtual-machine-parallels.pvmp

OVF: go.tij.me/virtual-machine-ovf.zip



ABOUT

Tijme Gommers

- Product Lead / Red Teamer
- Works at Northwave Cyber Security
- Forensics at Huntred (NPO3)
- Lives in the Netherlands
- Author of open-source software
[Kernel Mii](#), [Raivo OTP](#), [WikiRaider](#)
- Socials username is @tijme
[Twitter](#), [GitHub](#), [LinkedIn](#)





ABOUT

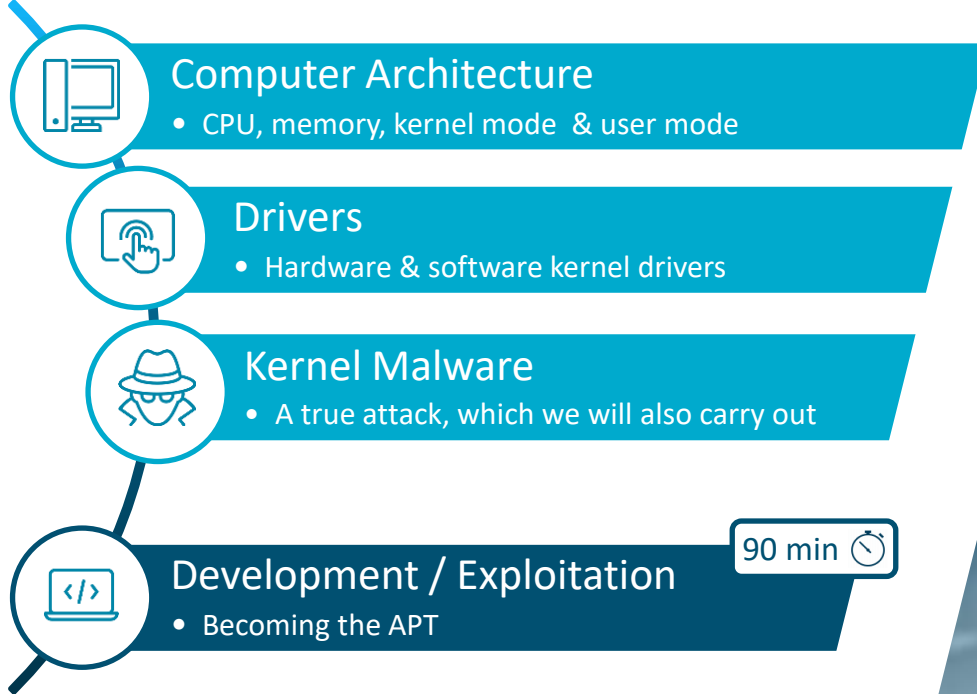
Jan-Jaap Korpershoek

- Red Teamer / Reverse Engineer
- Works at Northwave Cyber Security
- Background in technical computer science
- Lives in the Netherlands
- Social media
GitHub: [@JJK96](#)





INDEX





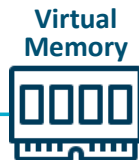
COMPUTER ARCHITECTURE 101

Virtual memory

- lsass.exe

mov rax, [rax]

rax contains address of some LSASS credential



int

4 bytes

Offset	Value
0x82AD1000	0x70
0x82AD1001	0x61
0x82AD1002	0x73
0x82AD1003	0x73
0x82AD1004	0x00
0x82AD1005	0x00
0x82AD1006	0x00
0x82AD1007	0x00
0x82AD1008	0x00
0x82AD1009	0x00
0x82AD100A	0x00
0x82AD100B	0x00
0x82AD100C	0x00
0x82AD100D	0x00



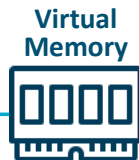
COMPUTER ARCHITECTURE 101

Virtual memory

- my_malware.exe

mov rax, [rax]

