TEXAS CYBER SUMMIT

CYBER SECURITY CONFERENCE

2021

TEXASCYBER.COM

Protected Process Light will be Protected - MemoryRanger Fills the Gap Again

Igor Korkin

Independent Researcher

2021

WHOAMI

- PhD, speaker at the ADFSL, BlackHat, HITB, IEEE SPW
- OS Security Researcher:
  - Rootkits, Anti-rootkits and EDRs
  - Memory Forensics for user- and kernel- modes
  - Bare-Metal Hypervisors against Attacks on Kernel Memory
- Fan of cross-disciplinary research — igorkorkin.blogspot.com
- Love traveling and powerlifting — igor.korkin

AGENDA

- Protected Process Light (PPL) — Algorithm and Attacks

AGENDA

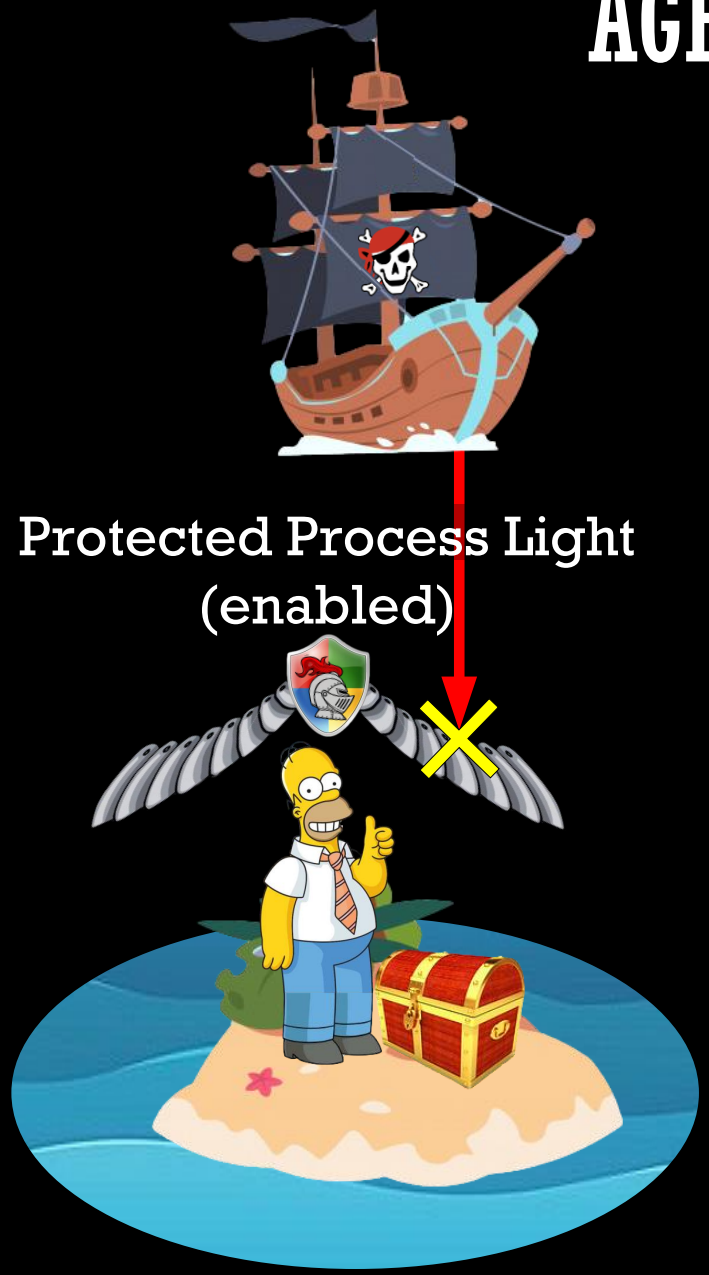
Users secrets are stored in process memory

AGENDA

Protected Process Light
(enabled)

PPL is enabled for the process to protect its memory

AGENDA



Attackers are trying to steal the secrets, but PPL blocks their access

Protected Process Light (enabled)

Thanks to PPL non-protected processes cannot do the following:

- Access to the protected process memory
- Inject code into the protected processes
- Terminate protected processes

Attackers are trying to steal the secrets, but PPL blocks their access

AGENDA

Protected Process Light
(enabled)

Protected Process Light
(disabled)

Attackers can disable PPL

AGENDA

Protected Process Light
(enabled)

Protected Process Light
(disabled)

User's secret
(stolen)

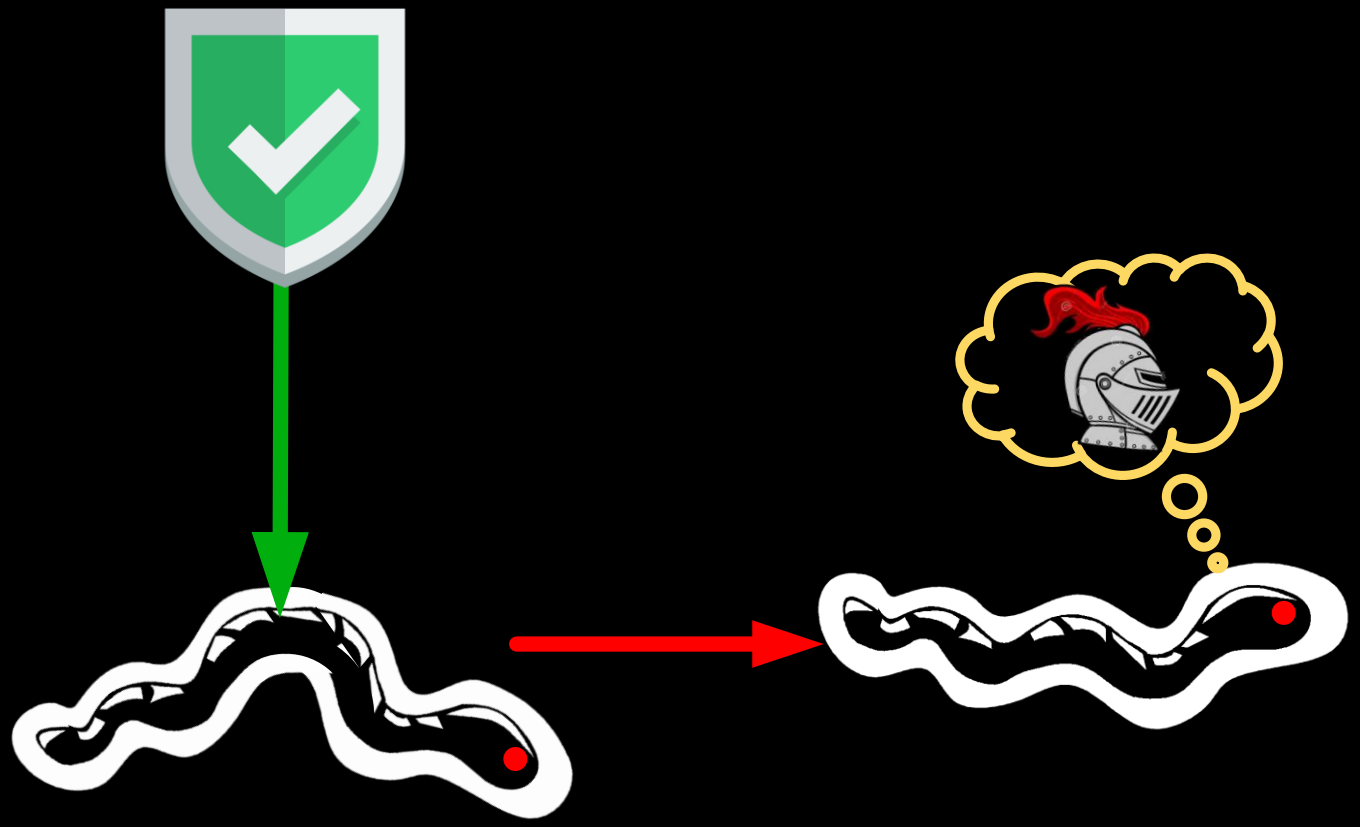
Attackers can steal users data easily

AGENDA

Malware is not protected

Attackers want to protect their malware processes

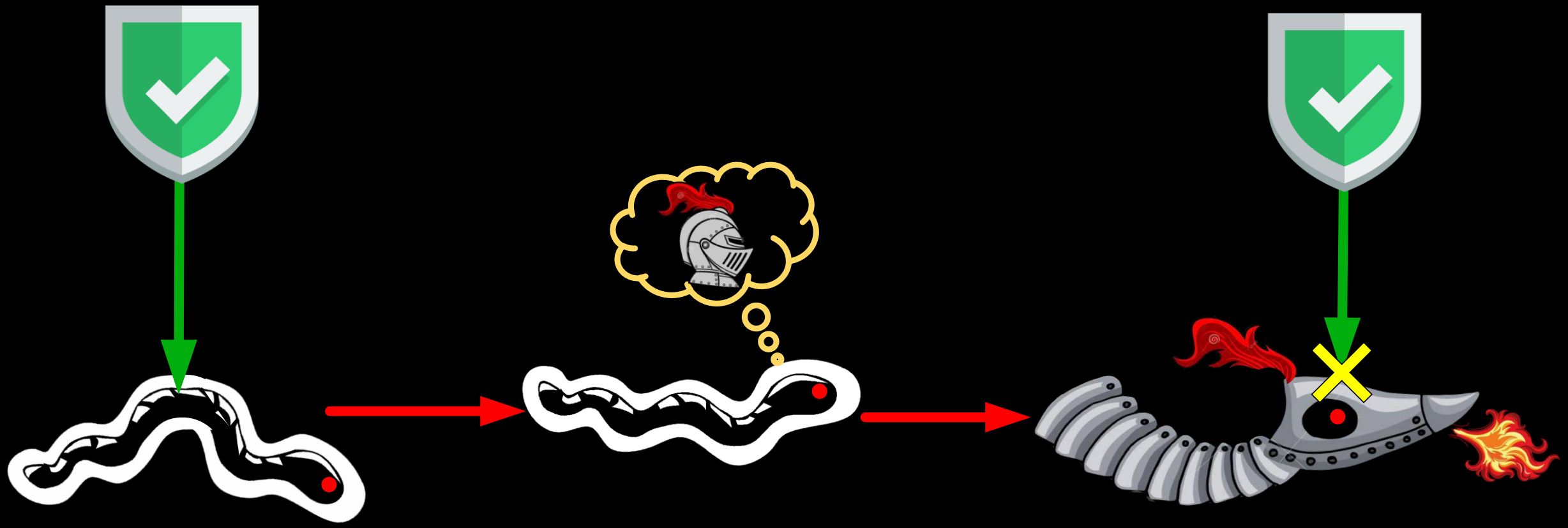
AGENDA



Malware is not protected

But PPL is enabled only for processes with a special signature

AGENDA



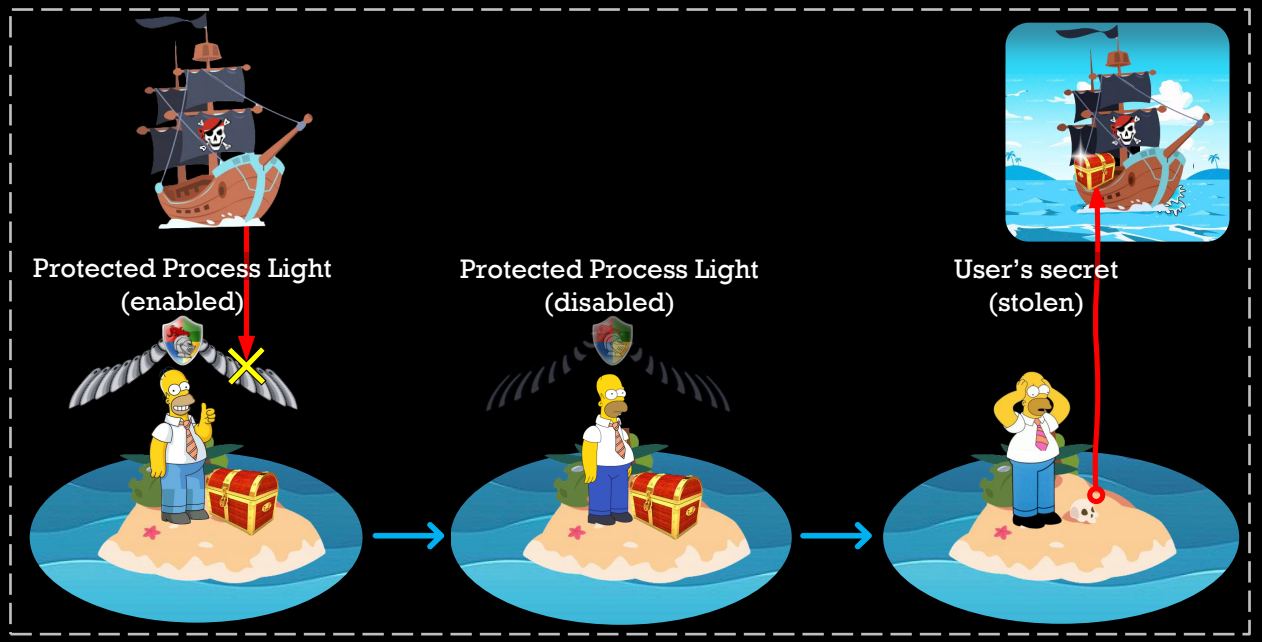
Malware is not protected

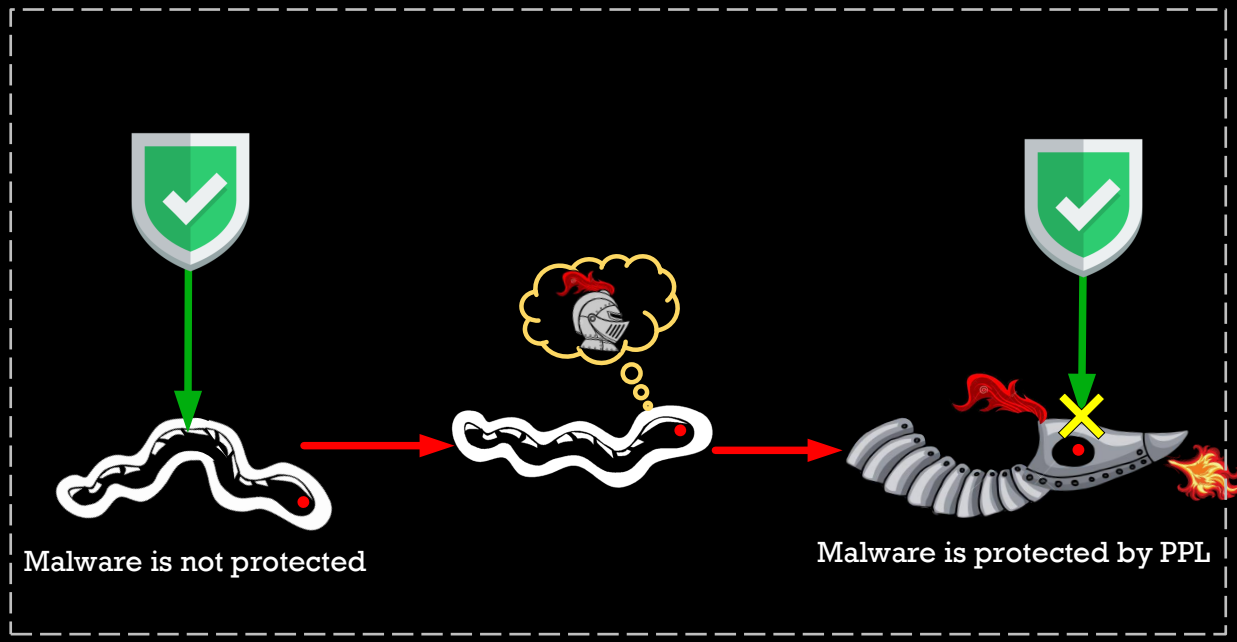
Malware is protected by PPL

Malware apps can illegally enable PPL to protect themselves

AGENDA

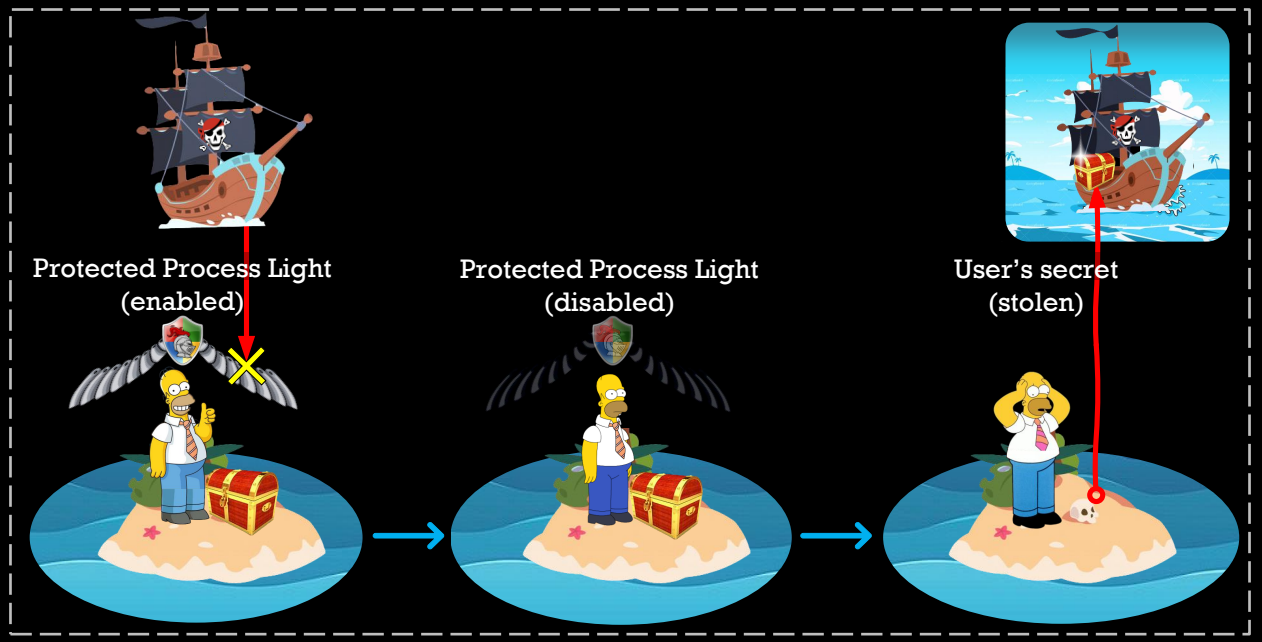
- Protected Process Light (PPL) — Algorithm and Attacks

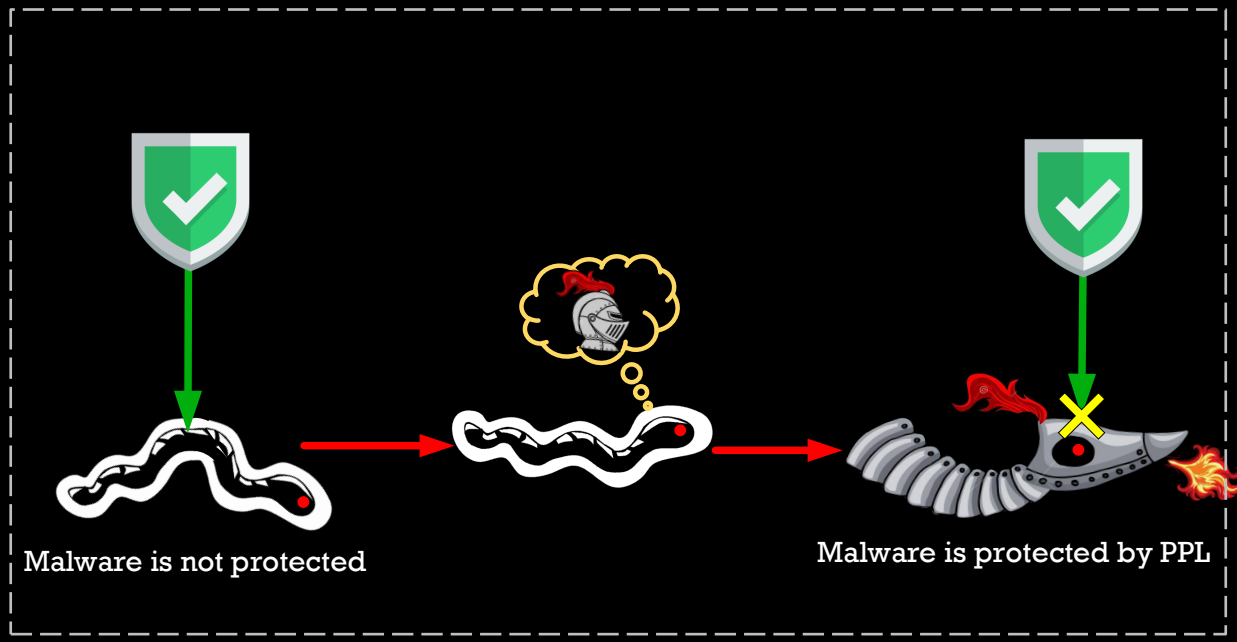




AGENDA

- Protected Process Light (PPL) — Algorithm and Attacks

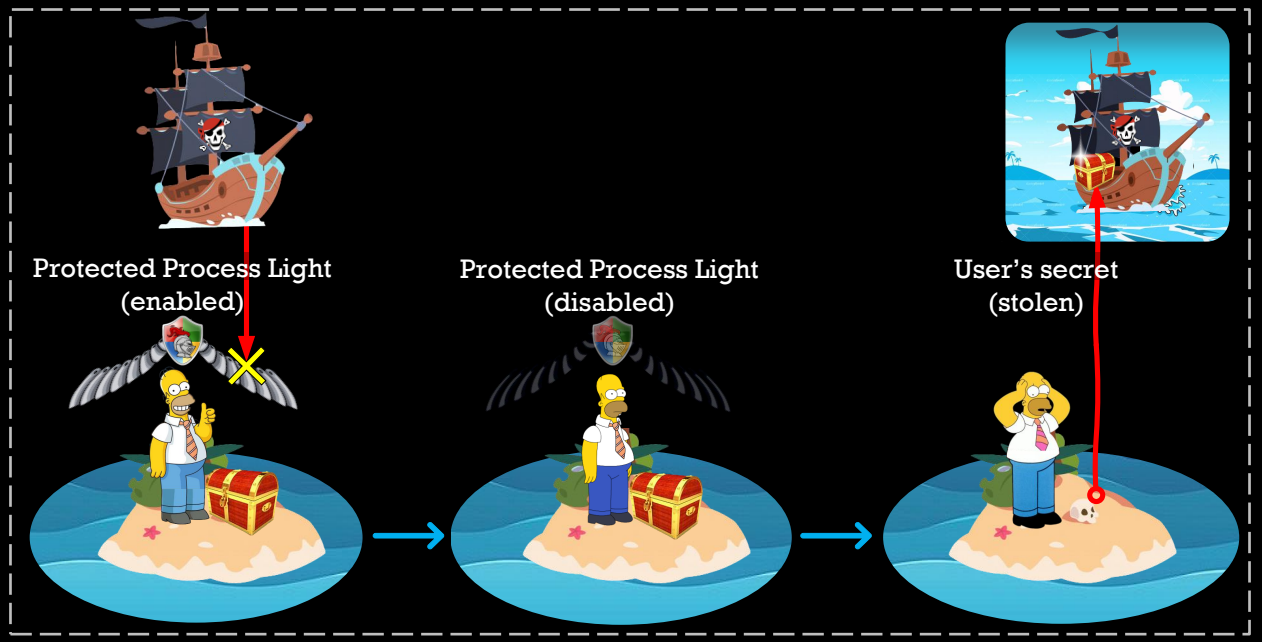


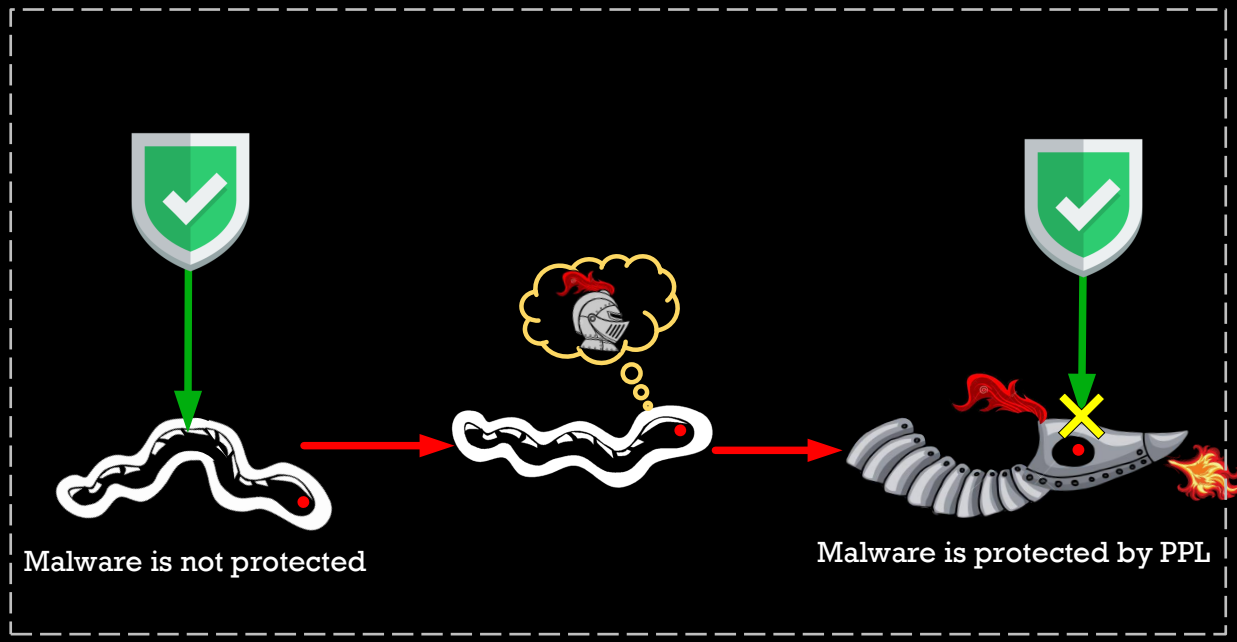


Changing kernel data can abuse PPL

AGENDA

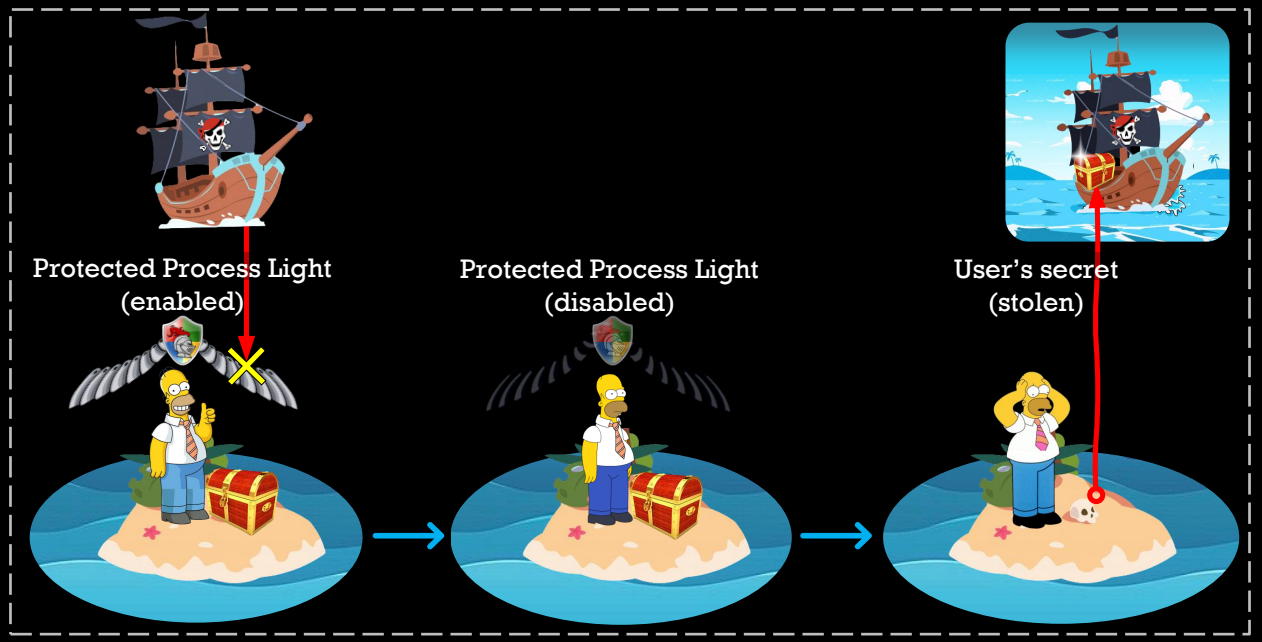
- Protected Process Light (PPL) — Algorithm and Attacks

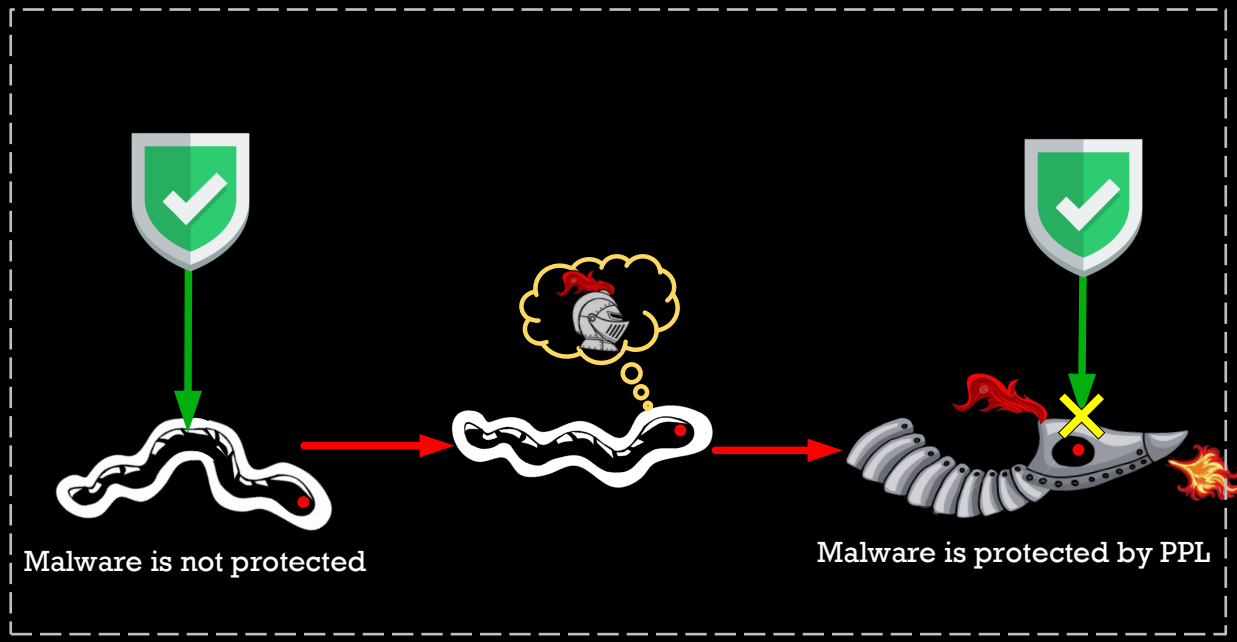




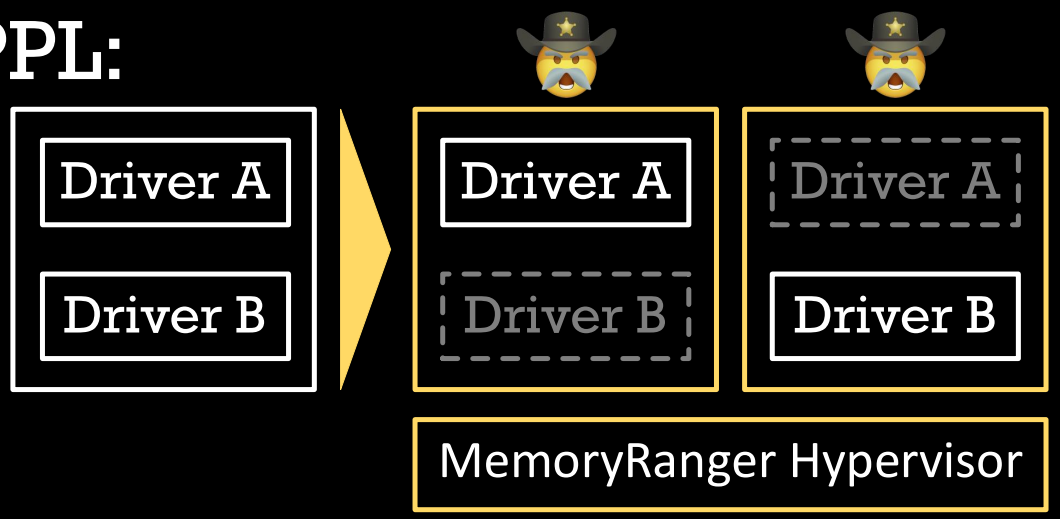
AGENDA

- Protected Process Light (PPL) — Algorithm and Attacks





- MemoryRanger blocks attacks on PPL:



Microsoft Windows OSes

Episode 1

Does Windows provide any feature to Protect Process Memory?

WINDOWS FEATURES TO PROTECT PROCESS MEMORY

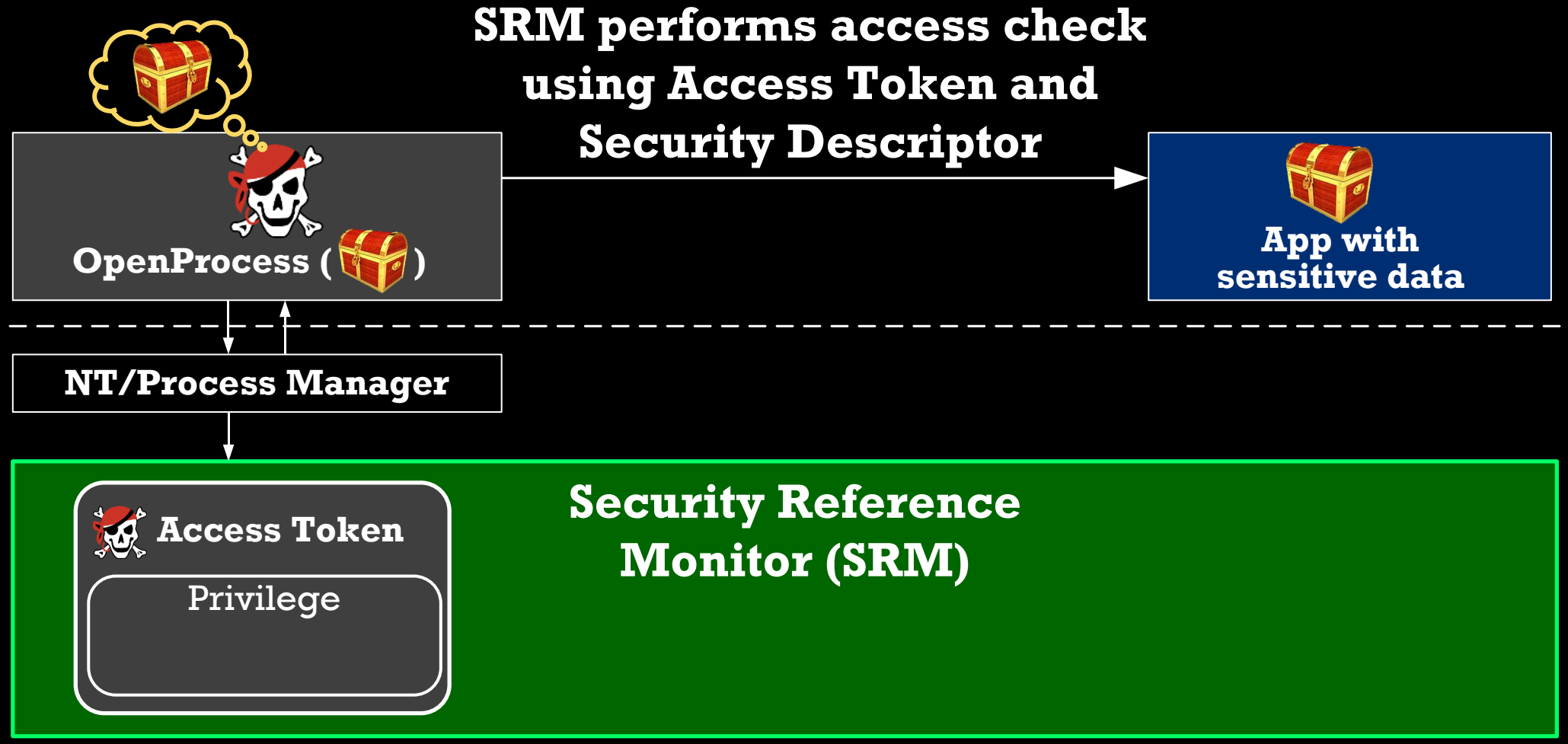
1. Security Reference Monitor (SRM)
2. Protected Process Light (PPL)
3. AppContainer Isolation
4. Windows Resource Protection (WRP, SFC)
5. Session 0 Isolation and Secure Desktop
6. Windows Memory Management (Virtual Memory and Enclave API)
7. Windows Integrity Control (WIC)
8. Mandatory Integrity Control (MIC)
    1. User Interface Privilege Isolation (IUPI)
    2. Enhanced Protected Mode (EPM)
9. Isolated User Mode (IUM) enabled by Hyper-V

WINDOWS FEATURES TO PROTECT PROCESS MEMORY: SRM

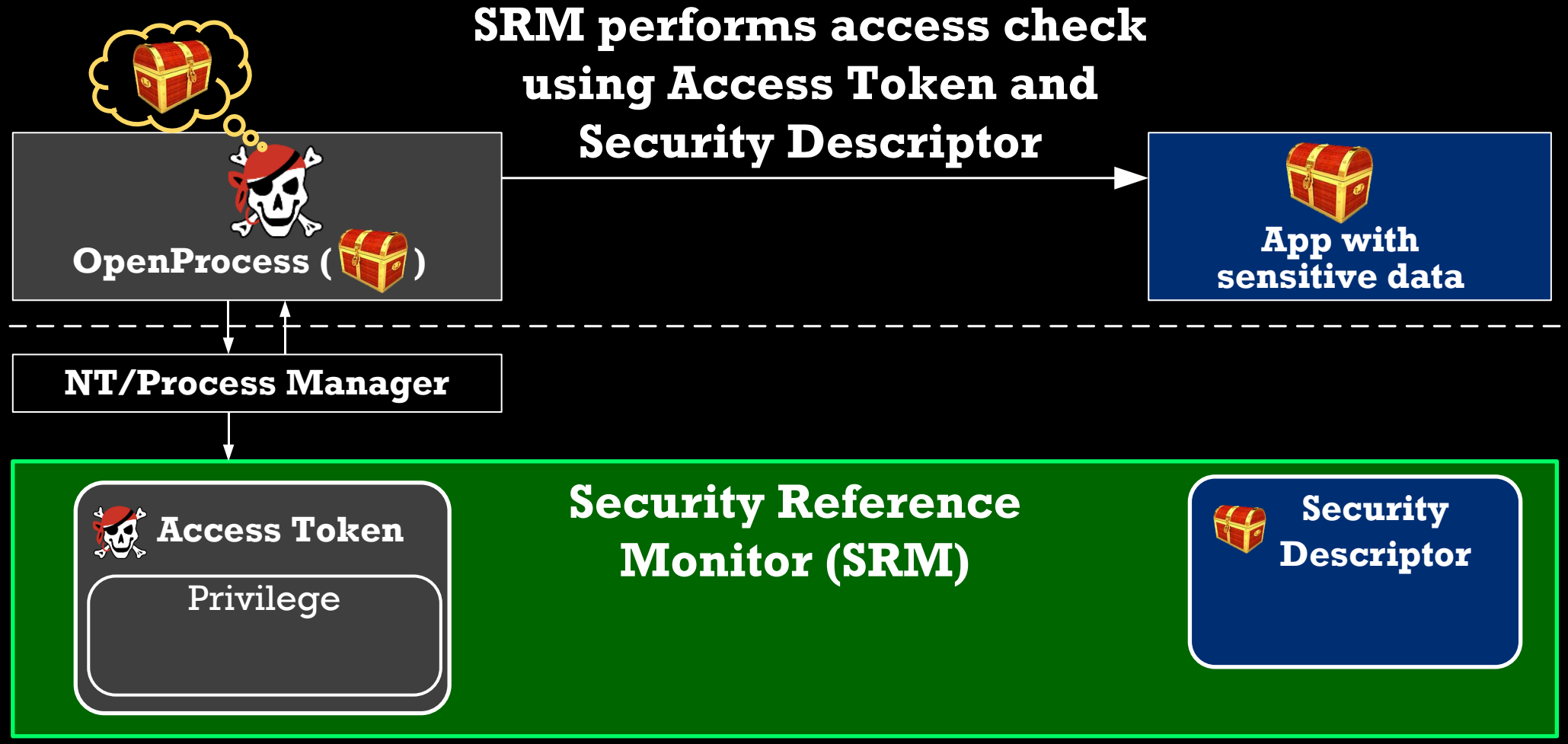
1. Security Reference Monitor (SRM)
2. Protected Process Light (PPL)
3. AppContainer Isolation
4. Windows Resource Protection (WRP, SFC)
5. Session 0 Isolation and Secure Desktop
6. Windows Memory Management (Virtual Memory and Enclave API)
7. Windows Integrity Control (WIC)
8. Mandatory Integrity Control (MIC)
    1. User Interface Privilege Isolation (IUPI)
    2. Enhanced Protected Mode (EPM)
9. Isolated User Mode (IUM) enabled by Hyper-V

A malware App calls OpenProcess() to Access Process Data

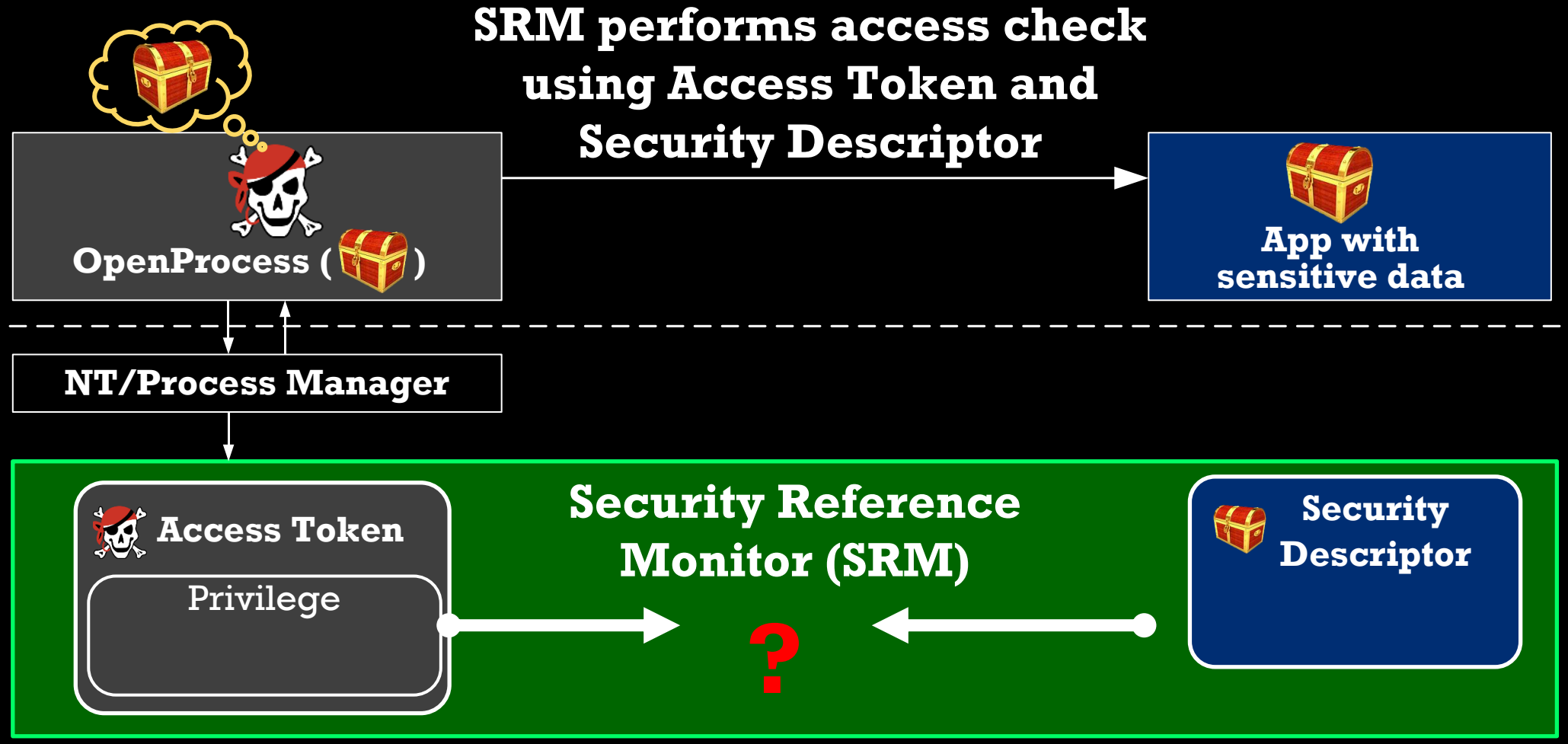
SRM checks access rights using Token and Security Descriptor