rax contains address of some LSASS credential



int

4 bytes

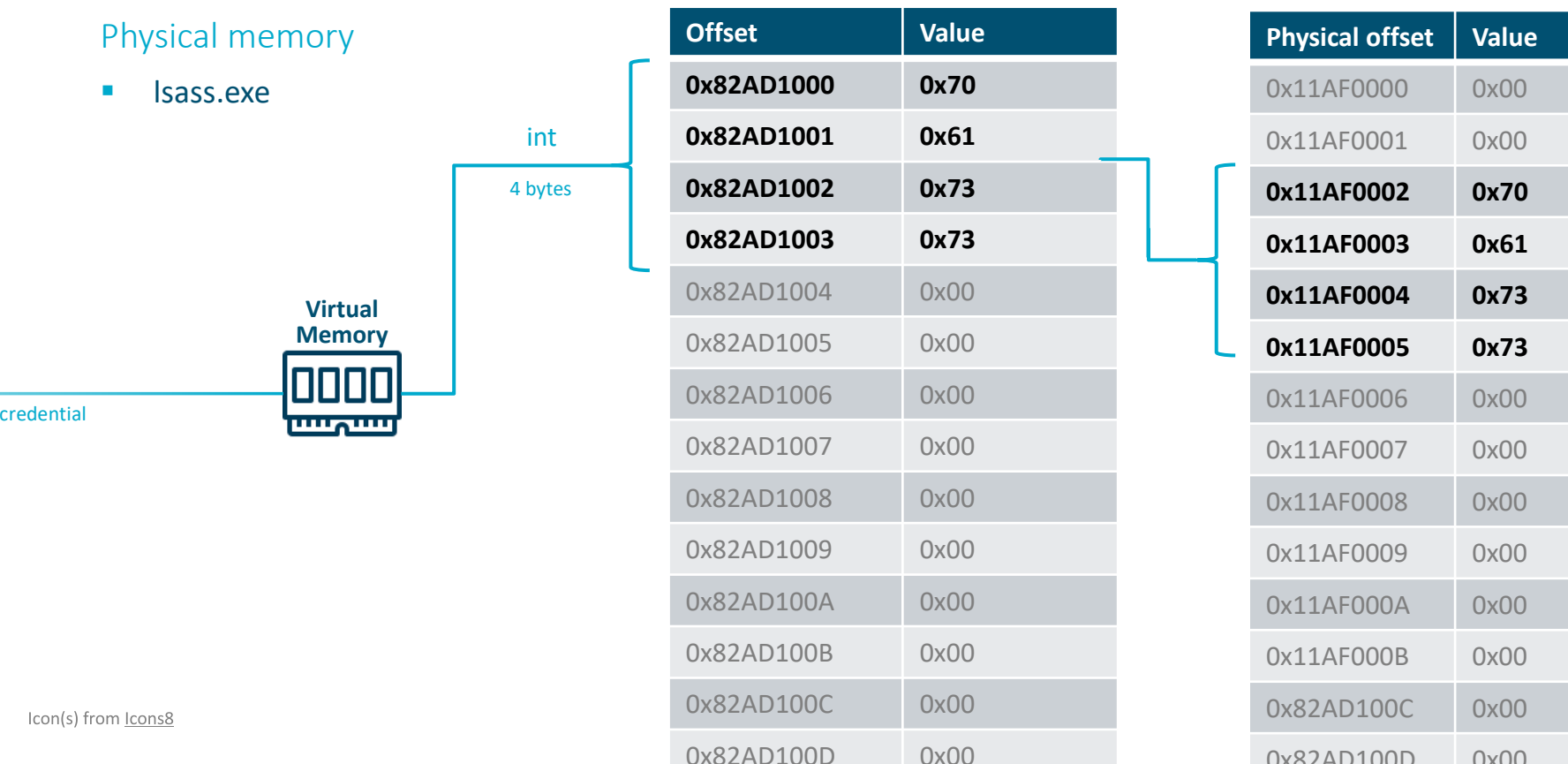
Offset	Value
0x82AD1000	0x00
0x82AD1001	0x00
0x82AD1002	0x00
0x82AD1003	0x00
0x82AD1004	0x00
0x82AD1005	0x00
0x82AD1006	0x00
0x82AD1007	0x00
0x82AD1008	0x00
0x82AD1009	0x00
0x82AD100A	0x00
0x82AD100B	0x00
0x82AD100C	0x00
0x82AD100D	0x00