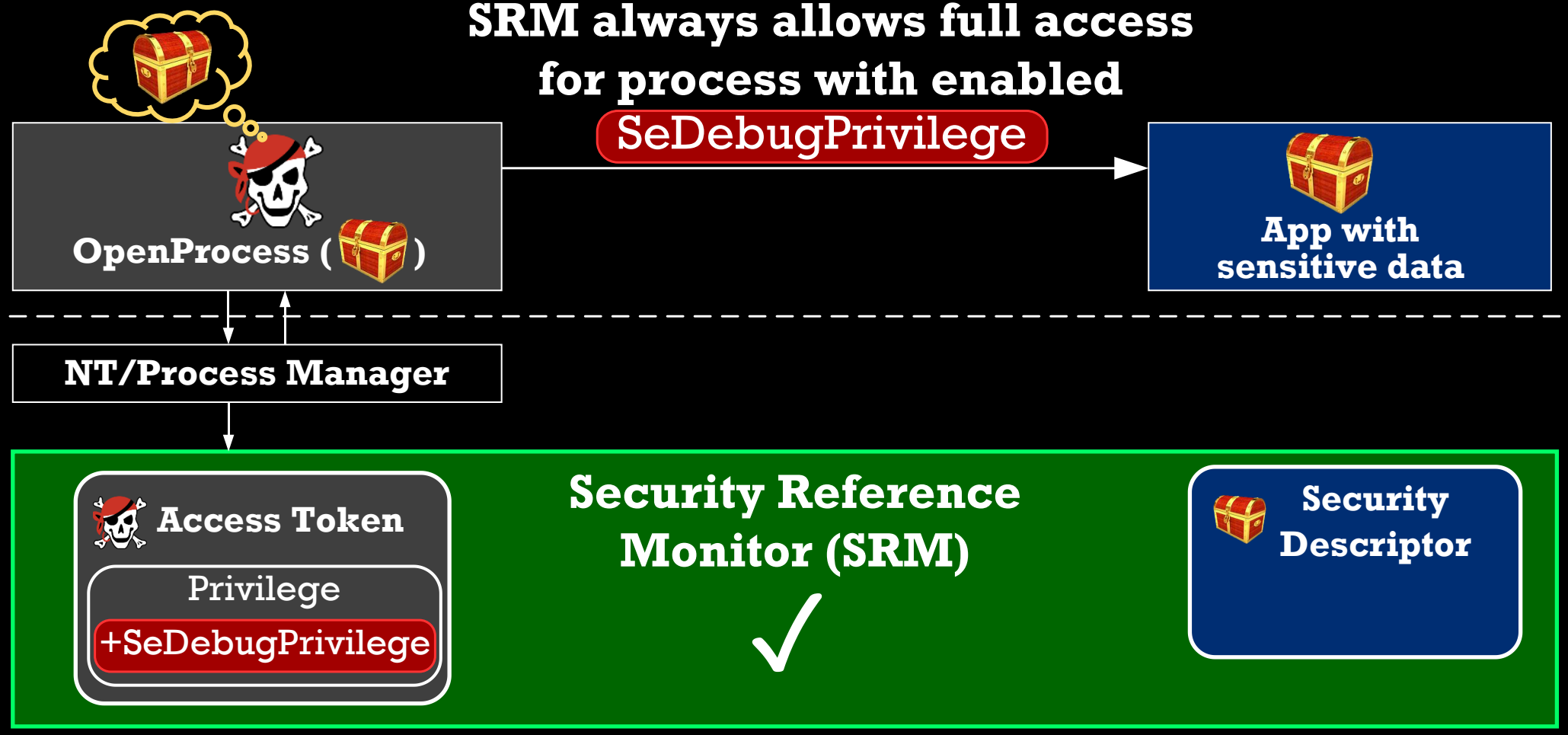
SRM checks access rights using Token and Security Descriptor



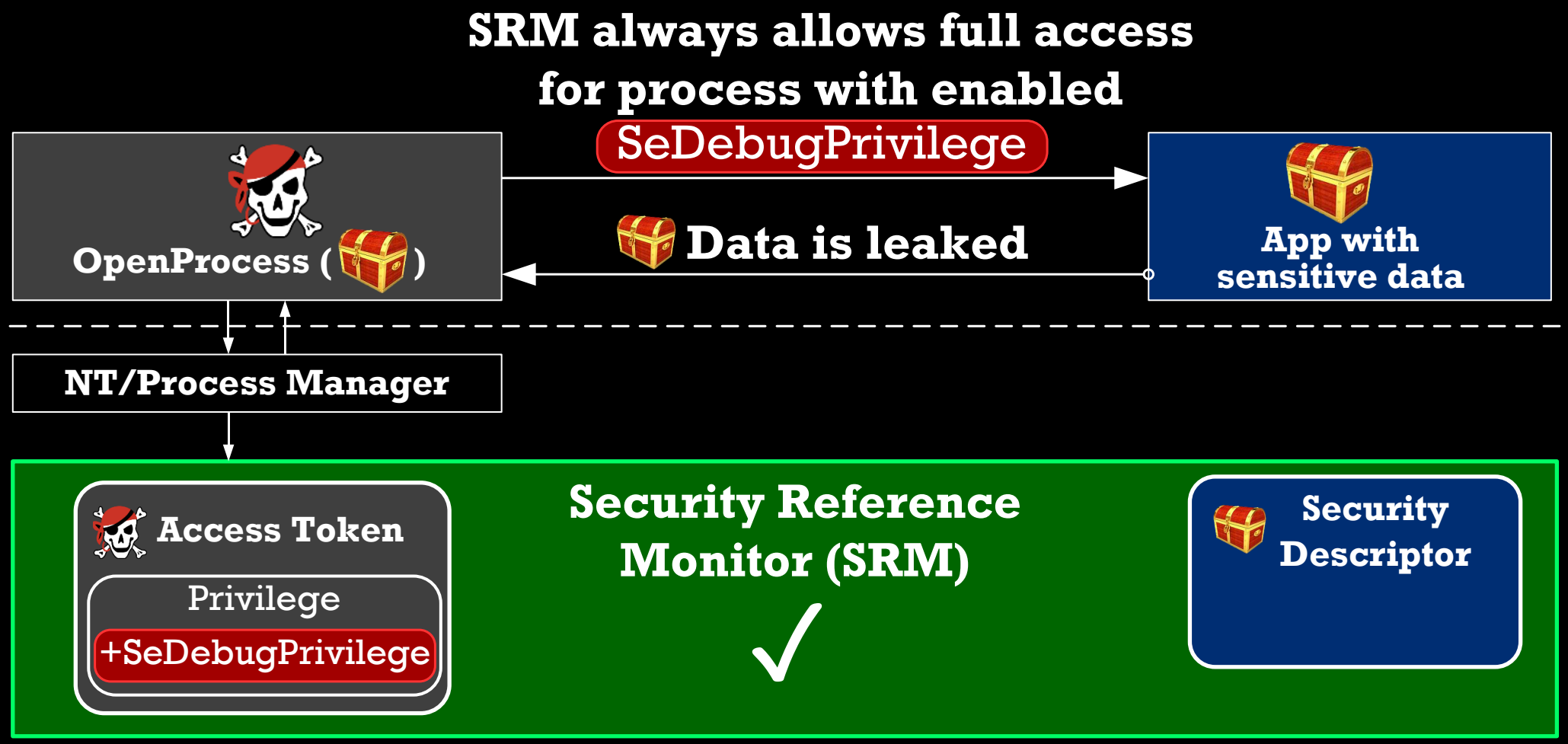
SRM checks access rights using Token and Security Descriptor



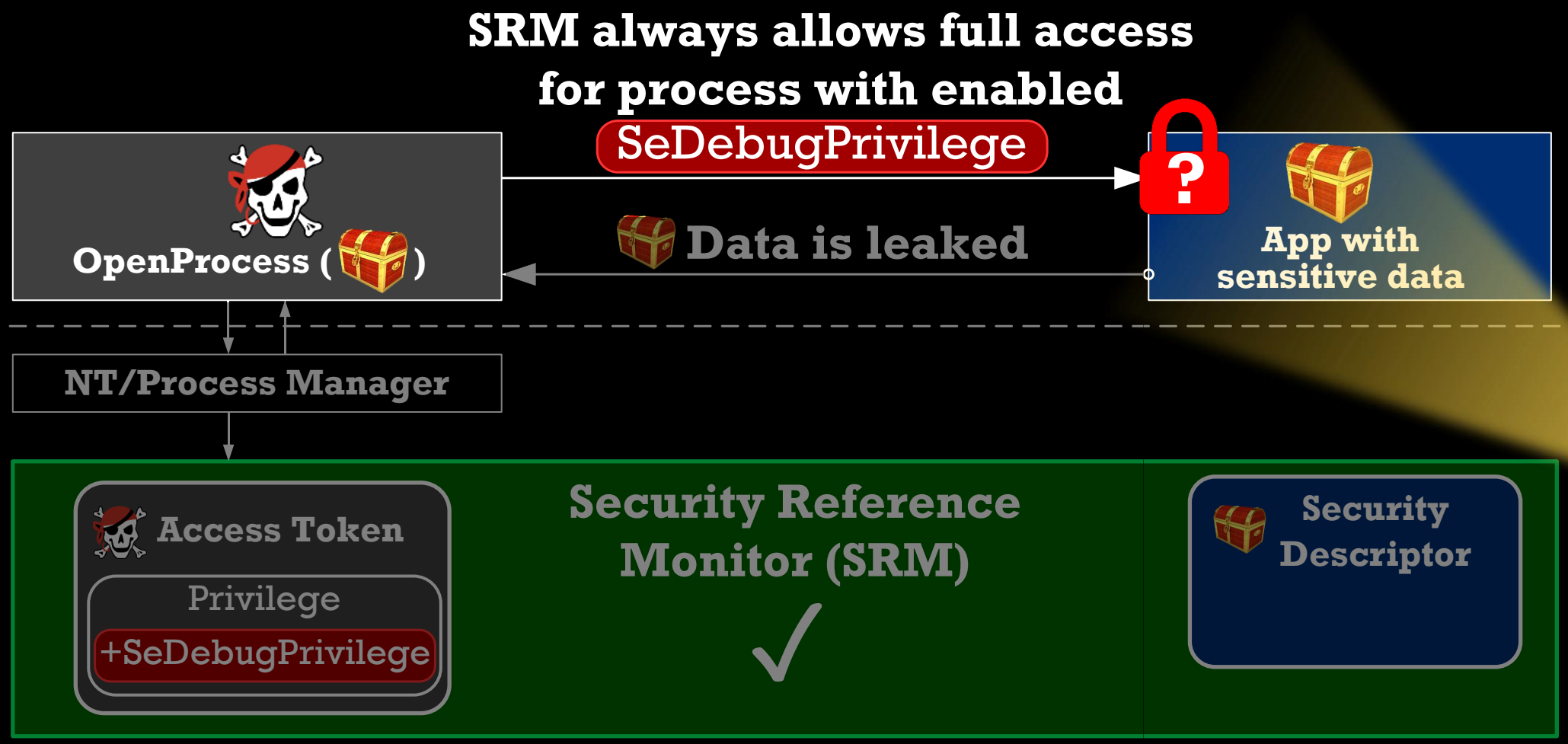
SRM allows full access for app with SeDebugPrivilege



In Windows Security Model any process with admin rights can acquire the debug privilege and access the memory of any process



Malware with enabled debug privilege can steal sensitive data



How to protect data from apps running with debug privilege?

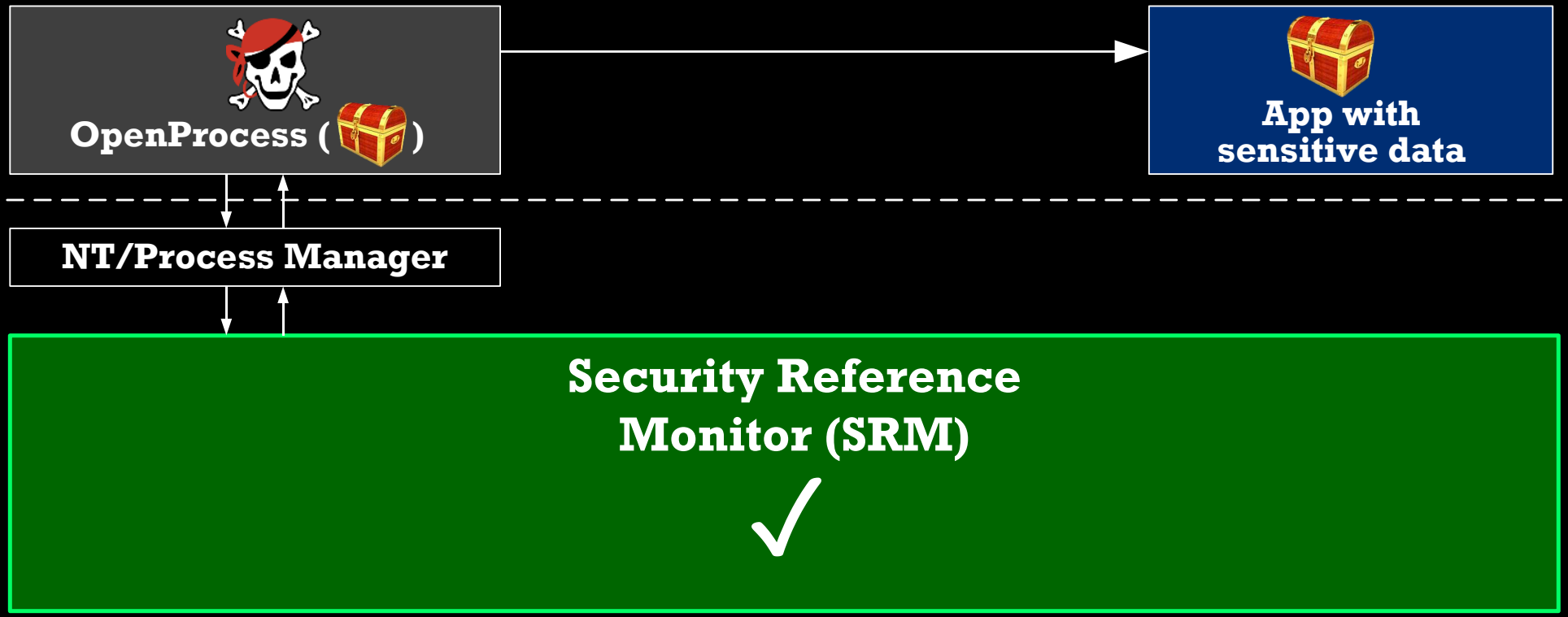
Episode 2

Protected Process Light?
What's that?

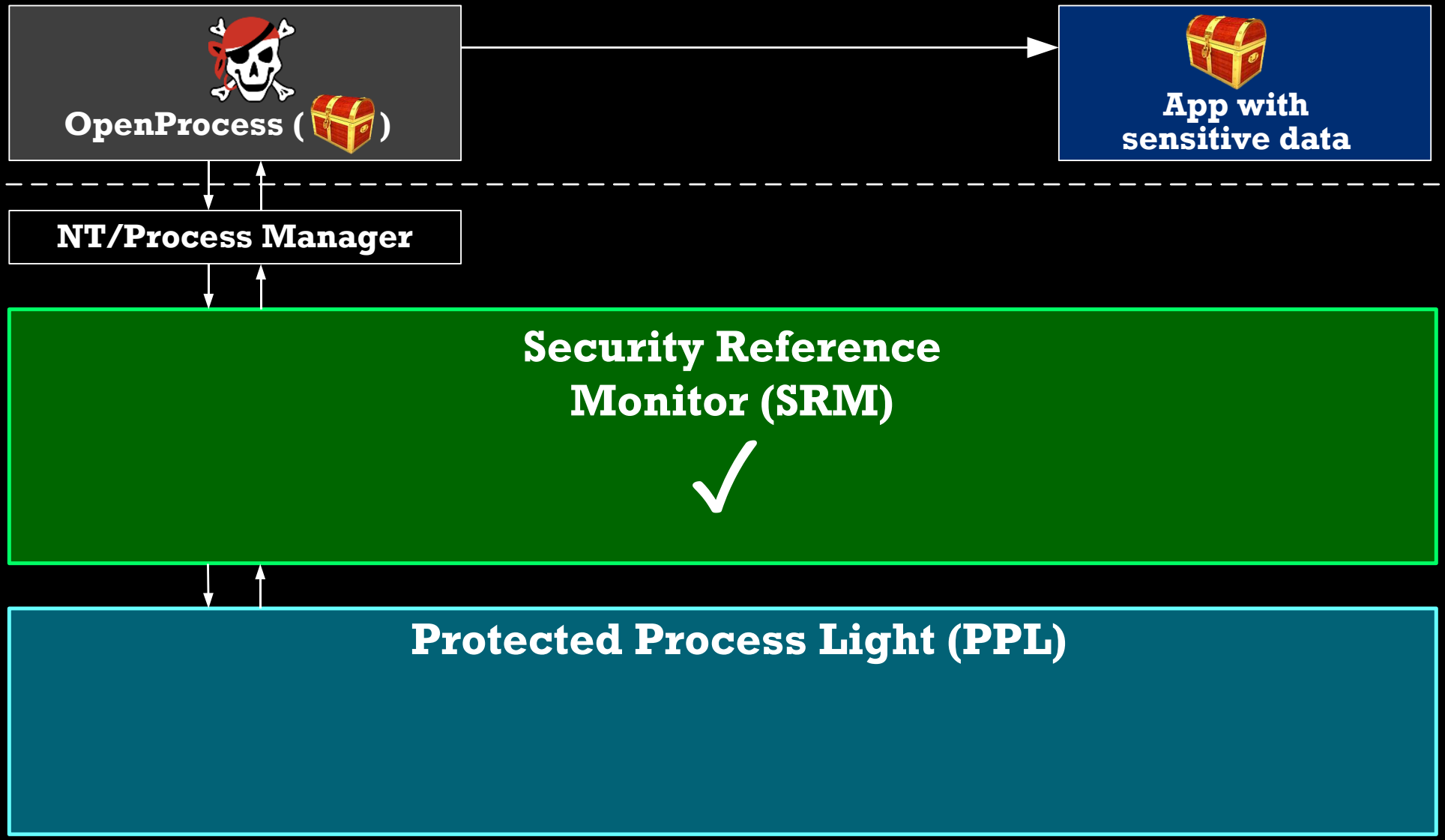
WINDOWS FEATURES TO PROTECT PROCESS MEMORY: PPL

1. Security Reference Monitor (SRM)
2. Protected Process Light (PPL)
3. AppContainer Isolation
4. Windows Resource Protection (WRP, SFC)
5. Session 0 Isolation and Secure Desktop
6. Windows Memory Management (Virtual Memory and Enclave API)
7. Windows Integrity Control (WIC)
8. Mandatory Integrity Control (MIC)
    1. User Interface Privilege Isolation (IUPI)
    2. Enhanced Protected Mode (EPM)
9. Isolated User Mode (IUM) enabled by Hyper-V

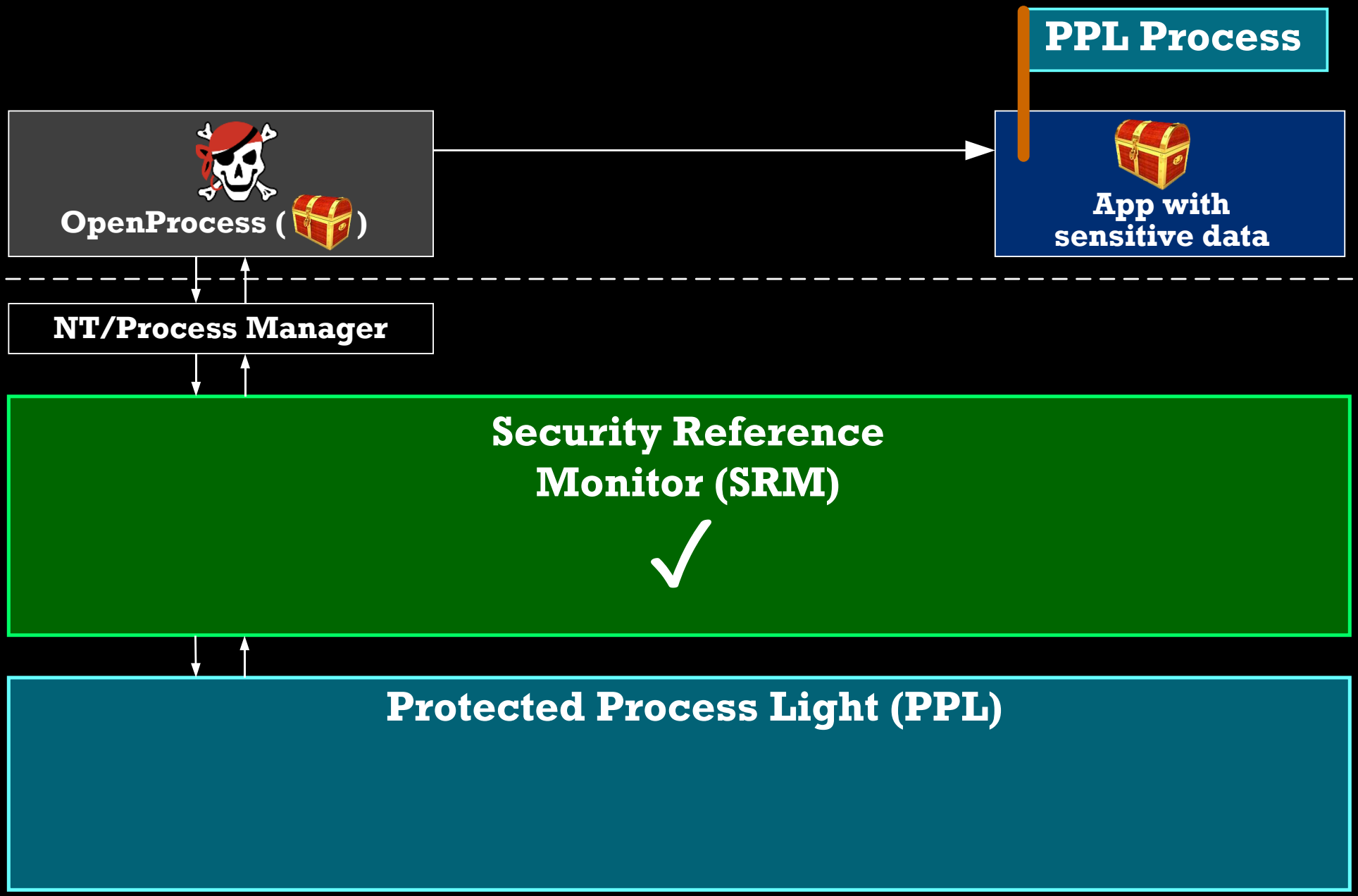
PPL restricts non-PPL apps running with debug privilege



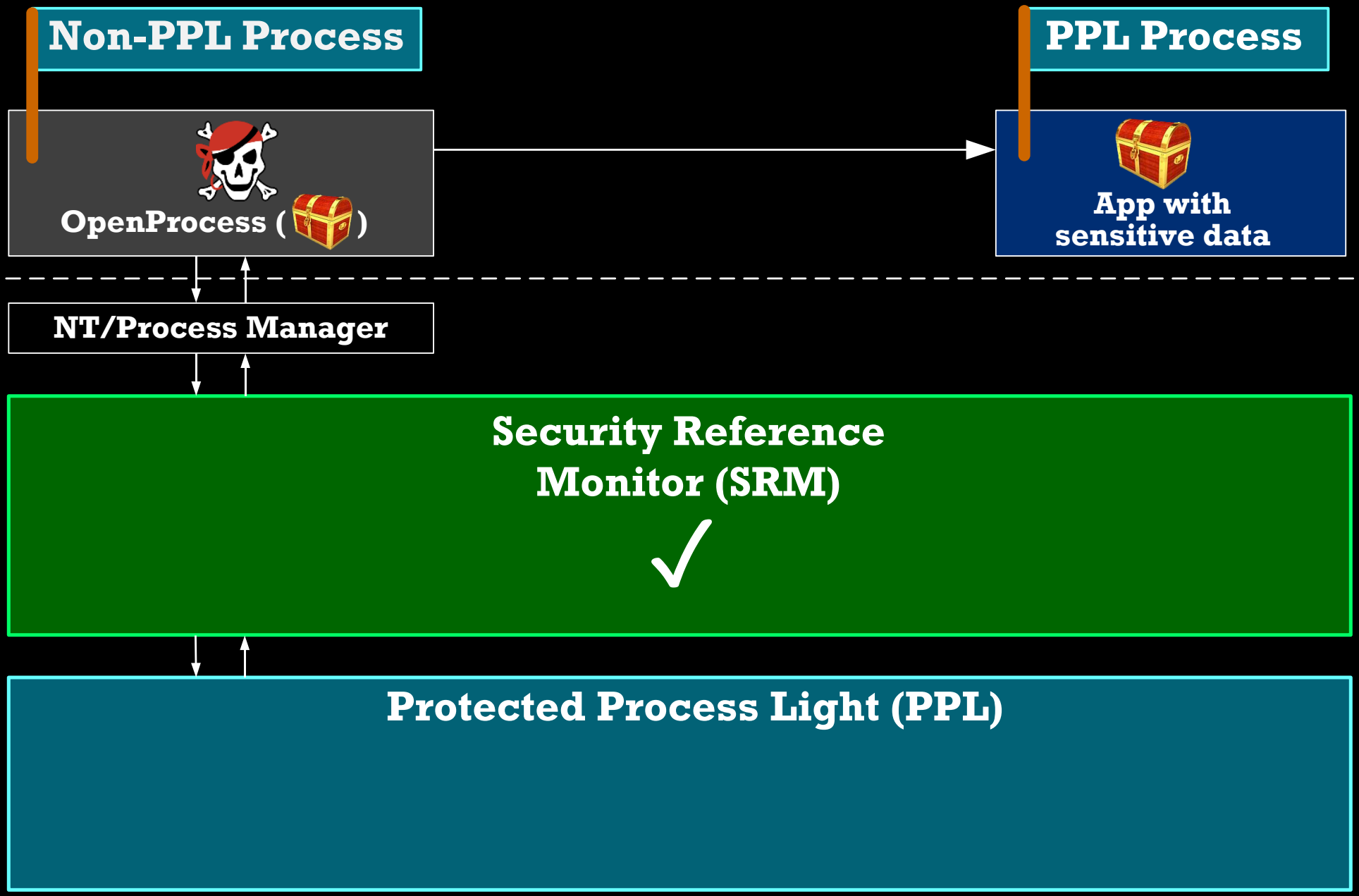
PPL restricts non-PPL apps running with debug privilege



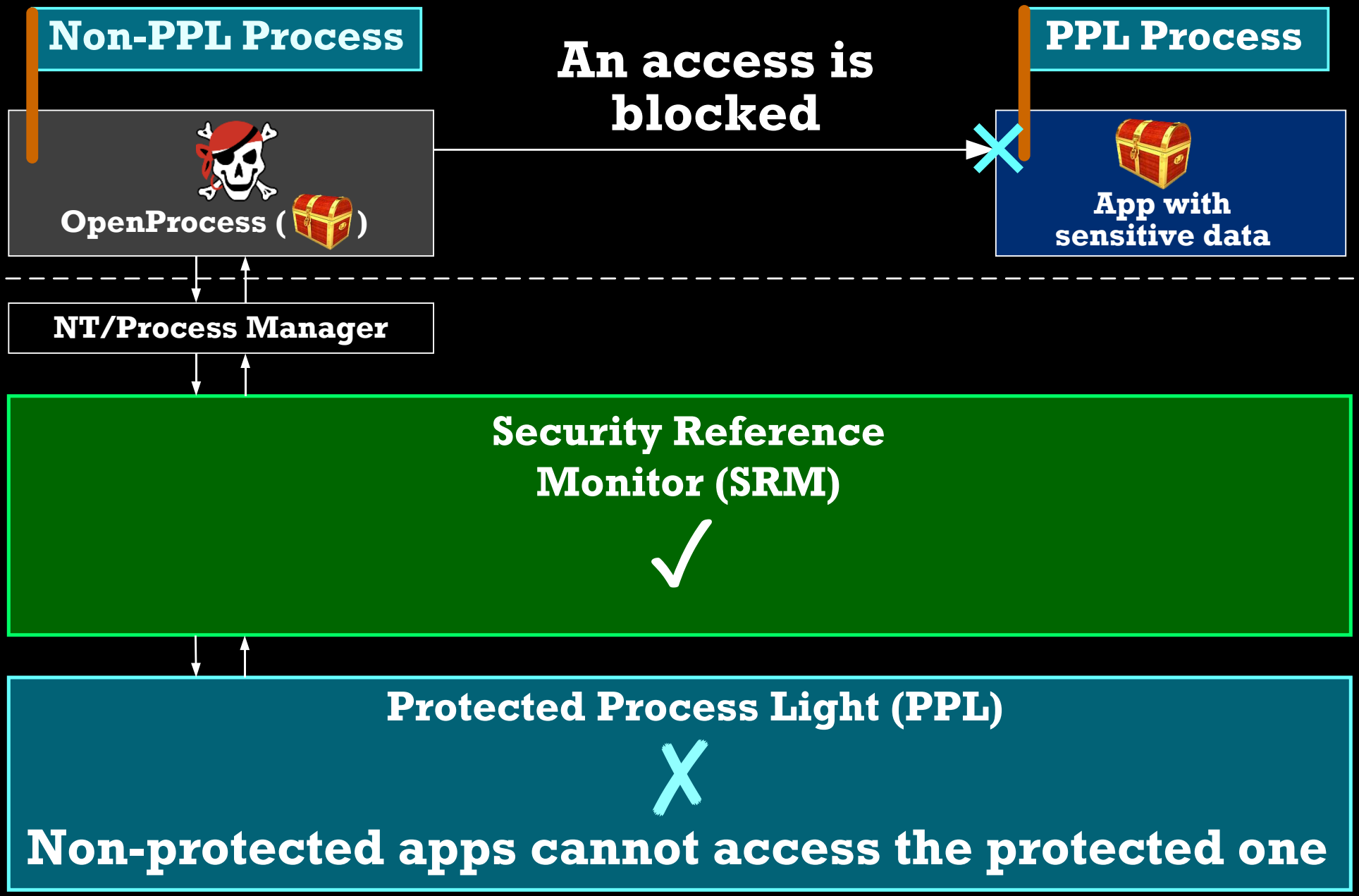
PPL restricts non-PPL apps running with debug privilege



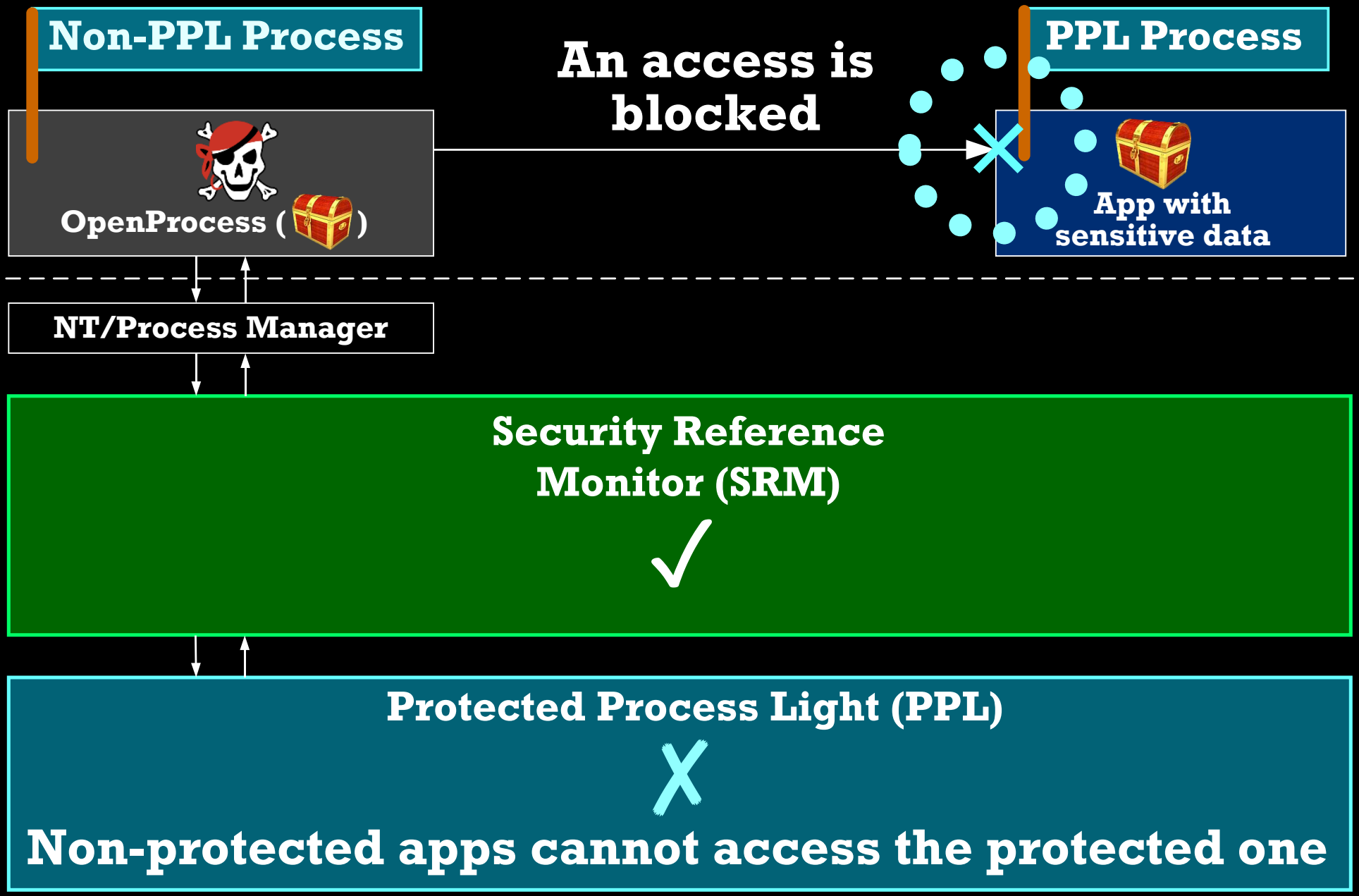
PPL restricts non-PPL apps running with debug privilege



PPL restricts non-PPL apps running with debug privilege



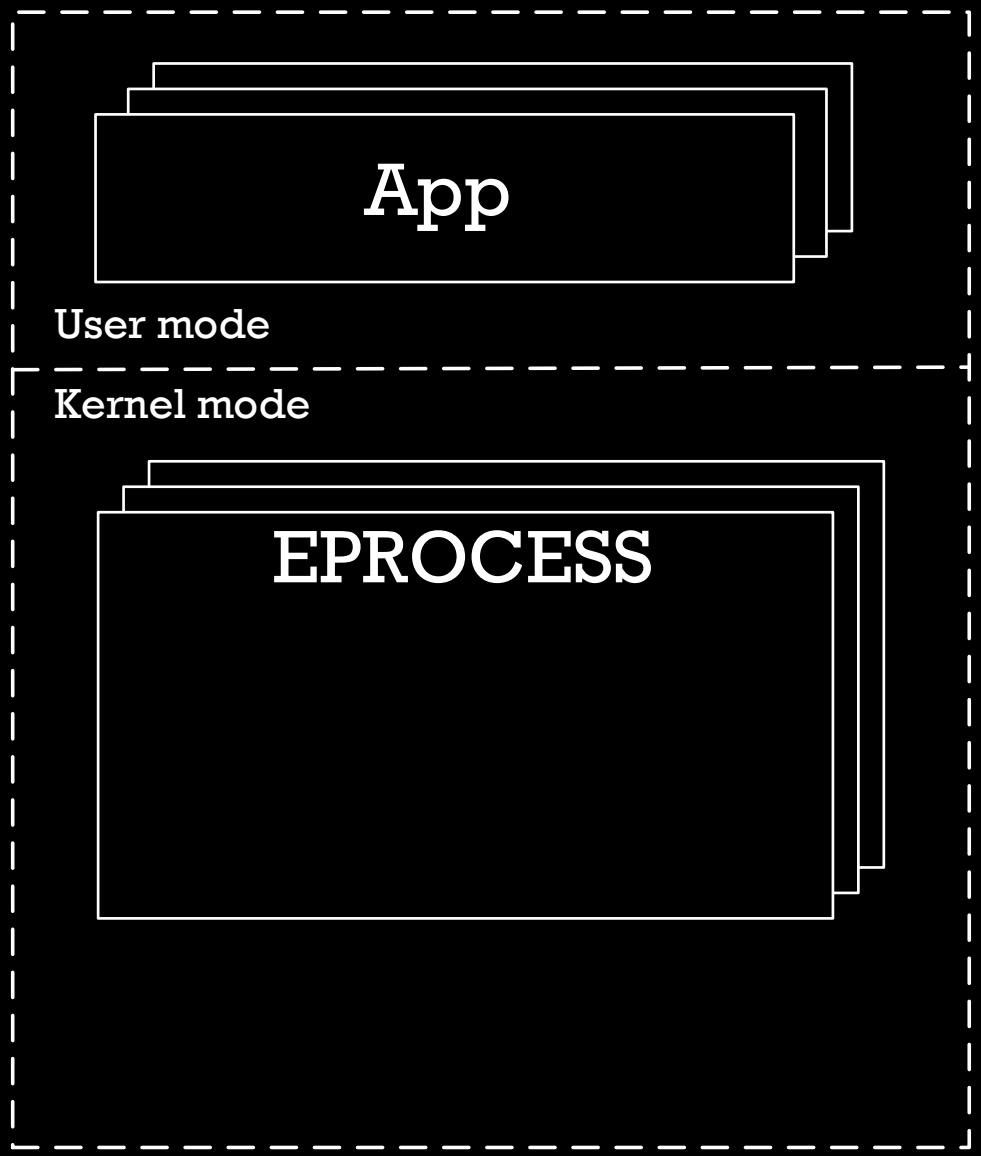
PPL restricts non-PPL apps running with debug privilege



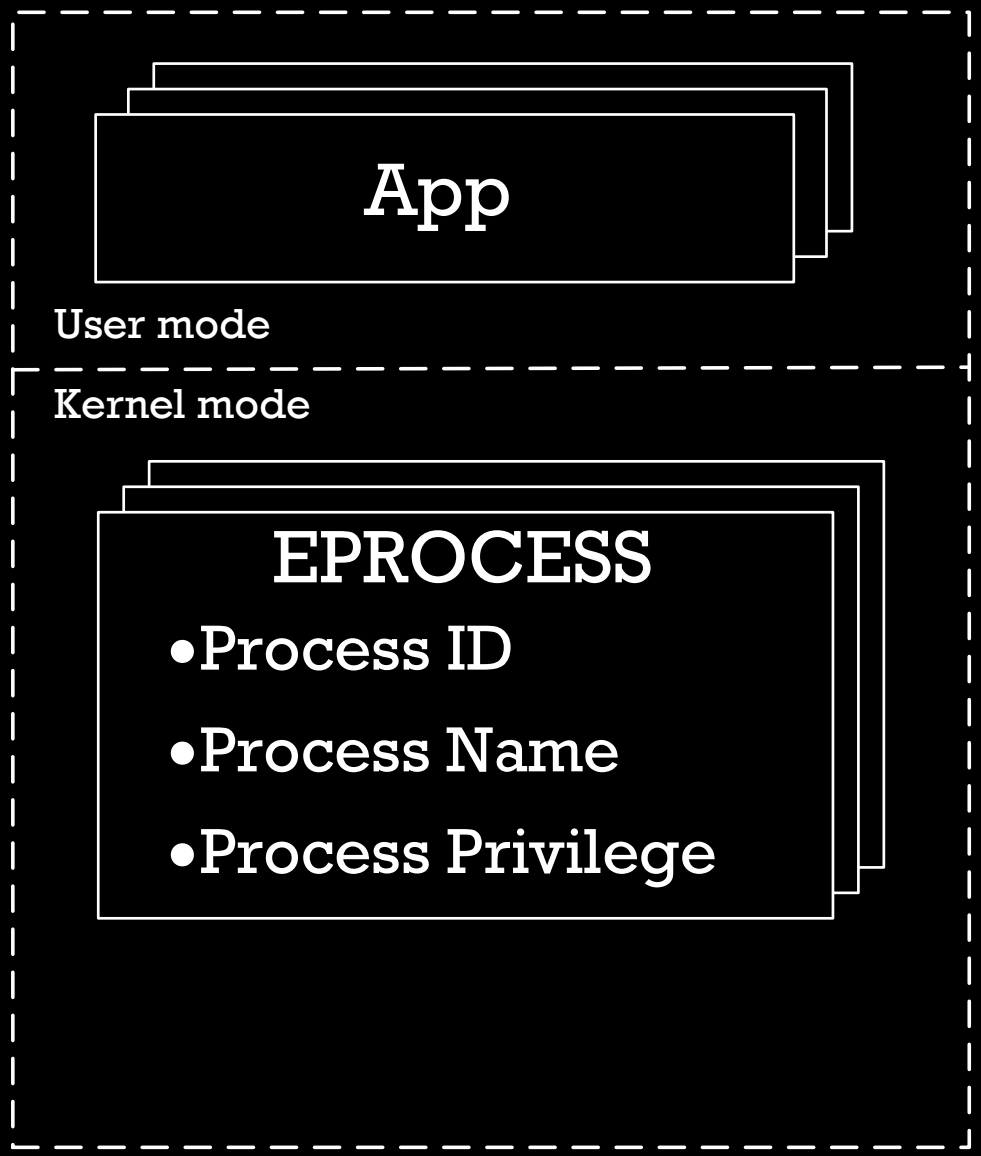
Episode 3

How does PPL work?