COMPUTER ARCHITECTURE 101

Physical memory

- Isass.exe

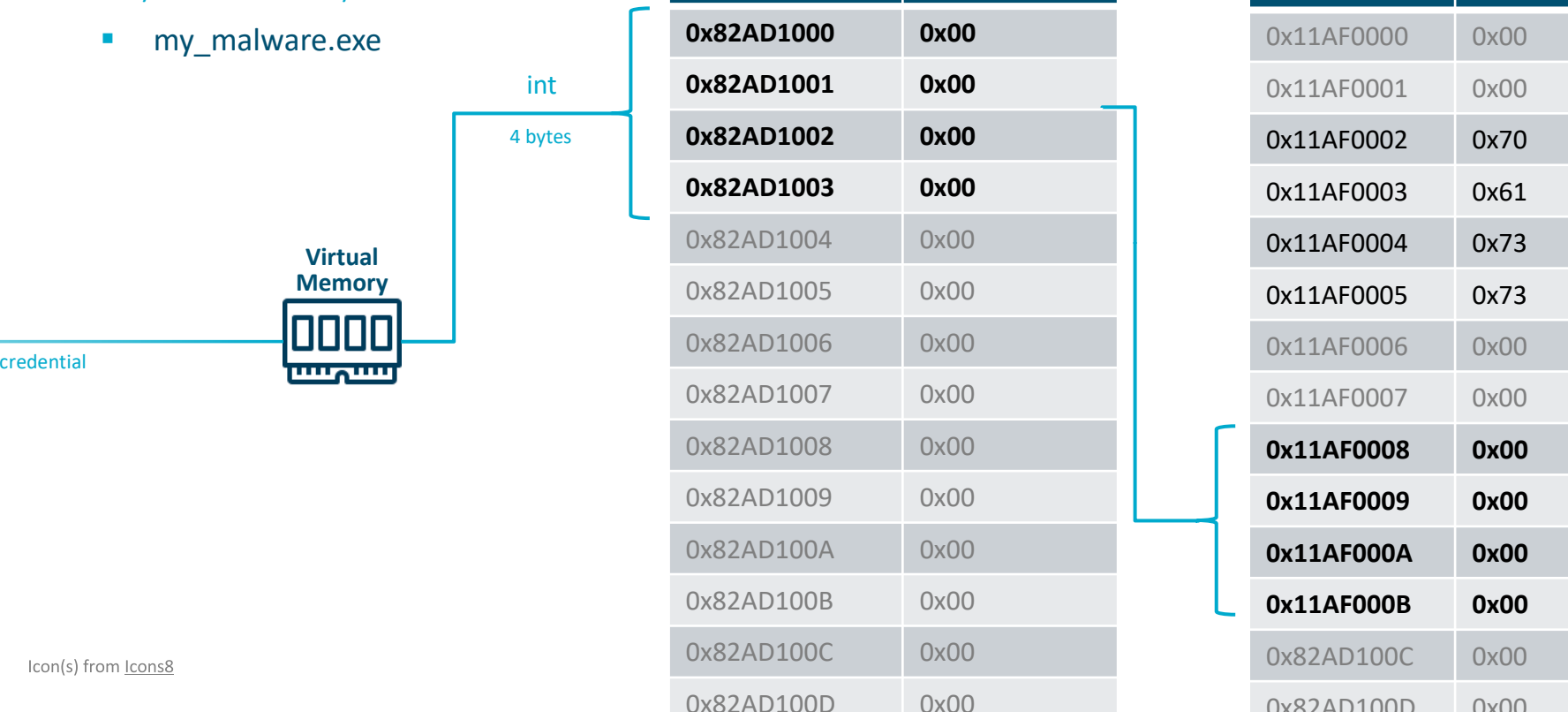




COMPUTER ARCHITECTURE 101

Physical memory

- my_malware.exe



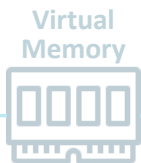


COMPUTER ARCHITECTURE 1

Physical memory

- lsass.exe
- my_malware.exe

credential



Windows Memory Mapping

Runs in kernel mode

Offset	Value
0x82AD1000	0x00
0x82AD1001	0x00
0x82AD1002	0x00
0x82AD1003	0x00
0x82AD1004	0x00
0x82AD1005	0x00
0x82AD1006	0x00
0x82AD1007	0x00
0x82AD1008	0x00
0x82AD1009	0x00
0x82AD100A	0x00
0x82AD100B	0x00
0x82AD100C	0x00
0x82AD100D	0x00

Physical offset	Value
0x11AF0000	0x00
0x11AF0001	0x00
0x11AF0002	0x70
0x11AF0003	0x61
0x11AF0004	0x73
0x11AF0005	0x73
0x11AF0006	0x00
0x11AF0007	0x00
0x11AF0008	0x00
0x11AF0009	0x00
0x11AF000A	0x00
0x11AF000B	0x00
0x82AD100C	0x00
0x82AD100D	0x00

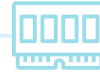


COMPUTER ARCHITECTURE 101

User mode to kernel mode



mimikatz

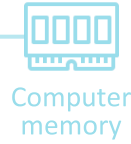


Computer
memory



COMPUTER ARCHITECTURE 101

User mode to kernel mode



Computer
memory



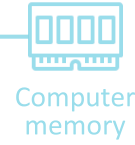
mimikatz



KERNEL32.dll
OpenProcess



COMPUTER ARCHITECTURE 101



User mode to kernel mode



mimikatz



KERNEL32.dll
OpenProcess



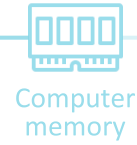
NTDLL.dll
NtOpenProcess





COMPUTER ARCHITECTURE 101

User mode to kernel mode



Computer memory



mimikatz



KERNEL32.dll
OpenProcess

NTDLL.dll
NtOpenProcess

Syscall table

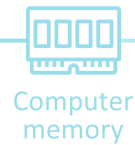
USER MODE

KERNEL MODE



COMPUTER ARCHITECTURE 101

User mode to kernel mode



Computer memory



mimikatz



KERNEL32.dll
OpenProcess

NTDLL.dll
NtOpenProcess

Syscall table

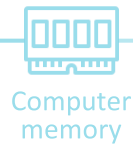
NtOsKrnI.exe
NtOpenProcess

USER MODE
KERNEL MODE



COMPUTER ARCHITECTURE 101

User mode to kernel mode



Computer memory



mimikatz



KERNEL32.dll
OpenProcess

NTDLL.dll
NtOpenProcess

KERNEL32.dll
ReadProcessMemory

...

Syscall table

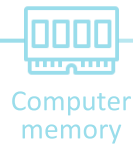
NtOsKrnI.exe
NtOpenProcess

USER MODE
KERNEL MODE



COMPUTER ARCHITECTURE 101

User mode to kernel mode



Computer memory



mimikatz



KERNEL32.dll
OpenProcess

NTDLL.dll
NtOpenProcess

KERNEL32.dll
ReadProcessMemory

...

Syscall table

Administrative rights?
Process is not protected?



NtOsKrnI.exe
NtOpenProcess

USER MODE
KERNEL MODE



MmMapIoSpace function (wdm) x

+

← ↻ 🔒 https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-mmapiospace

Windows Hardware Developer Explore ▾ Downloads ▾ Windows Driver Kit samples Resources ▾

Dashboard

Filter by title

Kernel

> Aux_klib.h

> Ioaccess.h

> Ioindex.h

> Miniport.h

> Ntddk.h

> Ntddsfio.h

> Ntddsysenv.h

> Ntifs.h

> Ntintsafe.h

> Ntpoapi.h

> Ntstrsafe.h

> Pcvirt.h

> Pep_x.h

> Pepfx.h

Download PDF

⋮ / Windows / Windows Drivers / API / Kernel / Wdm.h /

⊕ ✎ ⋮

MmMapIoSpace function (wdm.h)

Article • 01/13/2023 • 2 minutes to read

Feedback

The MmMapIoSpace routine maps the given physical address range to nonpaged system space.