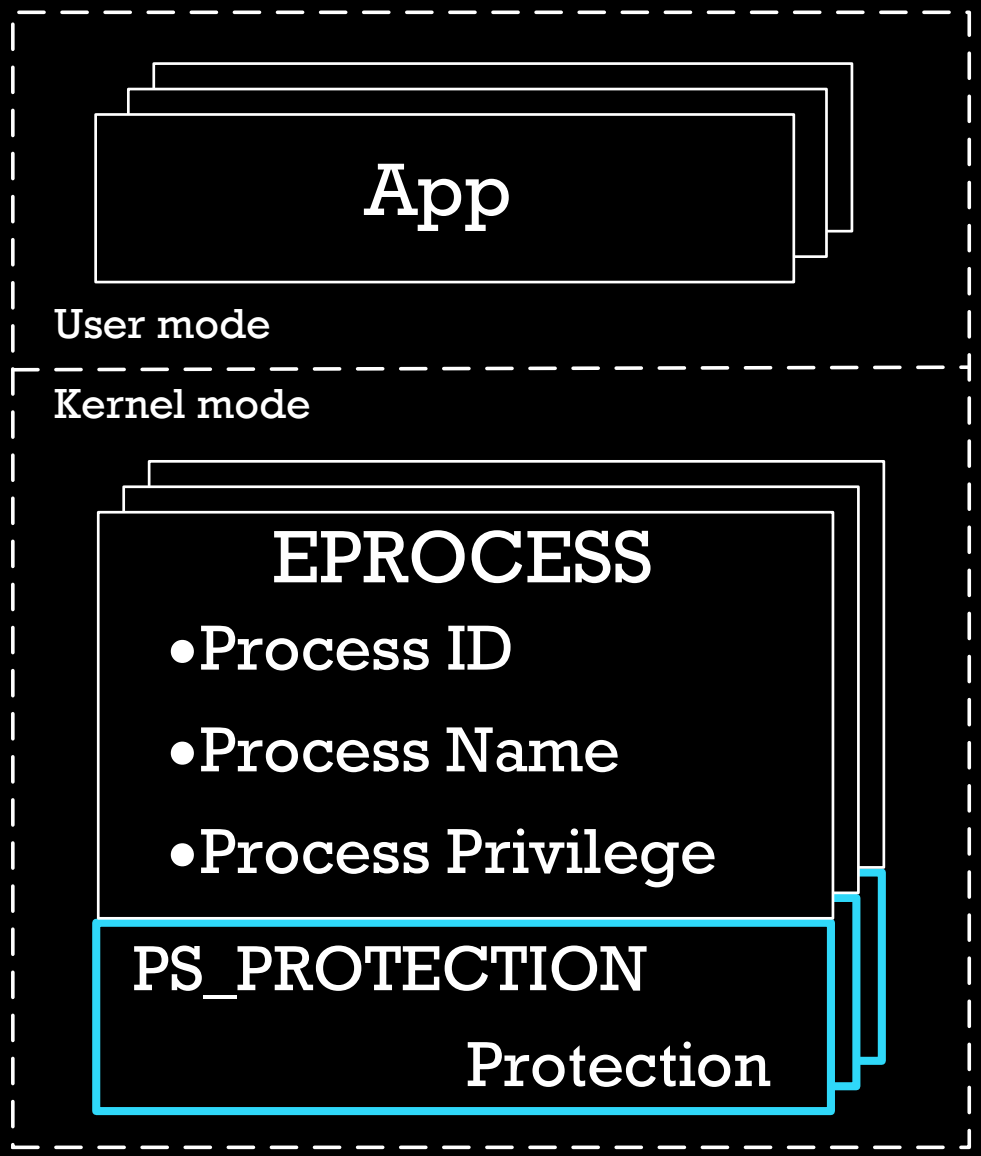
PPL: a new Protection field in EPROCESS



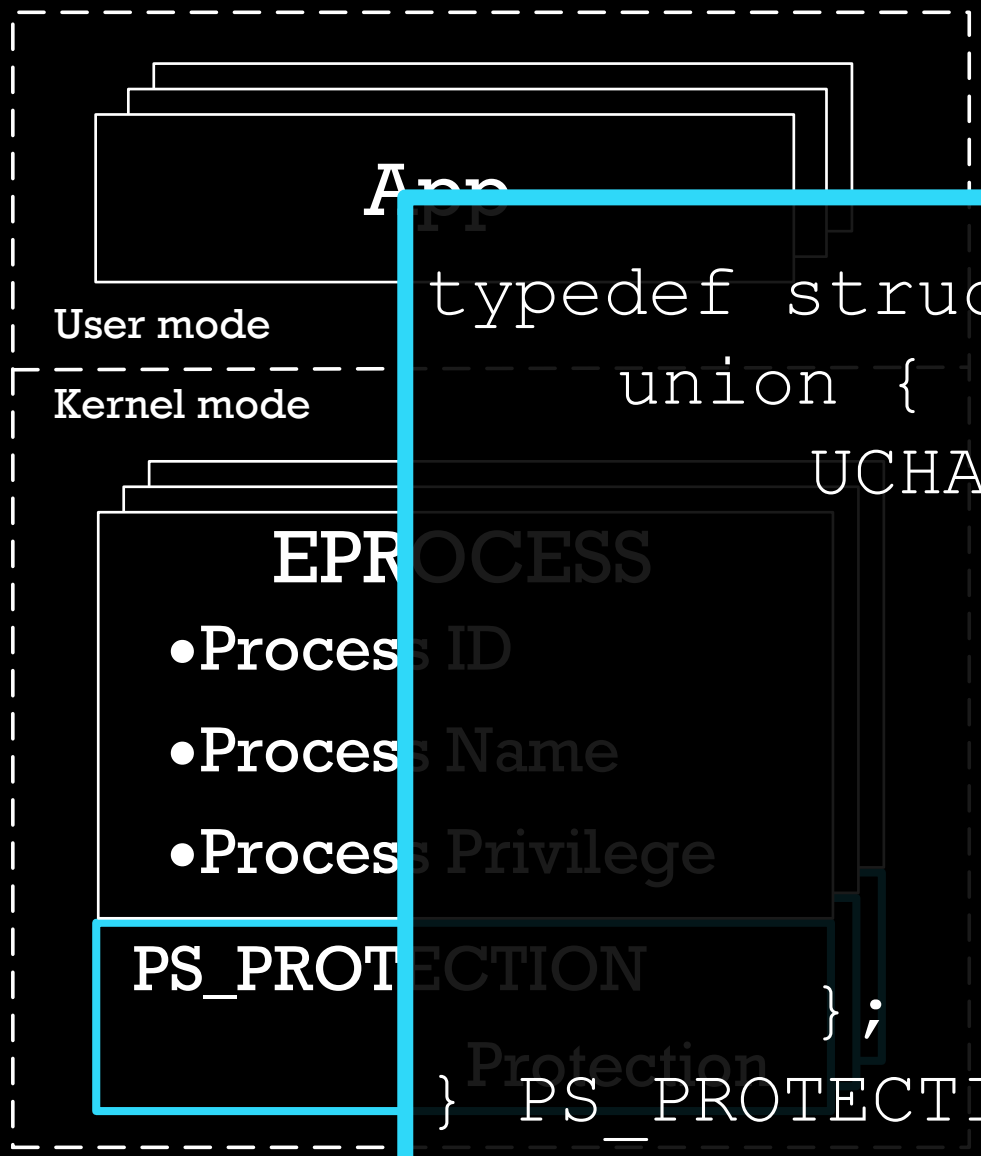
PPL: a new Protection field in EPROCESS



PPL: a new Protection field in EPROCESS

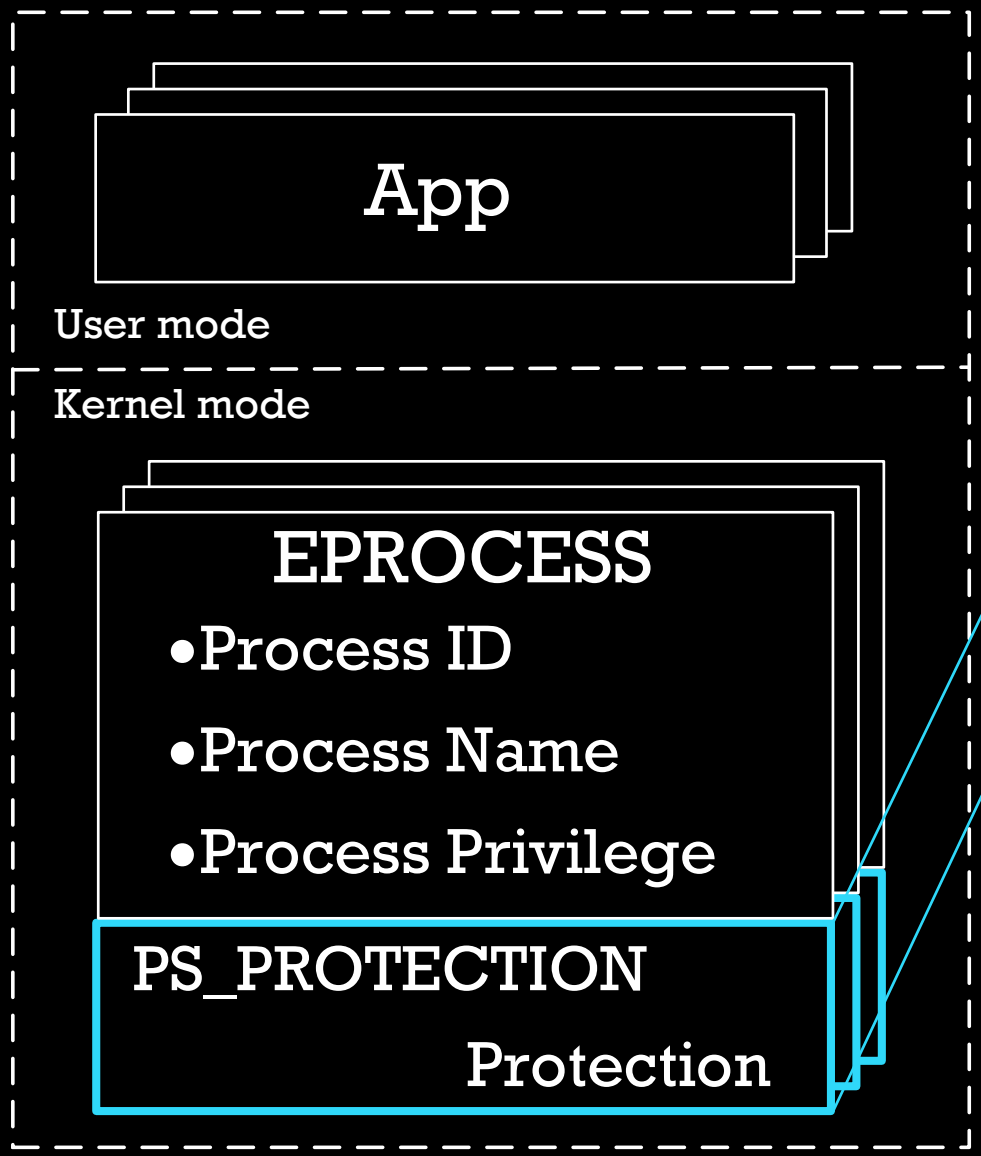


PPL: a new Protection field in EPROCESS




```
typedef struct _PS_PROTECTION {
    union {
        UCHAR Level;
        struct {
            UCHAR Type : 3;
            UCHAR Audit : 1;//<Reserved
            UCHAR Signer : 4;
        };
    };
} PS_PROTECTION, *PPS_PROTECTION;
```

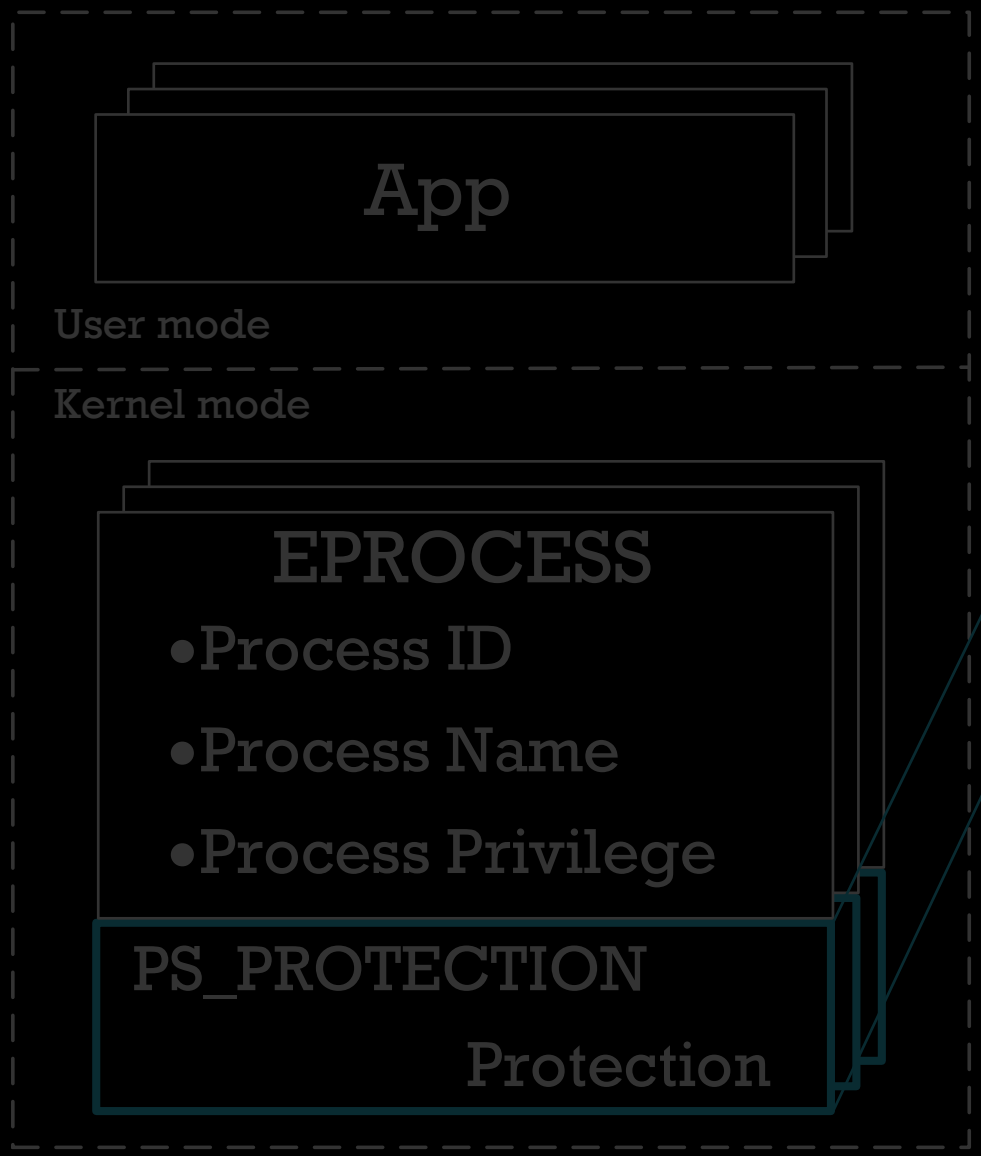
PPL: a new Protection field in EPROCESS



PS_PROTECTION Protection

Signer	Type

PPL: a new Protection field in EPROCESS



PS_PROTECTION Protection

Signer	Type
0 – None 1 – Authenticode 2 – CodeGen 3 – Antimalware 4 – Lsa 5 – Windows 6 – WinTcb 7 – WinSystem 8 – SignerApp	0 – None 1 – Light 2 – Full

EXAMPLES OF PROTECTION LEVEL

Process name	Protection Level
NisSrv	0x31
LSASS	0x41
SgrmBroker	0x62

NisSrv – Microsoft Network Realtime Inspection Service.
LSASS – Local Security Authority Subsystem Service.
SgrmBroker – System Guard Runtime Monitor Broker.

EXAMPLES OF PROTECTION LEVEL

Process name	Protection Level	Signer	Type
NisSrv	0x31	3 (Antimalware)	1 (Light)
LSASS	0x41	4 (Lsa)	1 (Light)
SgrmBroker	0x62	6 (WinTcb)	2 (Full)

NisSrv – Microsoft Network Realtime Inspection Service.
LSASS – Local Security Authority Subsystem Service.
SgrmBroker – System Guard Runtime Monitor Broker.

EXAMPLES OF PROTECTION LEVEL

Process name	Protection Level	Signer	Type
NisSrv	0x31 	3 (Antimalware)	1 (Light)
LSASS	0x41 	4 (Lsa)	1 (Light)
SgrmBroker	0x62 	6 (WinTcb)	2 (Full)

NisSrv – Microsoft Network Realtime Inspection Service.
LSASS – Local Security Authority Subsystem Service.
SgrmBroker – System Guard Runtime Monitor Broker.

Episode 4

How does Windows create protected processes?

PPL: CreateProcess

"System"

"MemCompression"

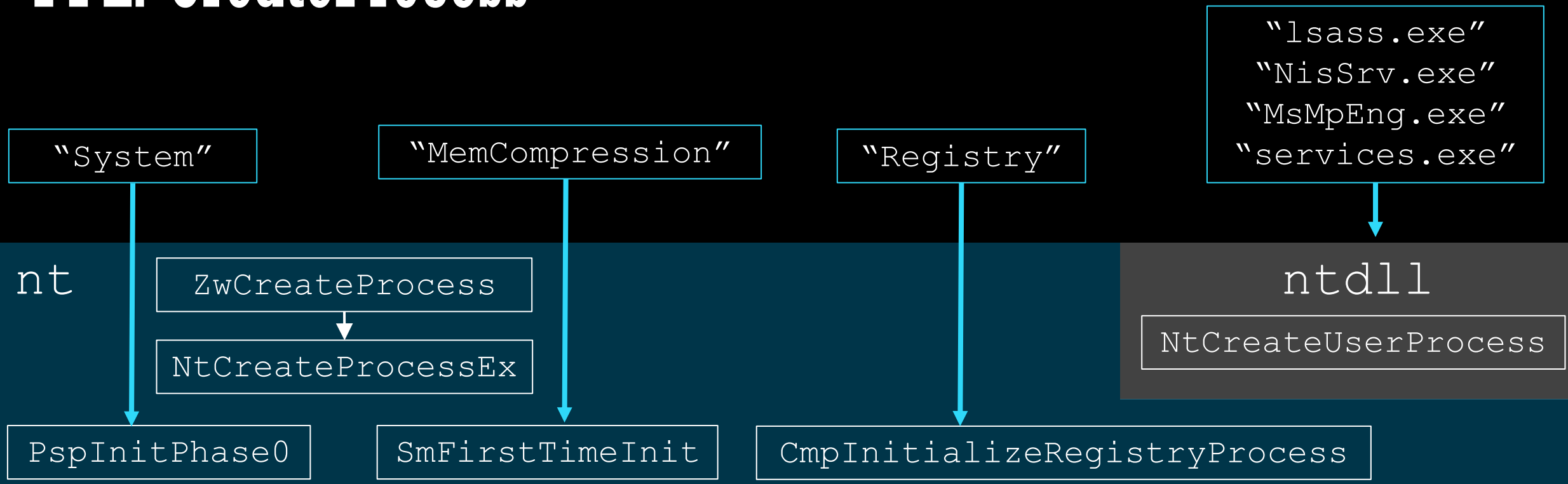
"Registry"

"lsass.exe"
"NisSrv.exe"
"MsMpEng.exe"
"services.exe"

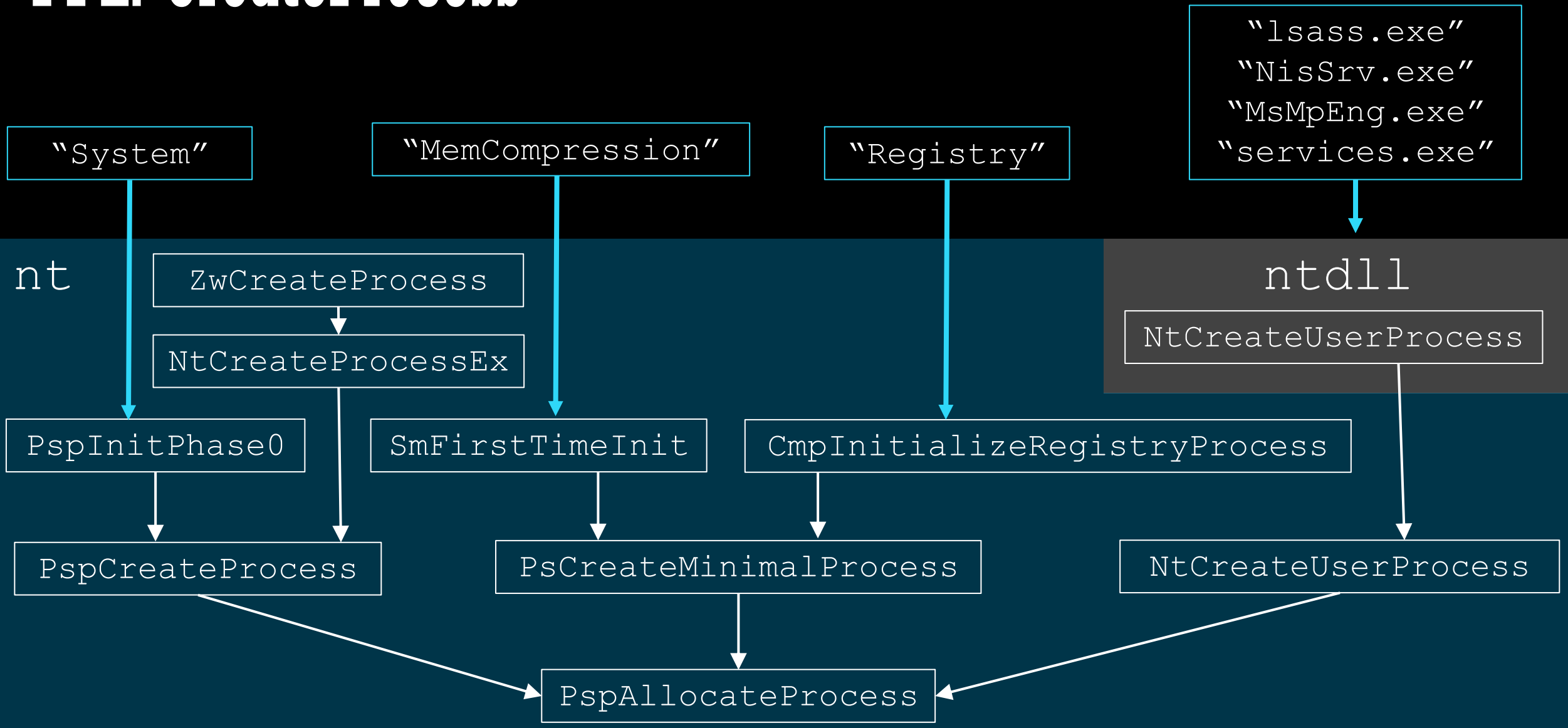
PPL: CreateProcess

Which OS functions are involved during creating of Protected Processes?

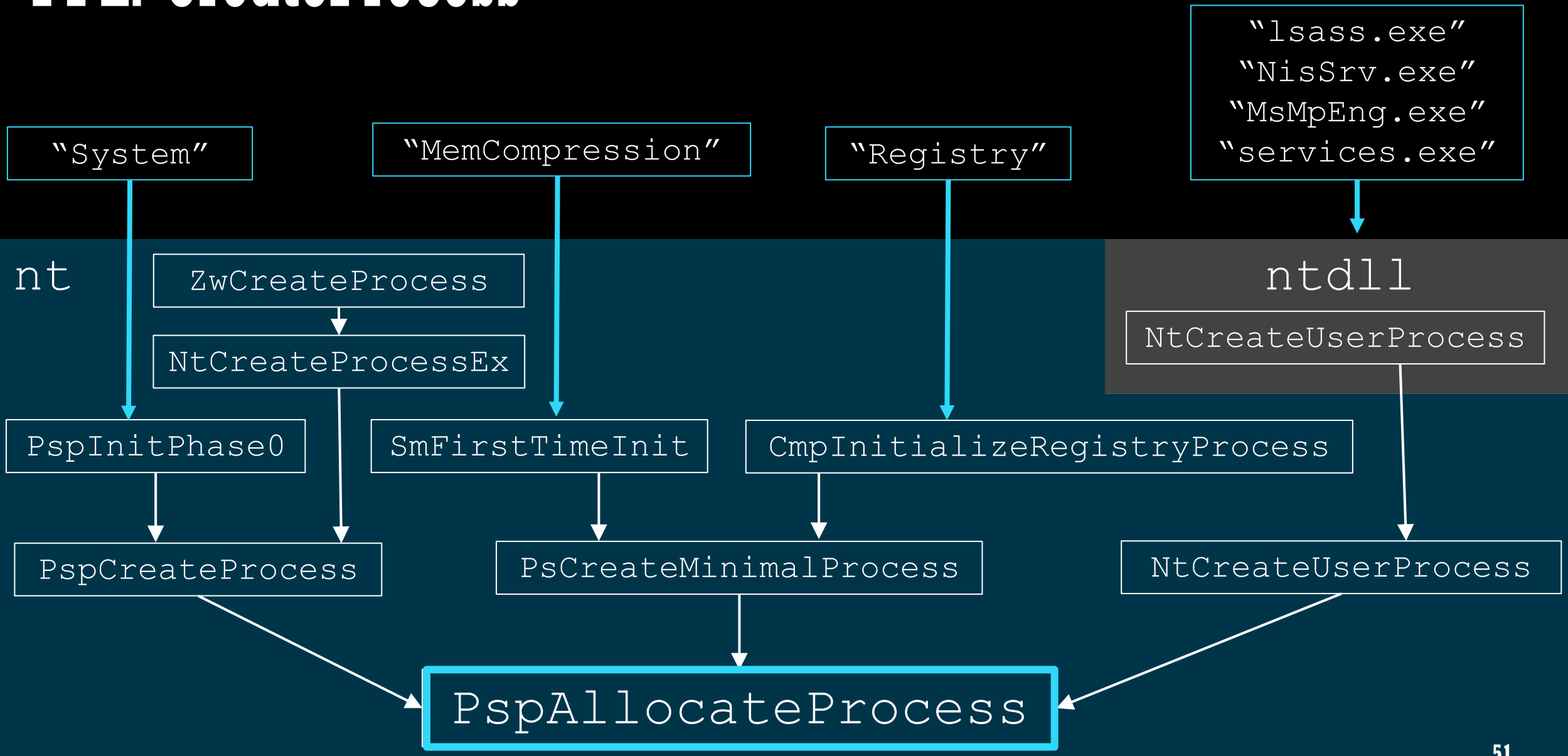
PPL: CreateProcess



PPL: CreateProcess



PPL: CreateProcess



PPL: PspAllocateProcess sets Protection.Level for the process

```
PspAllocateProcess:
      mov     rdx, [nt!PsProcessType]
      call    nt!ObCreateObject          ; allocates EPROCESS structure
      ..
      mov    r15,qword ptr [rsp+0B8h] ;  r15 is address of created EPROCESS
      mov    byte ptr [r15+87Ah], dil ;  87Ah is offset of Protection.Level
                                      ;  dil is the Protection.level
```

```
Command: dx -r1 (*((ntkrnlmp!_PS_PROTECTION
(*((ntkrnlmp!_PS_PROTECTION *)@r15+0x87A))
    [+0x000] Level              : 0x0 [Type: u
    [+0x000 ( 2: 0)] Type               : 0x0
    [+0x000 ( 3: 3)] Audit              : 0x0
    [+0x000 ( 7: 4)] Signer             : 0x0
```

Protection = 0x0

```
Command: dx -r1 (*((ntkrnlmp!_PS_PROTECTION
(*((ntkrnlmp!_PS_PROTECTION *)@r15+0x87A))
    [+0x000] Level              : 0x72 [Type:
    [+0x000 ( 2: 0)] Type               : 0x2
    [+0x000 ( 3: 3)] Audit              : 0x0
    [+0x000 ( 7: 4)] Signer             : 0x7
```

Protection = 0x72

PPL: PspAllocateProcess sets Protection.Level for the process

```
PspAllocateProcess:
      mov     rdx, [nt!PsProcessType]
      call    nt!ObCreateObject          ; allocates EPROCESS structure
      ..
      mov     r15,qword ptr [rsp+0B8h] ;  r15 is address of created EPROCESS
      mov     byte ptr [r15+87Ah], dil ;  87Ah is offset of Protection.Level
                                       ;  dil is the Protection.level
```

```
Command: dx -r1 (*((ntkrnlmp!_PS_PROTECTION
(*((ntkrnlmp!_PS_PROTECTION *)@r15+0x87A))
    [+0x000] Level              : 0x0 [Type: u
    [+0x000 ( 2: 0)] Type               : 0x0
    [+0x000 ( 3: 3)] Audit              : 0x0
    [+0x000 ( 7: 4)] Signer             : 0x0
```

Protection = 0x0

```
Command: dx -r1 (*((ntkrnlmp!_PS_PROTECTION
(*((ntkrnlmp!_PS_PROTECTION *)@r15+0x87A))
    [+0x000] Level              : 0x72 [Type:
    [+0x000 ( 2: 0)] Type               : 0x2
    [+0x000 ( 3: 3)] Audit              : 0x0
    [+0x000 ( 7: 4)] Signer             : 0x7
```

Protection = 0x72

PPL: PspAllocateProcess sets Protection.Level for the process

```
PspAllocateProcess:
      mov     rdx, [nt!PsProcessType]
      call    nt!ObCreateObject          ; allocates EPROCESS structure
      ..
      mov    r15,qword ptr [rsp+0B8h] ;  r15 is address of created EPROCESS
      mov    byte ptr [r15+87Ah], dil ;  87Ah is offset of Protection.Level
                                      ;  dil is the Protection.level
```

```
Command: dx -r1 (*((ntkrnlmp!_PS_PROTECTION
(*((ntkrnlmp!_PS_PROTECTION *)@r15+0x87A))
    [+0x000] Level              : 0x0 [Type: u
    [+0x000 ( 2: 0)] Type               : 0x0
    [+0x000 ( 3: 3)] Audit              : 0x0
    [+0x000 ( 7: 4)] Signer             : 0x0
```

Protection = 0x0

```
Command: dx -r1 (*((ntkrnlmp!_PS_PROTECTION
(*((ntkrnlmp!_PS_PROTECTION *)@r15+0x87A))
    [+0x000] Level              : 0x72 [Type:
    [+0x000 ( 2: 0)] Type               : 0x2
    [+0x000 ( 3: 3)] Audit              : 0x0
    [+0x000 ( 7: 4)] Signer             : 0x7
```

Protection = 0x72

PPL: PspAllocateProcess sets Protection.Level for the process

```
PspAllocateProcess:
        mov     rdx, [nt!PsProcessType]
        call    nt!ObCreateObject          ; allocates EPROCESS structure
        ..
        mov     r15,qword ptr [rsp+0B8h] ;  r15 is address of created EPROCESS
        mov     byte ptr [r15+87Ah], dil ;  87Ah is offset of Protection.Level
                                         ;  dil is the Protection.level
```

```
Command: dx -r1 (*((ntkrnlmp!_PS_PROTECTION
(*((ntkrnlmp!_PS_PROTECTION *)@r15+0x87A))
    [+0x000] Level              : 0x0 [Type: u
    [+0x000 ( 2: 0)] Type       : 0x0
    [+0x000 ( 3: 3)] Audit      : 0x0
    [+0x000 ( 7: 4)] Signer     : 0x0
```

Protection.Level = 0x0

```
Command: dx -r1 (*((ntkrnlmp!_PS_PROTECTION
(*((ntkrnlmp!_PS_PROTECTION *)@r15+0x87A))
    [+0x000] Level              : 0x72 [Type:
    [+0x000 ( 2: 0)] Type       : 0x2
    [+0x000 ( 3: 3)] Audit      : 0x0
    [+0x000 ( 7: 4)] Signer     : 0x7
```

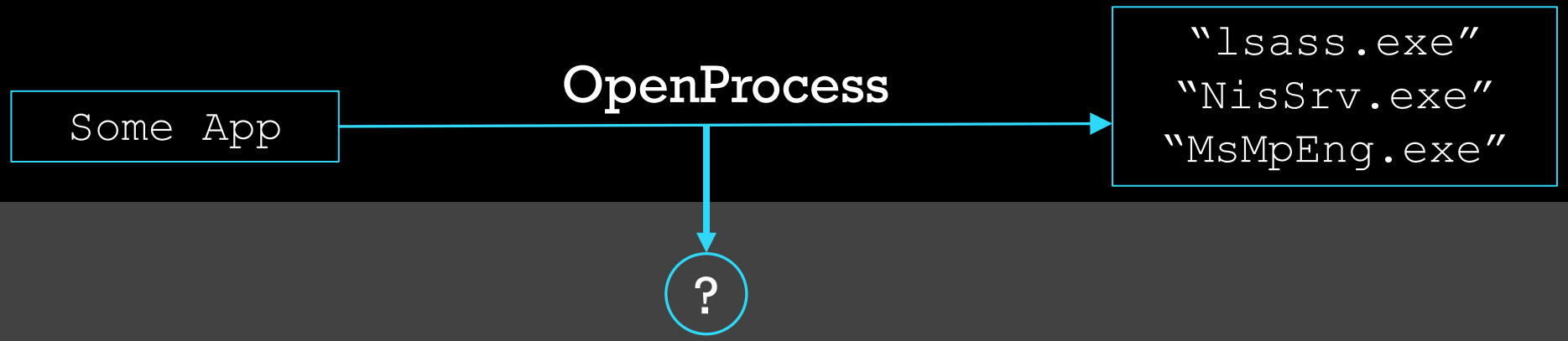
Protection.Level = 0x72

Episode 5

How to get access to the protected process memory?

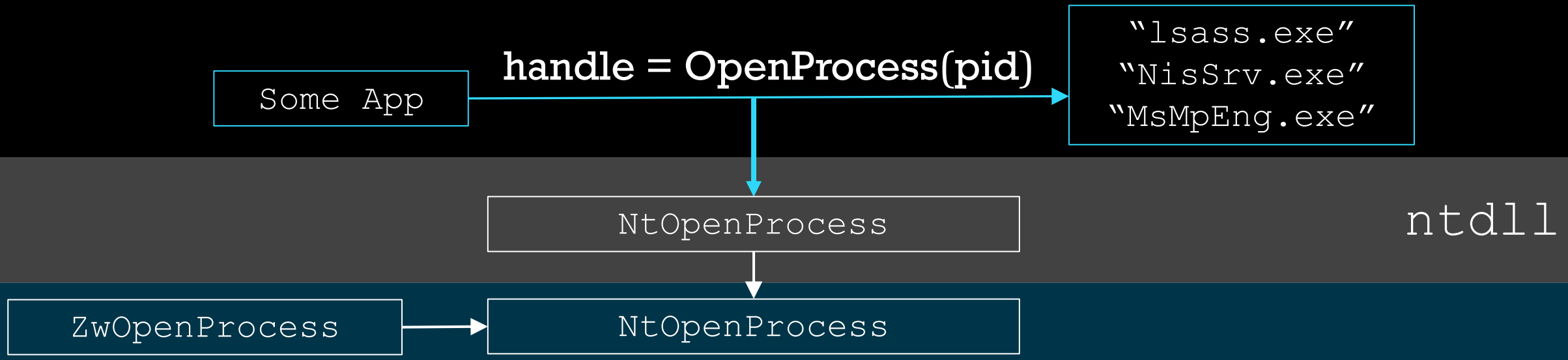
PPL: OpenProcess

PPL: OpenProcess

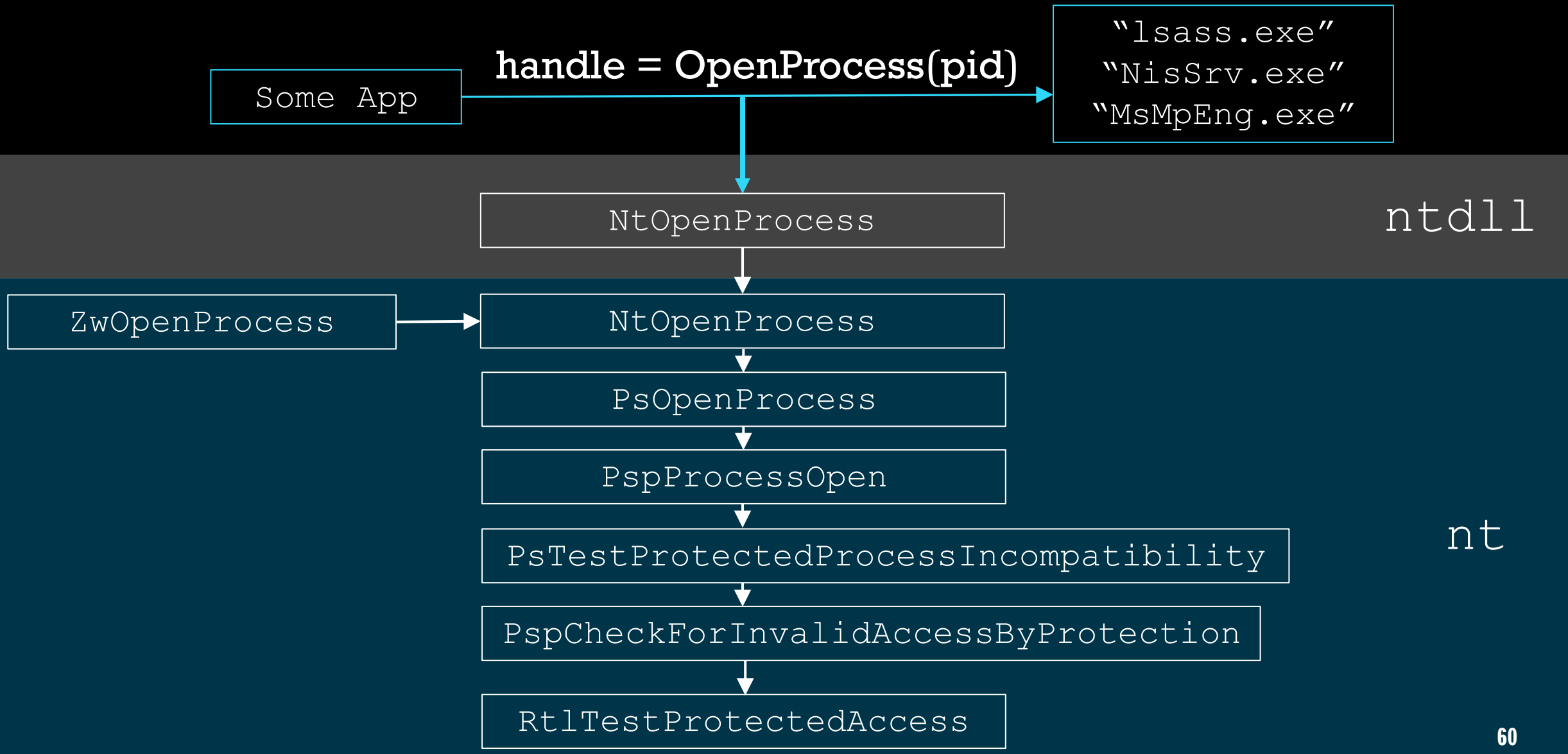


Which OS functions are involved in openning of protected processes?

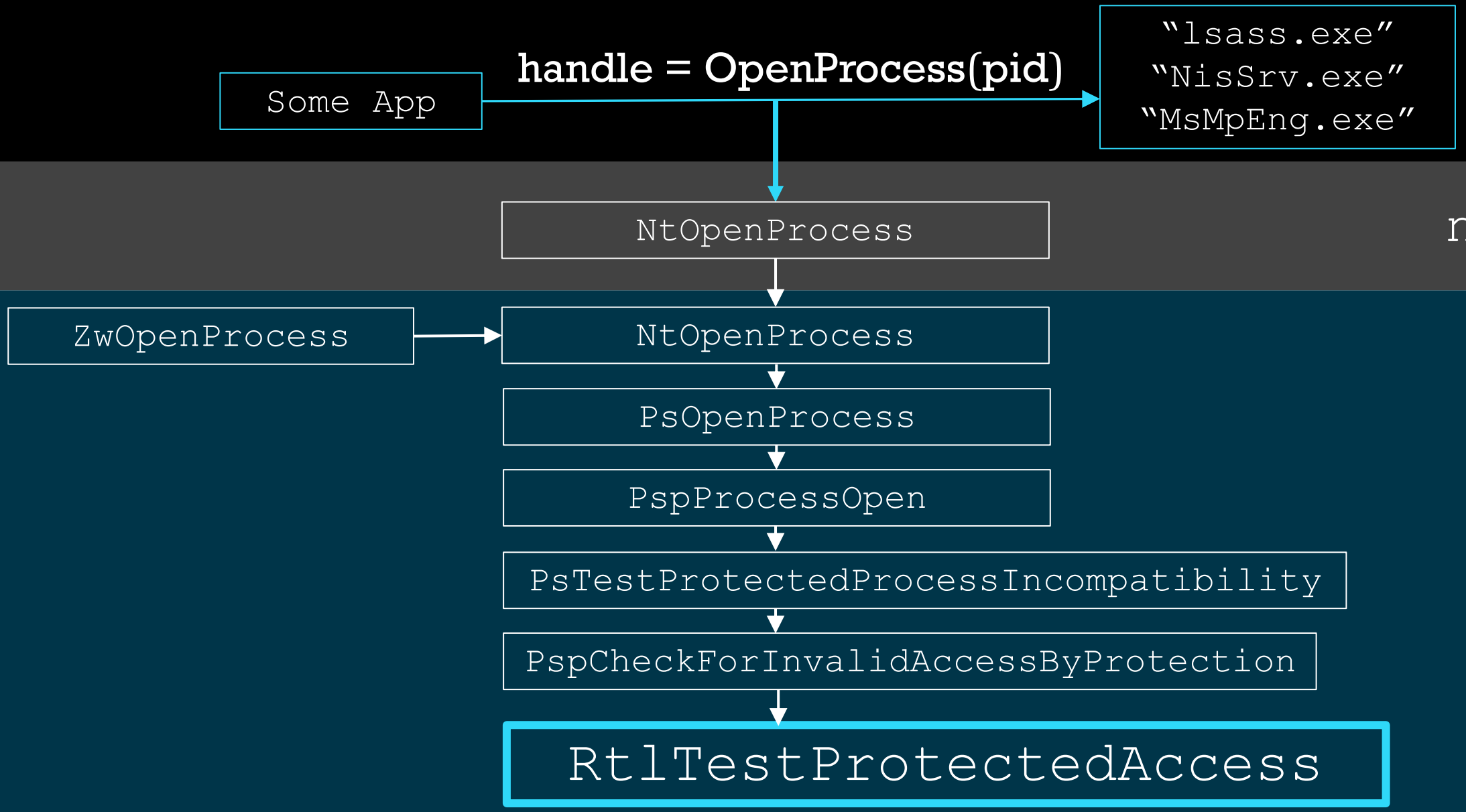
PPL: OpenProcess



PPL: OpenProcess



PPL: OpenProcess → RtlTestProtectedAccess



PPL: RtlTestProtectedAccess()

```
bool RtlTestProtectedAccess(PS_PROTECTION  CallerProt, PS_PROTECTION TargetProt)
{
        if (TargetProt.Type == 0)
                return true;

        if (CallerProt.Type < TargetProt.Type)
                return false;

        auto CallerDMask   = RtlProtectedAccess[CallerProt.Signer].DominateMask;
        auto TargetMask    = (1 << TargetProt.Signer);
        return (CallerDMask & TargetMask);
}
```

https://github.com/IgorKorkin/PPL

PPL: RtlTestProtectedAccess()

```
bool RtlTestProtectedAccess(PS_PROTECTION  CallerProt, PS_PROTECTION TargetProt)
{
        if (TargetProt.Type == 0)
                return true;

        if (CallerProt.Type < TargetProt.Type)
                return false;

        auto CallerDMask   = RtlProtectedAccess[CallerProt.Signer].DominateMask;
        auto TargetMask    = (1 << TargetProt.Signer);
        return (CallerDMask & TargetMask);
}
```

https://github.com/IgorKorkin/PPL

PPL: RtlTestProtectedAccess()

```
bool RtlTestProtectedAccess(PS_PROTECTION  CallerProt, PS_PROTECTION TargetProt)
{
    if (TargetProt.Type == 0)
        return true;

    if (CallerProt.Type < TargetProt.Type)
        return false;

    auto CallerDMask   = RtlProtectedAccess[CallerProt.Signer].DominateMask;
    auto TargetMask    = (1 << TargetProt.Signer);
    return (CallerDMask & TargetMask);
}
```

https://github.com/IgorKorkin/PPL

PPL: RtlTestProtectedAccess()

```
bool RtlTestProtectedAccess(PS_PROTECTION  CallerProt, PS_PROTECTION TargetProt)
{
        if (TargetProt.Type == 0)
                return true;

        if (CallerProt.Type < TargetProt.Type)
                return false;

        auto CallerDMask   = RtlProtectedAccess[CallerProt.Signer].DominateMask;
        auto TargetMask    = (1 << TargetProt.Signer);
        return (CallerDMask & TargetMask);
}
```

https://github.com/IgorKorkin/PPL

PPL: RtlTestProtectedAccess()

```
bool RtlTestProtectedAccess(PS_PROTECTION  CallerProt, PS_PROTECTION TargetProt)
{
        if (TargetProt.Type == 0)
                return true;

        if (CallerProt.Type < TargetProt.Type)
                return false;

        auto CallerDMask   = RtlProtectedAccess[CallerProt.Signer].DominateMask;
        auto TargetMask    = (1 << TargetProt.Signer);
        return (CallerDMask & TargetMask);
}
```

https://github.com/IgorKorkin/PPL

RtlProtectedAccess Array

Index	Signer	DominateMask
0	none	0
1	Authenticode	2
2	CodeGen	4
3	Antimalware	0x108
4	Lsa	0x110
5	Windows	0x13e
6	WinTCB	0x17e
7	WinSystem	0x1fe
8	SignerApp	0

RtlProtectedAccess Array

Index	Signer	DominateMask	Bit Explanation
0	none	0	n/a
1	Authenticode	2	10
2	CodeGen	4	100
3	Antimalware	0x108	1 0000 1000
4	Lsa	0x110	1 0001 0000
5	Windows	0x13e	1 0011 1110
6	WinTCB	0x17e	1 0111 1110
7	WinSystem	0x1fe	1 1111 1110
8	SignerApp	0	n/a

RtlProtectedAccess Array

Index	Signer	DominateMask	Bit Explanation
0	none	0	n/a
1	Authenticode	2	10
2	CodeGen	4	100
3	Antimalware	0x108	1 0000 1000
4	Lsa	0x110	1 0001 0000
5	Windows	0x13e	1 0011 1110
6	WinTCB	0x17e	1 0111 1110
7	WinSystem	0x1fe	1 1111 1110
8	SignerApp	0	n/a

1 – Authenticode

2 – CodeGen

4 – Lsa

8 – SignerApp

1-8 – All Signers

RtlProtectedAccess Array

Index	Signer	DominateMask	Bit Explanation
0	none	0	n/a
1	Authenticode	2	10
2	CodeGen	4	100
3	Antimalware	0x108	1 0000 1000
4	Lsa	0x110	8 7654 3210 1 0001 0000
5	Windows	0x13e	1 0011 1110
6	WinTCB	0x17e	1 0111 1110
7	WinSystem	0x1fe	1 1111 1110
8	SignerApp	0	n/a