Syntax

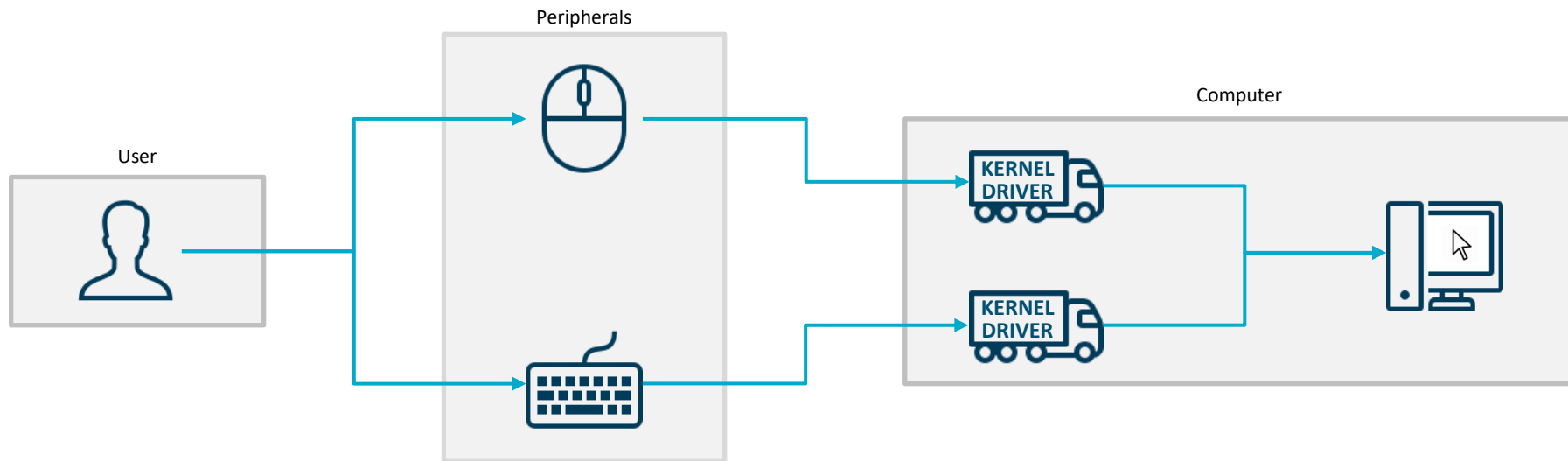
C++ Copy

```
PVOID MmMapIoSpace(  
    [in] PHYSICAL_ADDRESS    PhysicalAddress,  
    [in] SIZE_T               NumberOfBytes,  
    [in] MEMORY_CACHING_TYPE CacheType  
);
```



DRIVERS 101

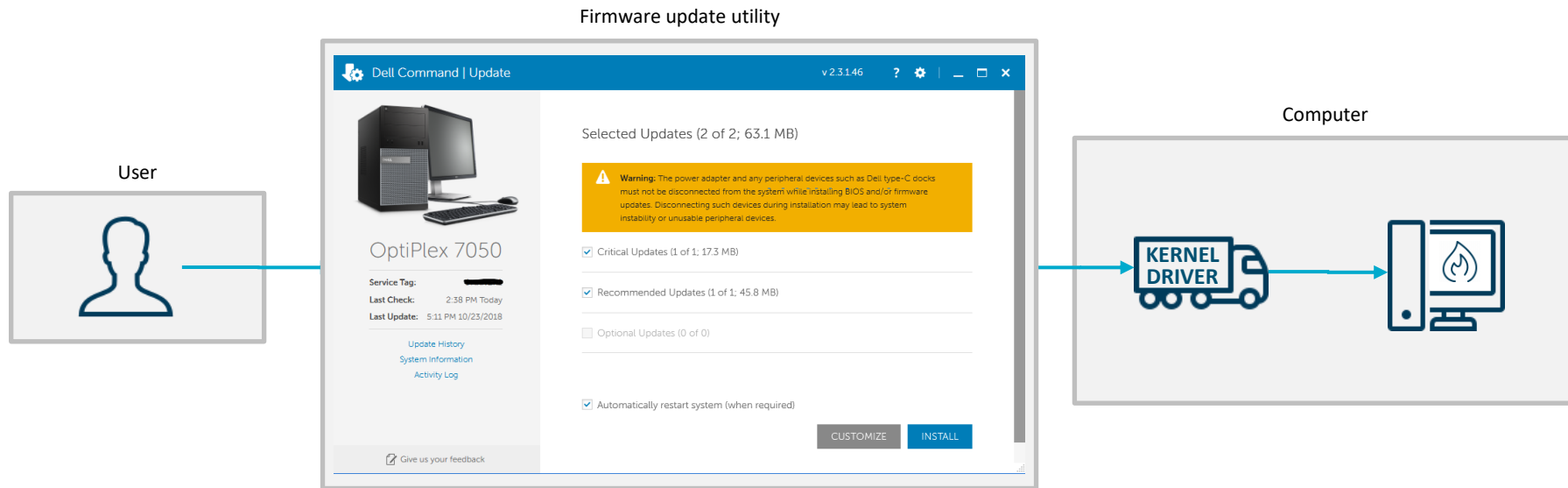
Hardware drivers





DRIVERS 101

Software drivers

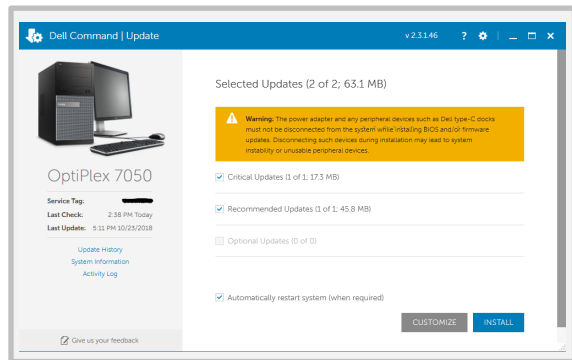




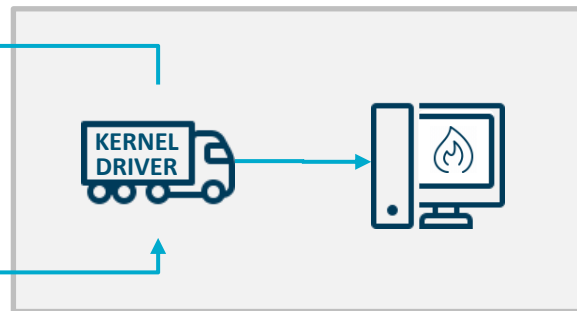
DRIVERS 101

Input/output control codes (IOCTL's)

Firmware update utility



Computer



Input/output control code (IOCTL)
0x8000400C: Read firmware status

Input/output control code (IOCTL)
0x8000500D: Update firmware to given binary

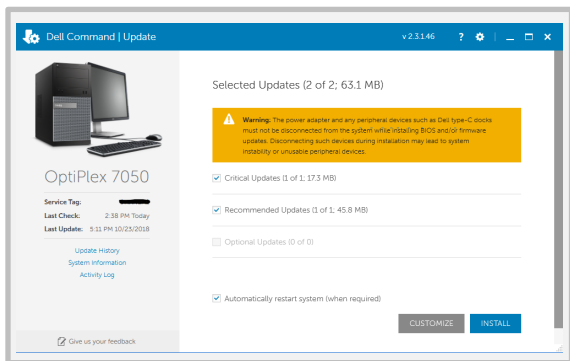
Note: These IOCTLs are fictional



DRIVERS 101

Input/output control codes (IOCTL's)

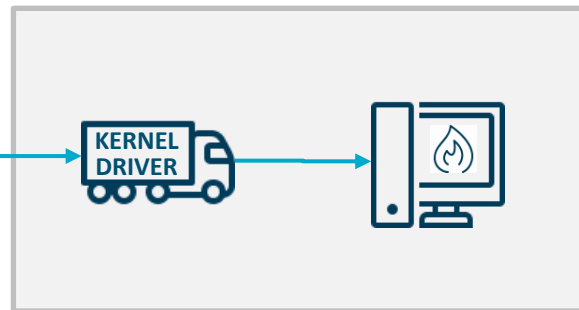
Firmware update utility



Input/output control code (IOCTL)
0x8000500D: Update firmware to given binary

```
char * data = ...;  
size_t size = 0x1000;  
  
HANDLE handle = CreateFile("\\\\.\\dbutil_2_3", ...)  
  
DeviceIoControl(handle, 0x8000500D, data, size, ...)
```

Computer



Note: This example is fictional



DRIVERS 101

Switch statement of supported IOCTL's in driver source code

When someone calls DeviceIoControl(...)



```
void IRP_MJ_DEVICE_CONTROL (DEVICE_OBJECT* device, IRP* irp) {
    DeviceIoControl* IoControl = irp->Tail->CurrentStackLocation->DeviceIoControl;
    VOID* UserBuffer = irp->Associated Irp->SystemBuffer;

    switch (IoControl->IoControlCode) {
        case 0x8000400C:
            UserBuffer readFirmwareStatus();
            break;
        case 0x8000500D:
            writeFirmware(UserBuffer);
            break;
        case 0x8000600E:
            break;
        default:
            puts("Unknown IOCTL");
    }
}
```

Note: These IOCTLs are fictional



Updating your system's firmware. Do not power down your system.