- 1 – Authenticode
- 2 – CodeGen
- 4 – Lsa
- 8 – SignerApp
- 1-8 – All Signers

RtlProtectedAccess Array

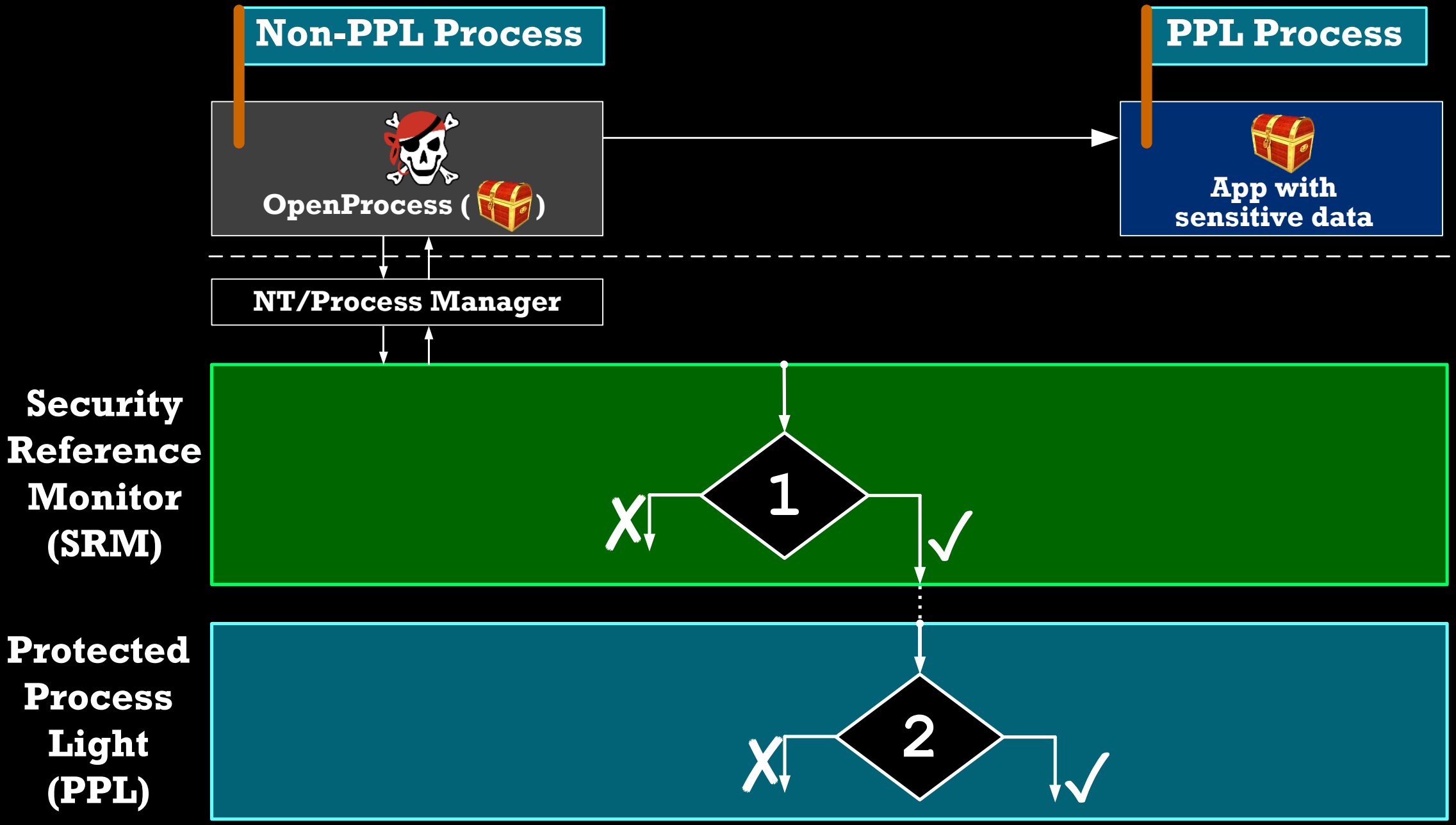
Index	Signer	DominateMask	Bit Explanation
0	none	0	n/a
1	Authenticode	2	10
2	CodeGen	4	100
3	Antimalware	0x108	1 0000 1000
4	Lsa	0x110	1 0001 0000
5	Windows	0x13e	1 0011 1110
6	WinTCB	0x17e	1 0111 1110
7	WinSystem	0x1fe	8 7654 3210 1 1111 1110
8	SignerApp	0	n/a

- 1 – Authenticode
- 2 – CodeGen
- 4 – Lsa
- 8 – SignerApp
- 1-8 – All Signers

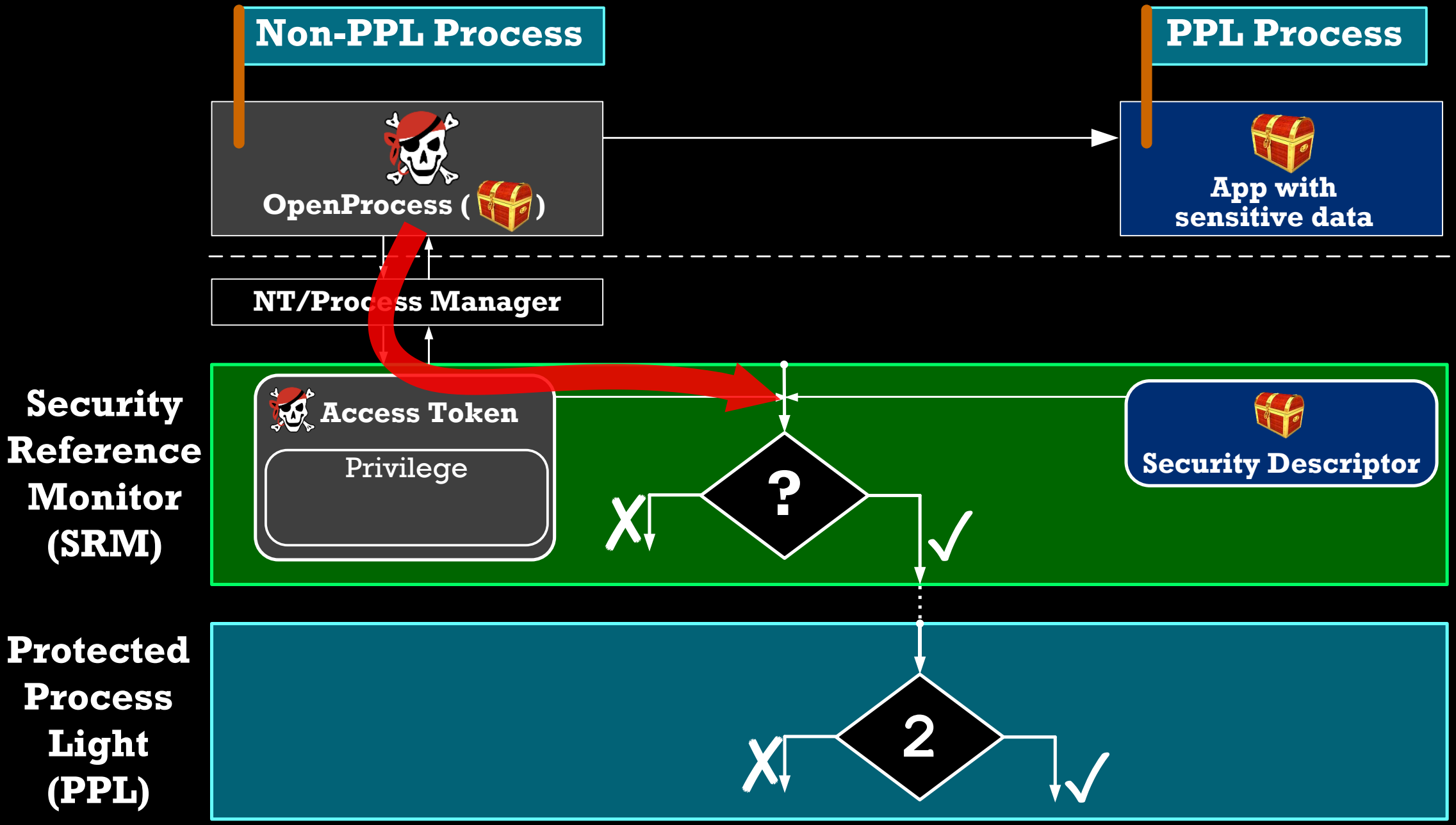
Episode 6

Two Windows security features SRM and PPL are playing together and losing

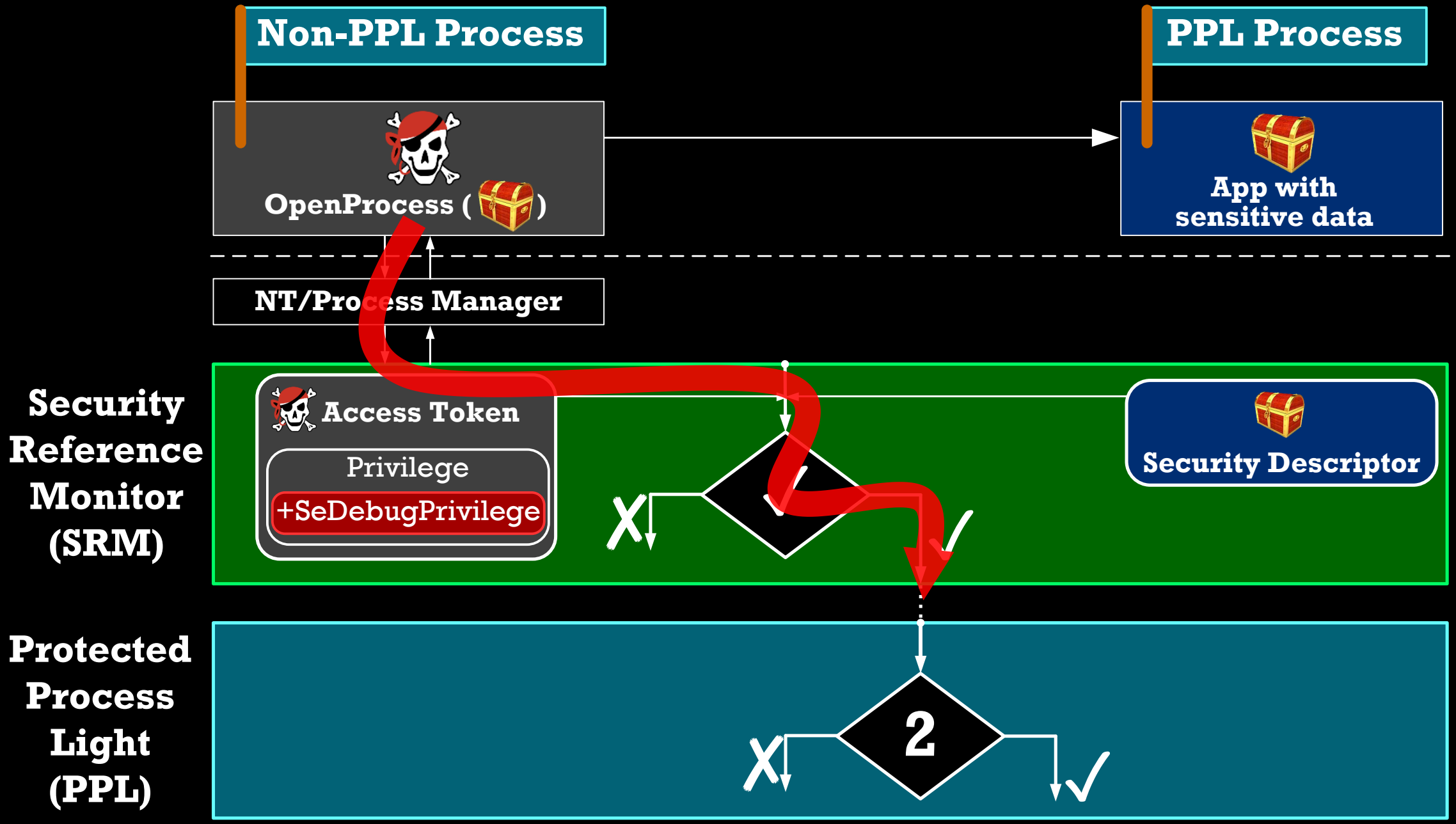
OpenProcess performs two checks: SRM and then PPL



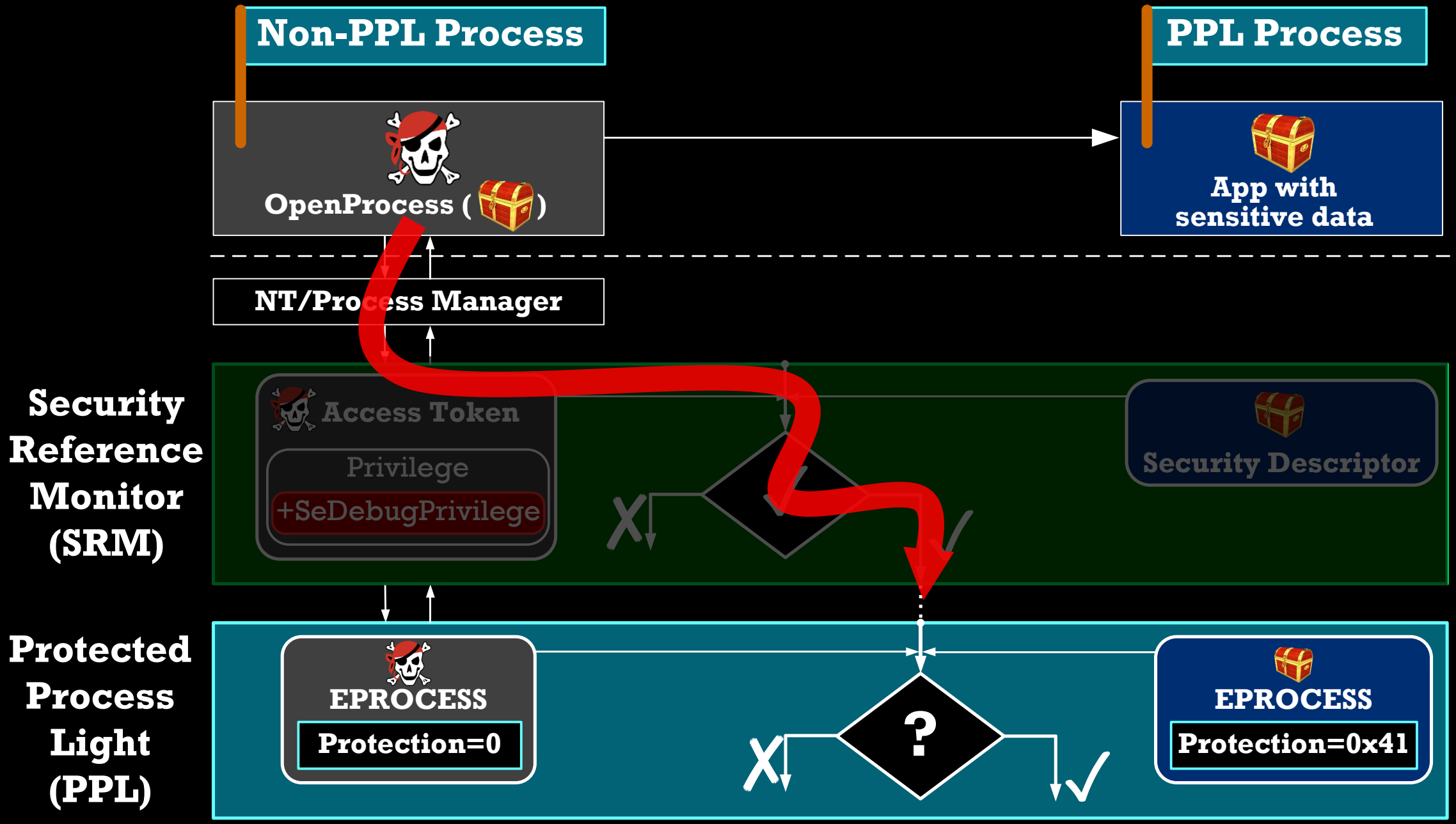
OpenProcess performs two checks: SRM and then PPL



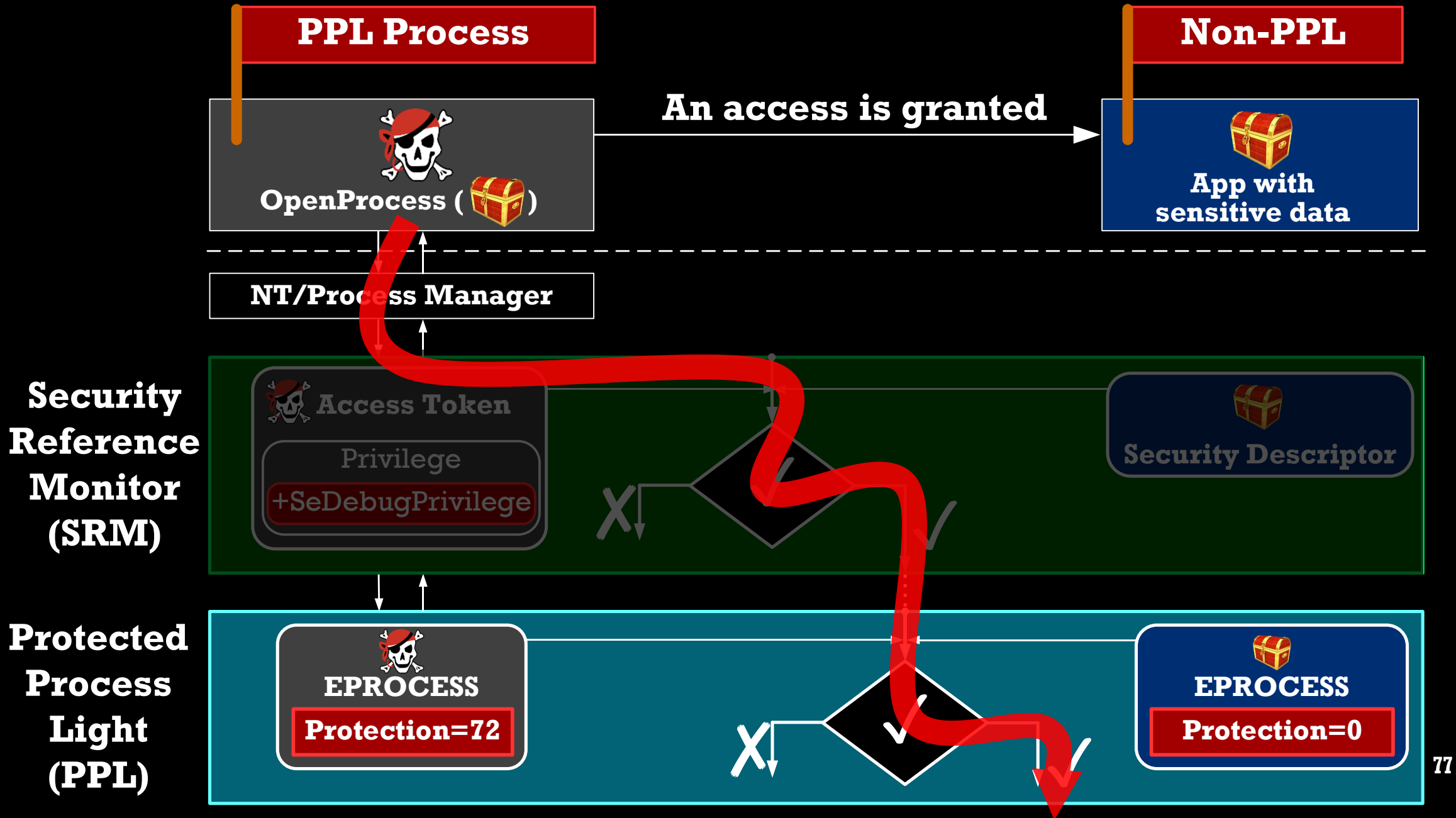
OpenProcess performs two checks: SRM and then PPL



OpenProcess performs two checks: SRM and then PPL



OpenProcess performs two checks: SRM and then PPL



Episode 7

Attacks on PPL?

PROTECTION LEVEL CAN BE ILLEGALLY CHANGED BY DRIVER

Non PPL Process

OpenProcess ()