Waiting for Intel(R) ME FW Update to complete





DRIVERS 101

Installation

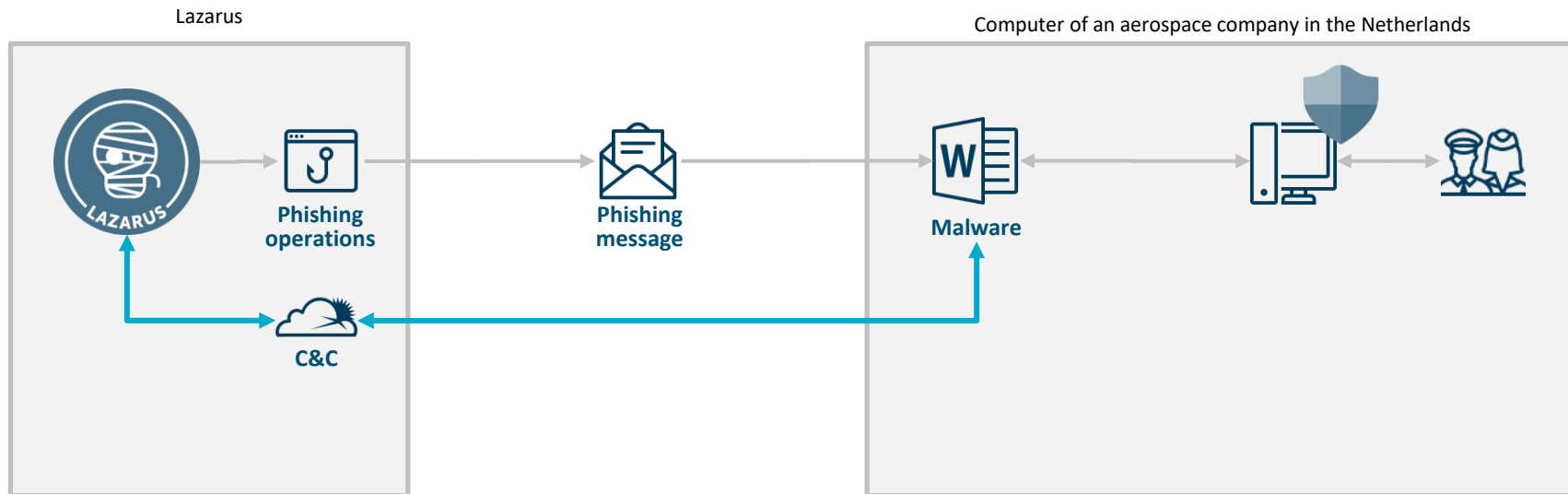
- Driver signing useful for threat actors, but probably not for red teams.
- In this lab we'll exploit an existing driver instead!
`dbutil_2_3.sys`





A TRUE ATTACK (2021)

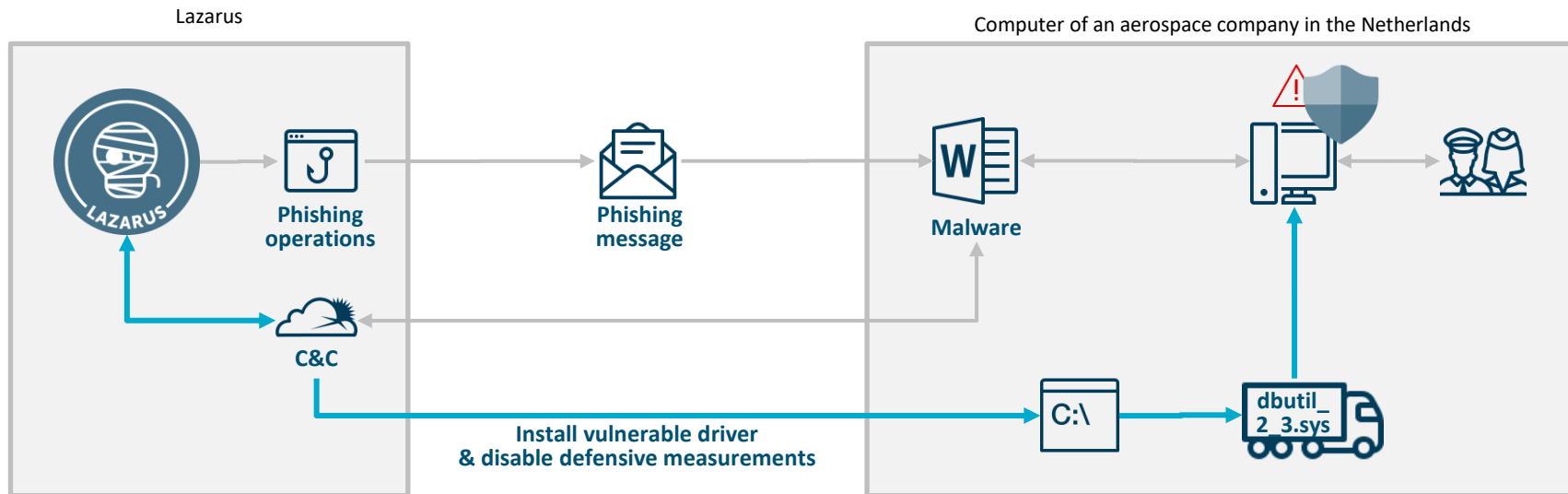
Research by 





A TRUE ATTACK (2021)

Research by 





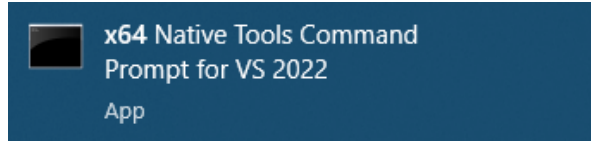
EXERCISES



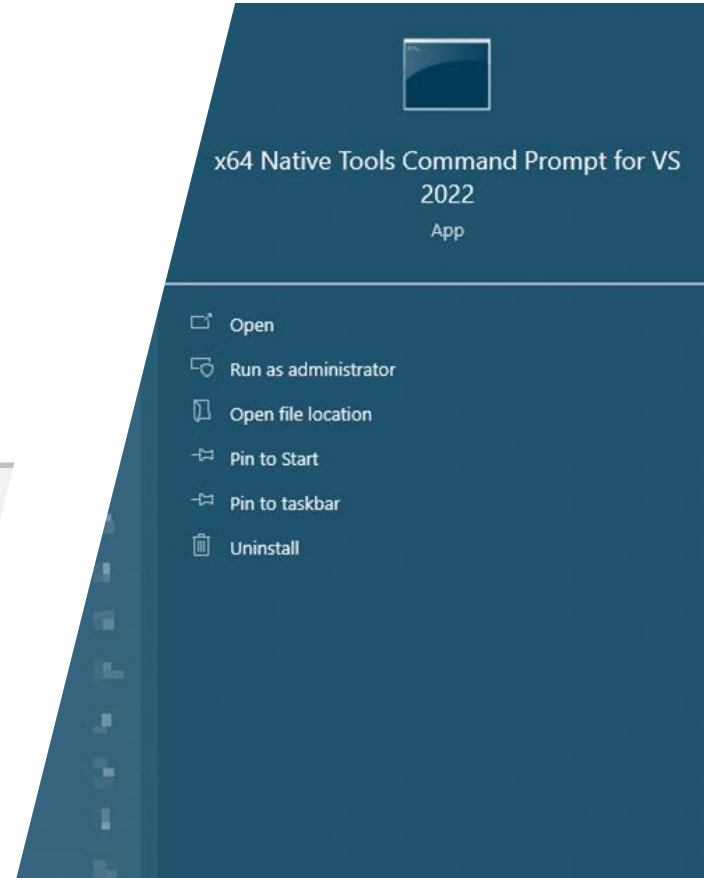
PROGRAMMING

Compiling C files

1. Start “x64 Native Tools Command Prompt For VS”



2. Compile your code





BECOMING THE APT

Exercise 1

25 min 

1. Boot your virtual machine (64-bit Windows 10).
2. Download the exercise files at go.tij.me/exercise1-good-luck.zip.
3. Enable LSASS PPL by running enable-lsass-ppl.reg.
4. Install the vulnerable driver.

```
sc.exe create dbutil_2_3 binPath= "c:\dbutil_2_3.sys" type= kernel start= auto  
sc.exe start dbutil_2_3
```

5. Reboot your virtual machine.
#protip: create a snapshot you can revert to.
6. Find the driver device name using WinObj (sysinternals).
7. Compile exercise1.c and use it to open a handle to the driver.





DEBUGGING

Debugging the kernel using WinDBG

1. Enable debugging mode.

```
bcdedit.exe -debug on
```



2. Reboot your virtual machine.
#protip: create a snapshot you can revert to.
3. Start WinDBG (sysinternals) as administrator.

Start debugging



Launch executable



Launch executable (advanced)
Supports Time Travel Debugging



Attach to process
Supports Time Travel Debugging



Open dump file



Open trace file



Connect to remote debugger



Connect to process server



Attach to kernel



Launch app package

Net COM Local

No debugger configuration
local kernel debugging



DEBUGGING

Calling convention

1. Suppose we perform a memmove.

```
memmove(void* dst, void* src, size_t len);
```

2. Then:

rcx = <dst>

rdx = <src>

r8 = <length>

call memmove

```
sub_15294 proc near
var_28= qword ptr -28h
var_20= qword ptr -20h
var_18= dword ptr -18h

push    rbx
sub     rsp, 40h
mov     rbx, rcx
mov     ecx, [rcx+8]
cmp     ecx, 18h
jnb     short loc_152AC
```

```
loc_152AC:
mov     r9, [rbx]
lea     r8, [rsp+48h+var_28]
mov     rax, [r9]
mov     [r8], rax      ; var_28 = .ignore
mov     rax, [r9+8]
mov     [r8+8], rax    ; var_20 = .unk1
mov     rax, [r9+10h]
mov     [r8+10h], rax  ; var_18 = .unk2
mov     rax, [rbx+10h]
test    rax, rax
jz      short loc_152E1
```



REVERSING KERNEL DRIVER DEMO



Recycle Bin



Mac Files



Microsoft
Edge



IDA Freeware
8.2



Activate Windows
Go to Settings to activate Windows.



BECOMING THE APT

Exercise 2

45 min 

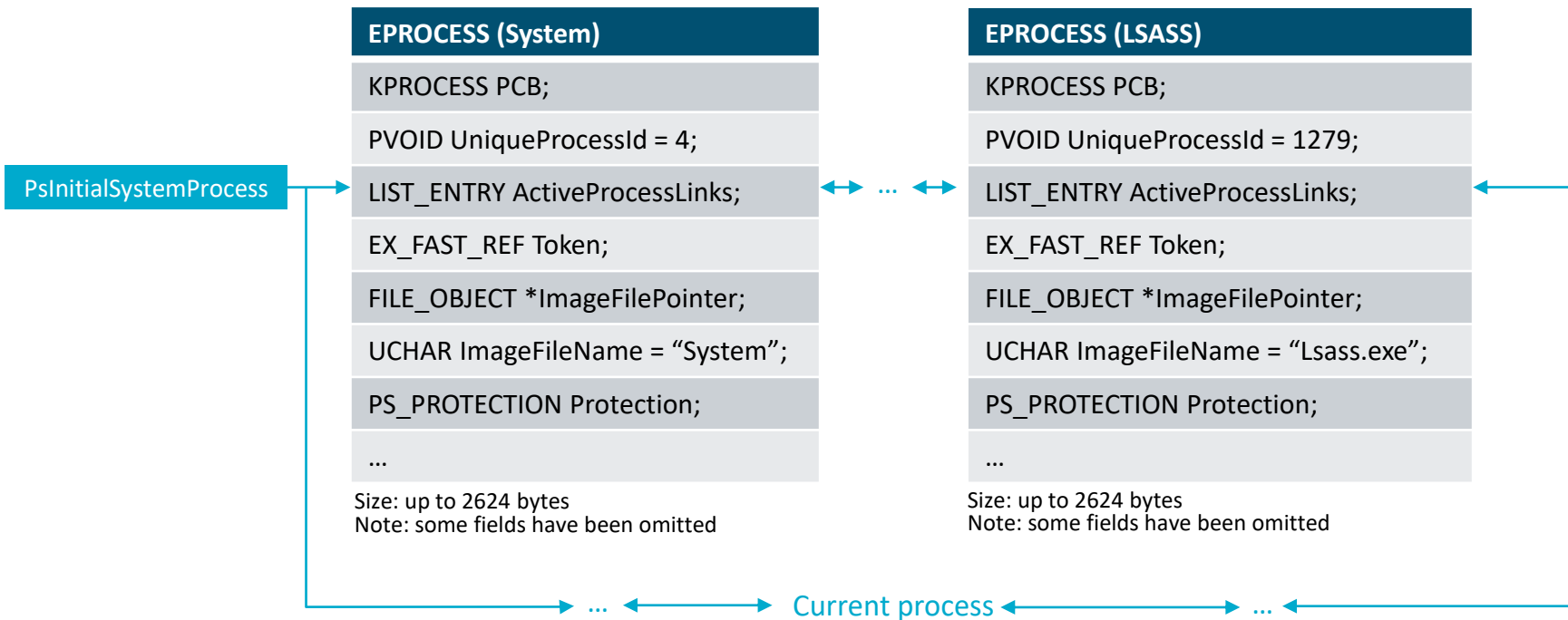
1. Download the exercise file at go.tij.me/exercise2-you-can-do-it
2. Use IDA Free to reverse the vulnerable driver.
 - 2.1. Find the I/O control code (IOCTL) callback function.
 - 2.2. Find the given vulnerable IOCTL numbers.
3. Adjust exercise2.c to interact with the IOCTL (legitimately).





WINDOWS KERNEL 101

You can find most structs on vergiliusproject.com





WINDOWS KERNEL 101

You can find most structs on vergiliusproject.com

```
typedef struct _LIST_ENTRY {  
    struct _LIST_ENTRY *Flink;  
    struct _LIST_ENTRY *Blink;  
} LIST_ENTRY, *PLIST_ENTRY, PRLIST_ENTRY;
```



WINDBG 101

KD ", Local Connection - WinDbg 1.2210.3001.0 (Administrator)

File Home View Breakpoints Time Travel Model Scripting Source Memory Comm

Break Go Step Out Step Into Step Over Step Out Back Step Into Back Step Over Back Go Back Restart Stop Debugging Detach Settings Source Assembly Local Help Feedback Hub

Flow Control Reverse Flow Control End Preferences Help

Disassembly Registers Memory 0 Command X

Microsoft (R) Windows Debugger Version 10.0.25200.1003 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Connected to Windows 10 19041 x64 target at (Sat Apr 15 15:35:51.049 2023 (UTC + 2:00)), ptr64 TRUE
Symbol search path is: srv*
Executable search path is:
Windows 10 Kernel Version 19041 MP (4 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Edition build lab: 19041.1.amd64fre.vb_release.191206-1406
Machine Name:
Kernel base = 0xfffff800`53000000 PsLoadedModuleList = 0xfffff800`53c2a290
Debug session time: Sat Apr 15 15:35:52.628 2023 (UTC + 2:00)
System Uptime: 0 days 0:04:41.610

lkd> dt nt!_eprocess
+0x000 Pcb : _KPROCESS
+0x438 ProcessLock : _EX_PUSH_LOCK
+0x440 UniqueProcessId : Ptr64 Void



BECOMING THE APT

Exercise 3 20 min + your own time

1. Download the exercise file at go.tij.me/exercise3-final-boss.c.
2. Understand the vulnerability.
3. Adjust exploit.c to exploit the vulnerability.
 - 2.1. Find PsInitialSystemProcess.
 - 2.2. Iterate over all other EPROCESS's.
 - 2.3. Identify which EPROCESS is LSASS.
 - 2.4. Disable Protected Process Light on LSASS.
4. Use ProcDump to dump LSASS!

```
procdump -ma lsass.exe lsass.dmp
```





RECOMMENDED LITERATURE

1. **Low-Level Programming**
Igor Zhirkov
2. **Windows Kernel Programming**
Pavel Yosifovich
3. **Practical Reverse Engineering**
Bruce Dang
4. **Windows Internals**
Pavel Yosifovich