PPL Process

App with sensitive data

Access Token

Security Descriptor

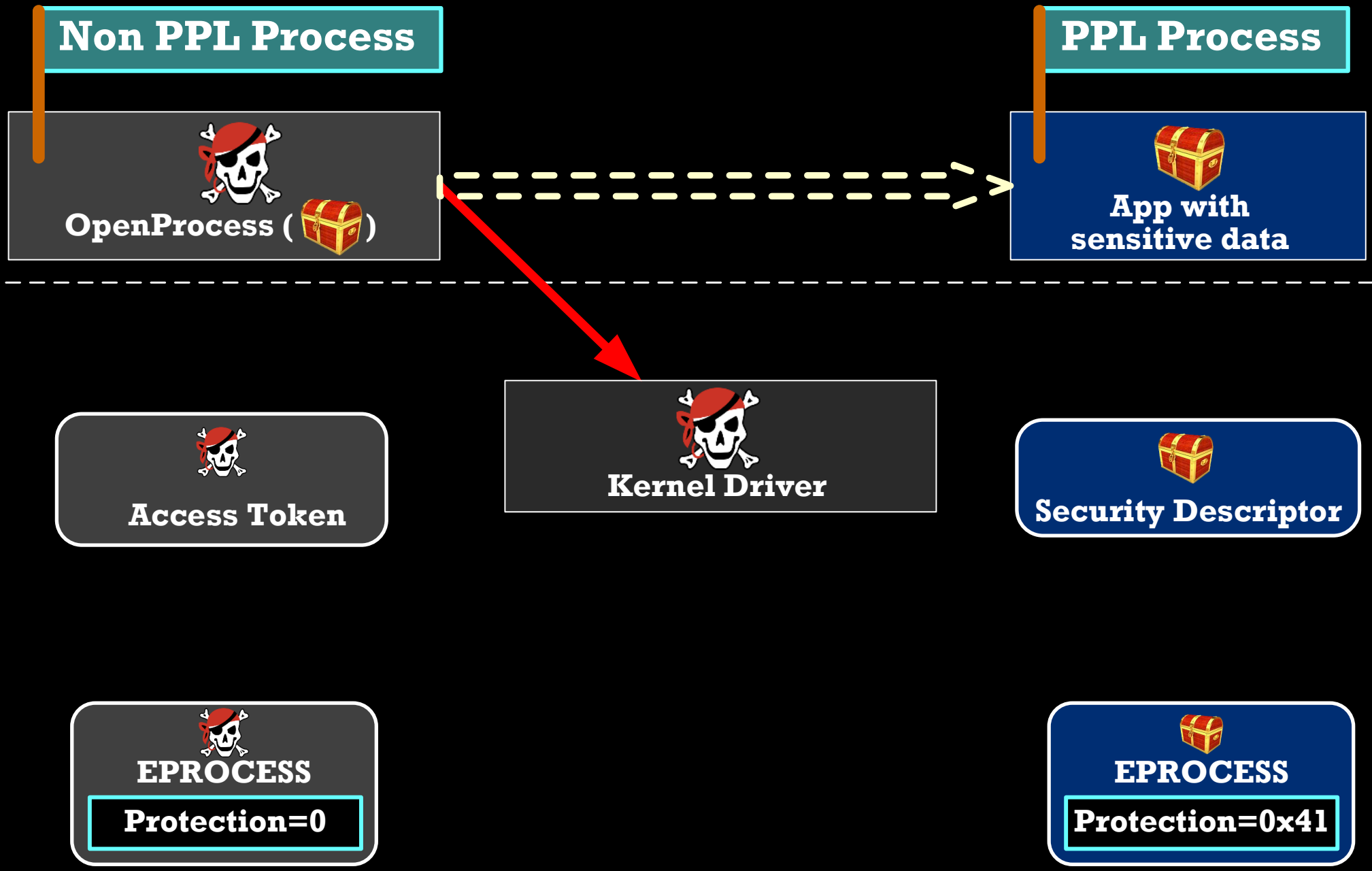
EPROCESS

Protection=0

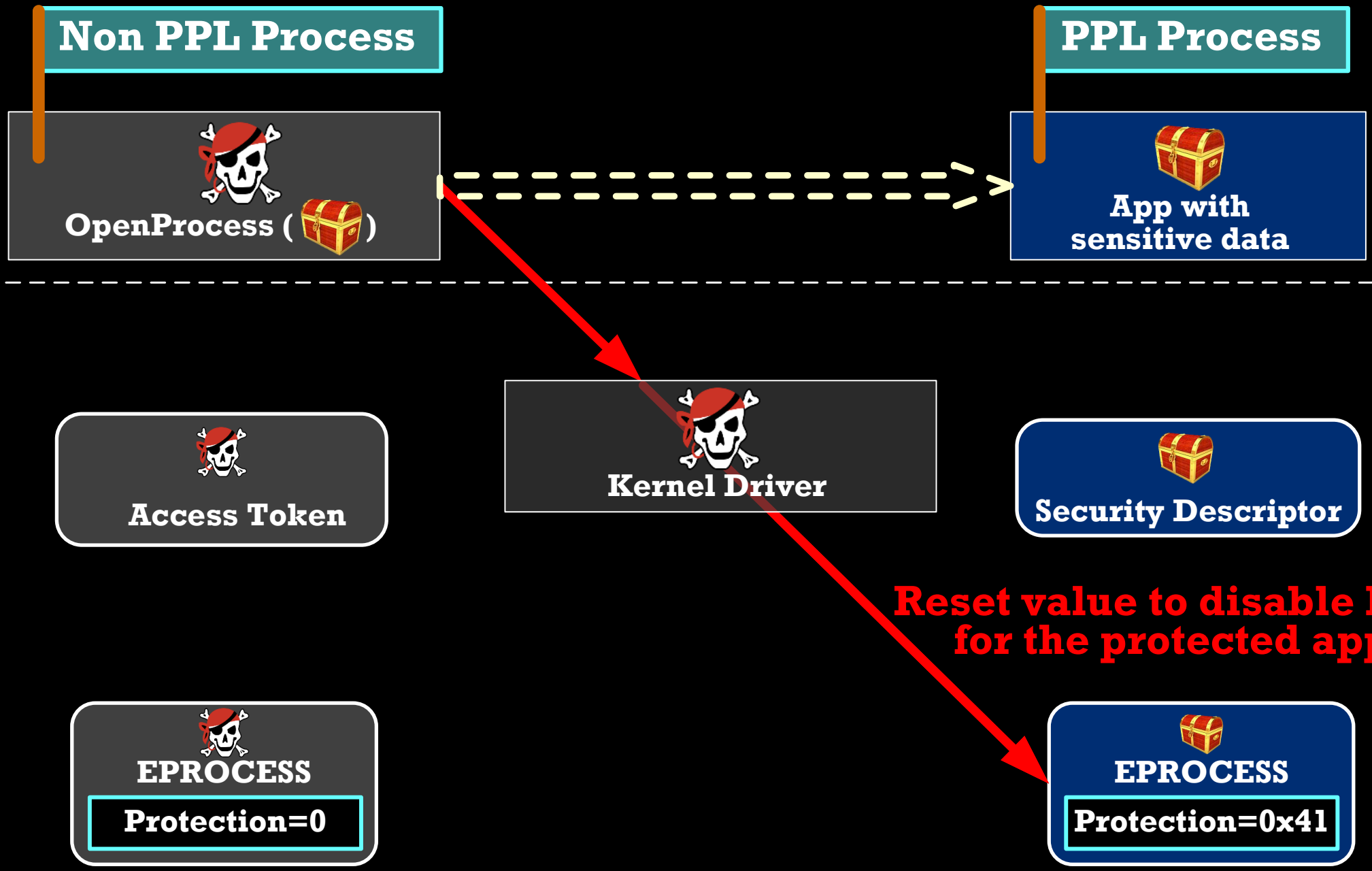
EPROCESS

Protection=0x41

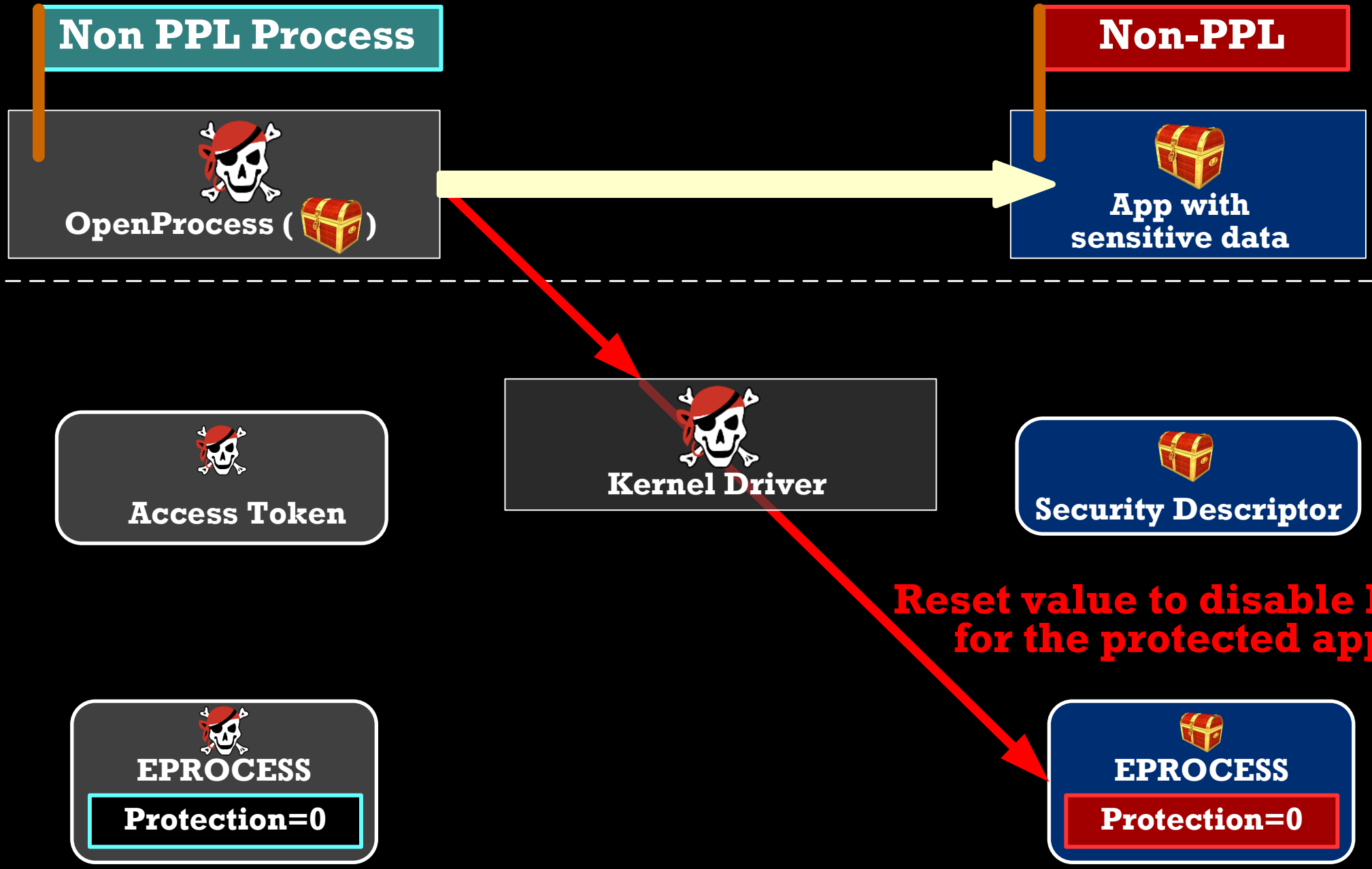
PROTECTION LEVEL CAN BE ILLEGALLY CHANGED BY DRIVER



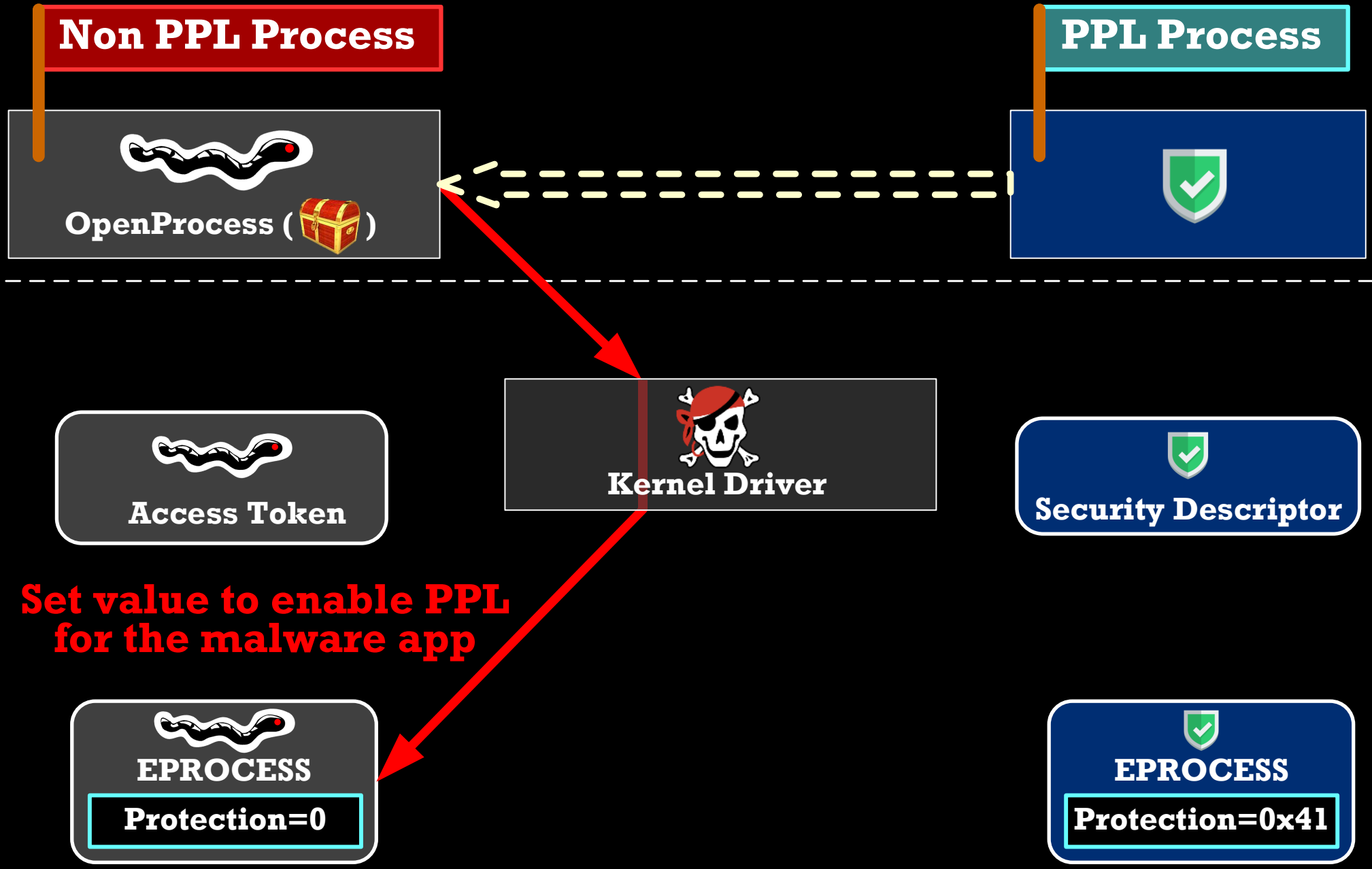
PROTECTION LEVEL CAN BE ILLEGALLY CHANGED BY DRIVER



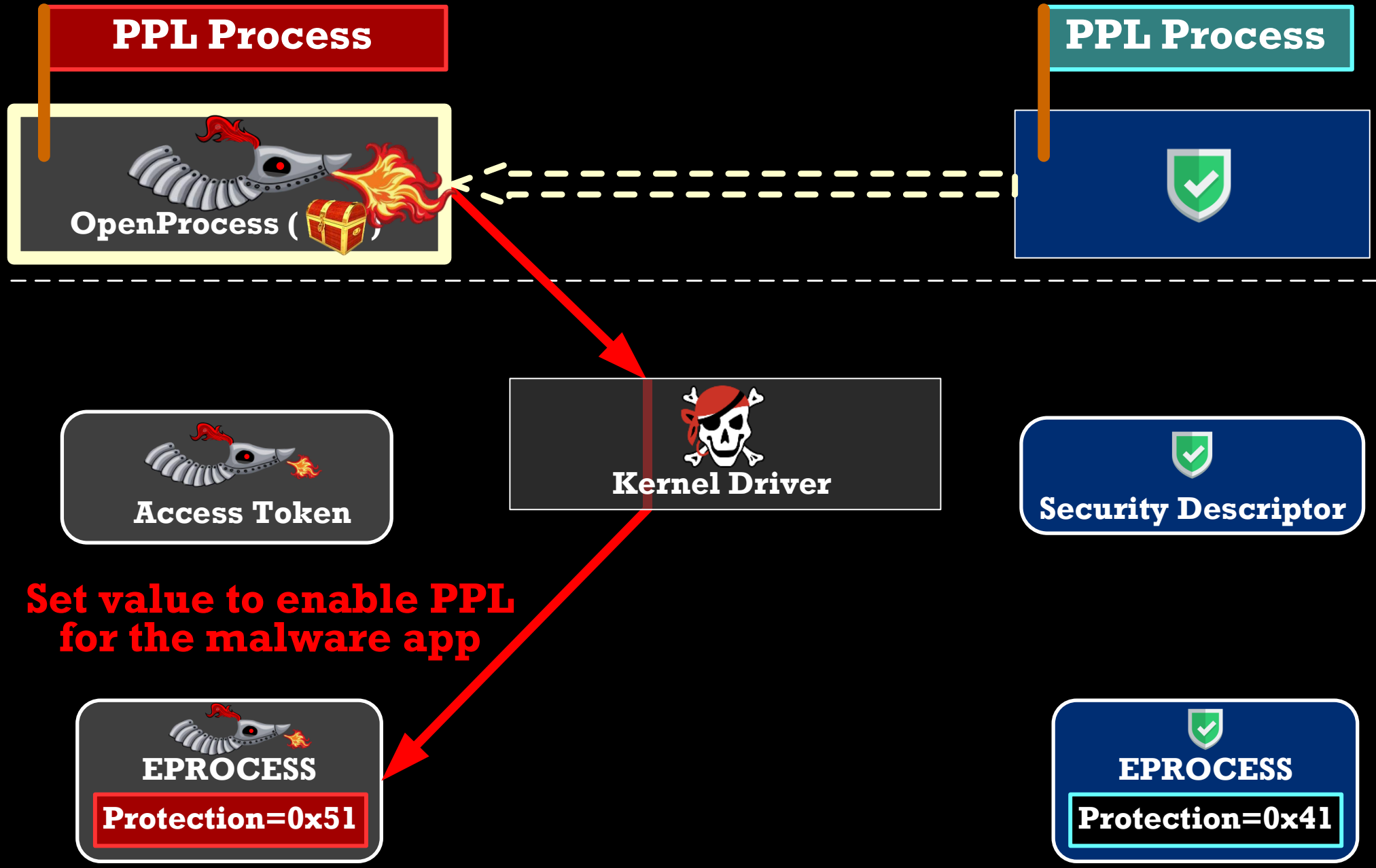
PROTECTION LEVEL CAN BE ILLEGALLY CHANGED BY DRIVER



PROTECTION LEVEL CAN BE ILLEGALLY CHANGED BY DRIVER

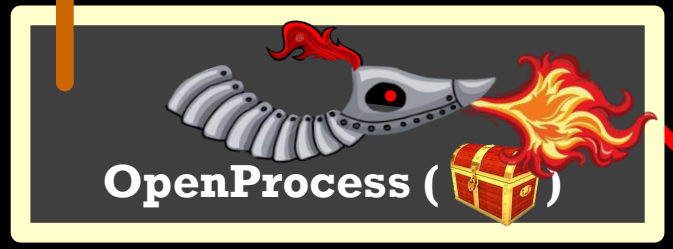


PROTECTION LEVEL CAN BE ILLEGALLY CHANGED BY DRIVER



PROTECTION LEVEL CAN BE ILLEGALLY CHANGED BY DRIVER

PPL Process

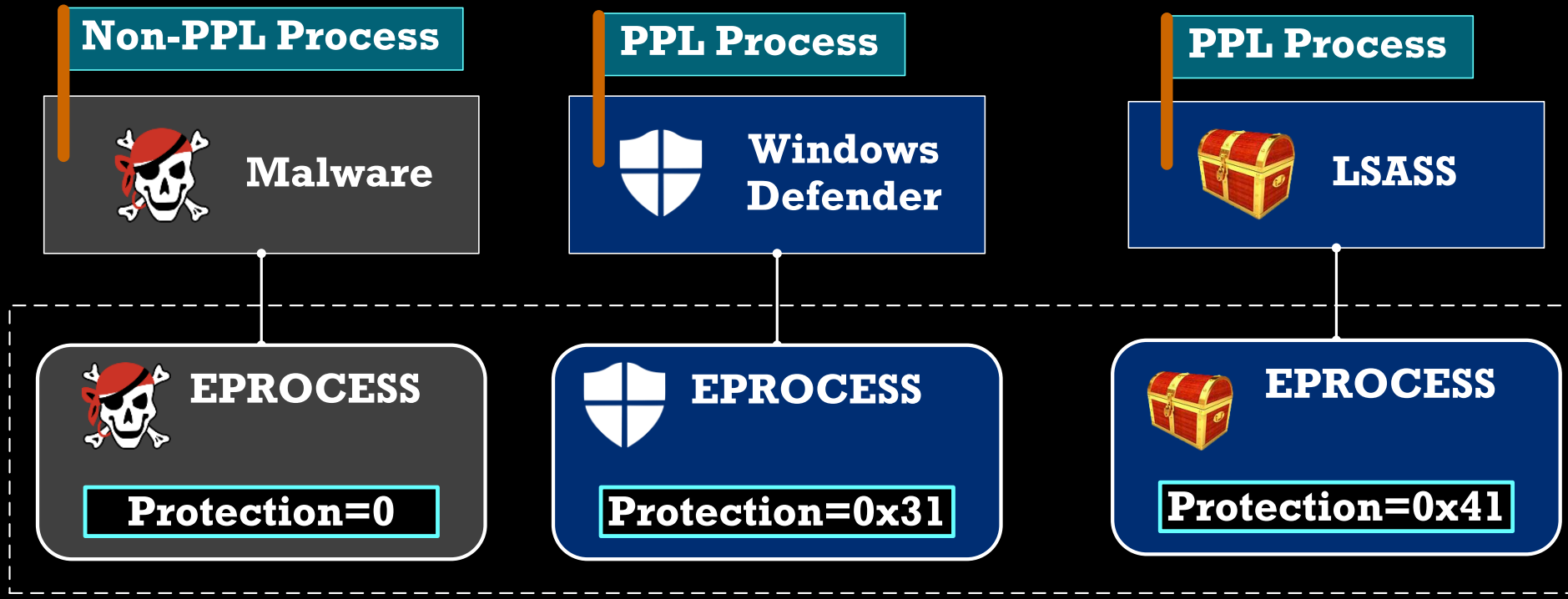


Non-PPL

Set value to enable PPL for the malware app

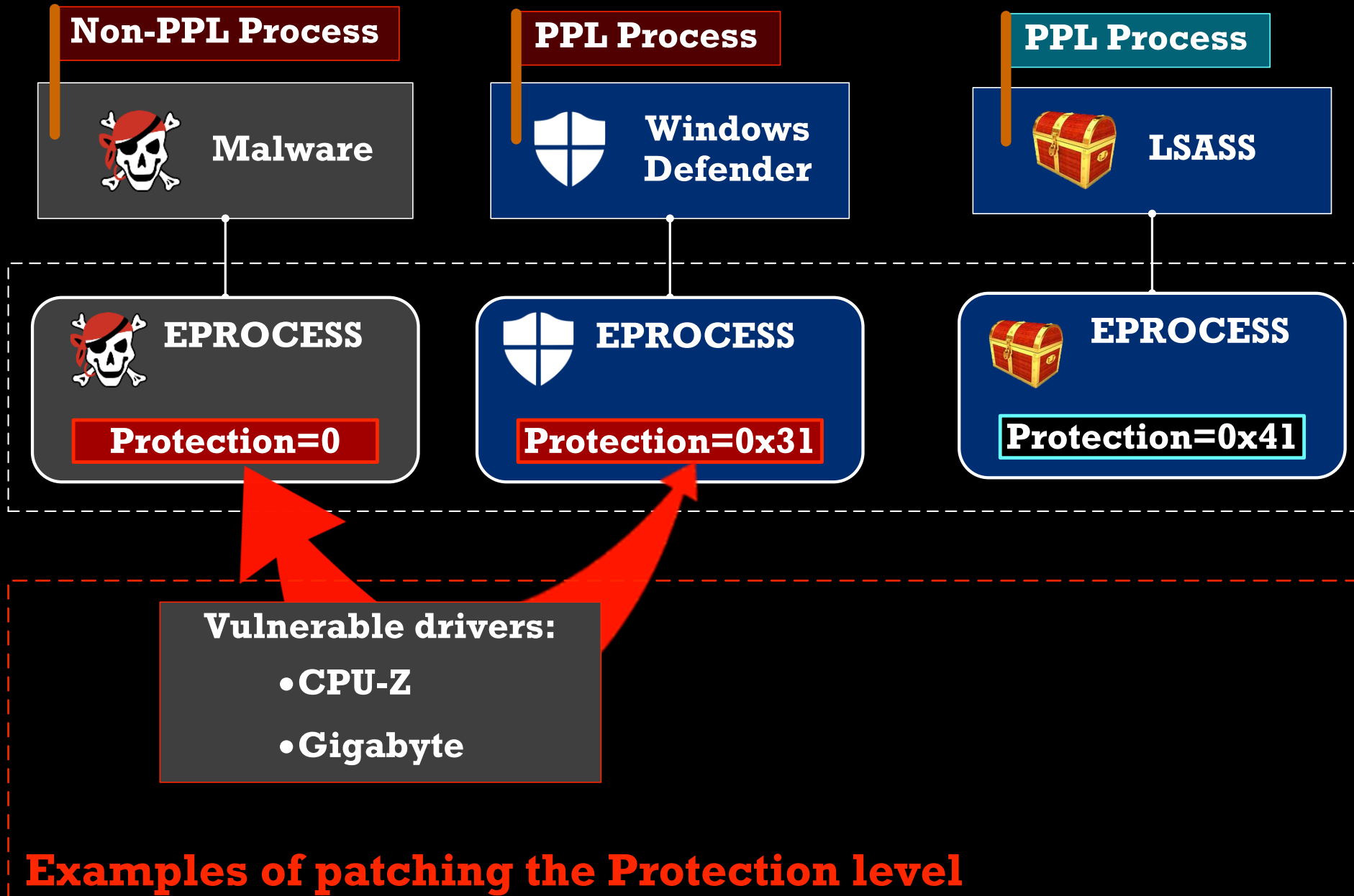
Reset value to disable PPL for the protected app

Attacks on PPL: Drivers Can Modify Protection Byte

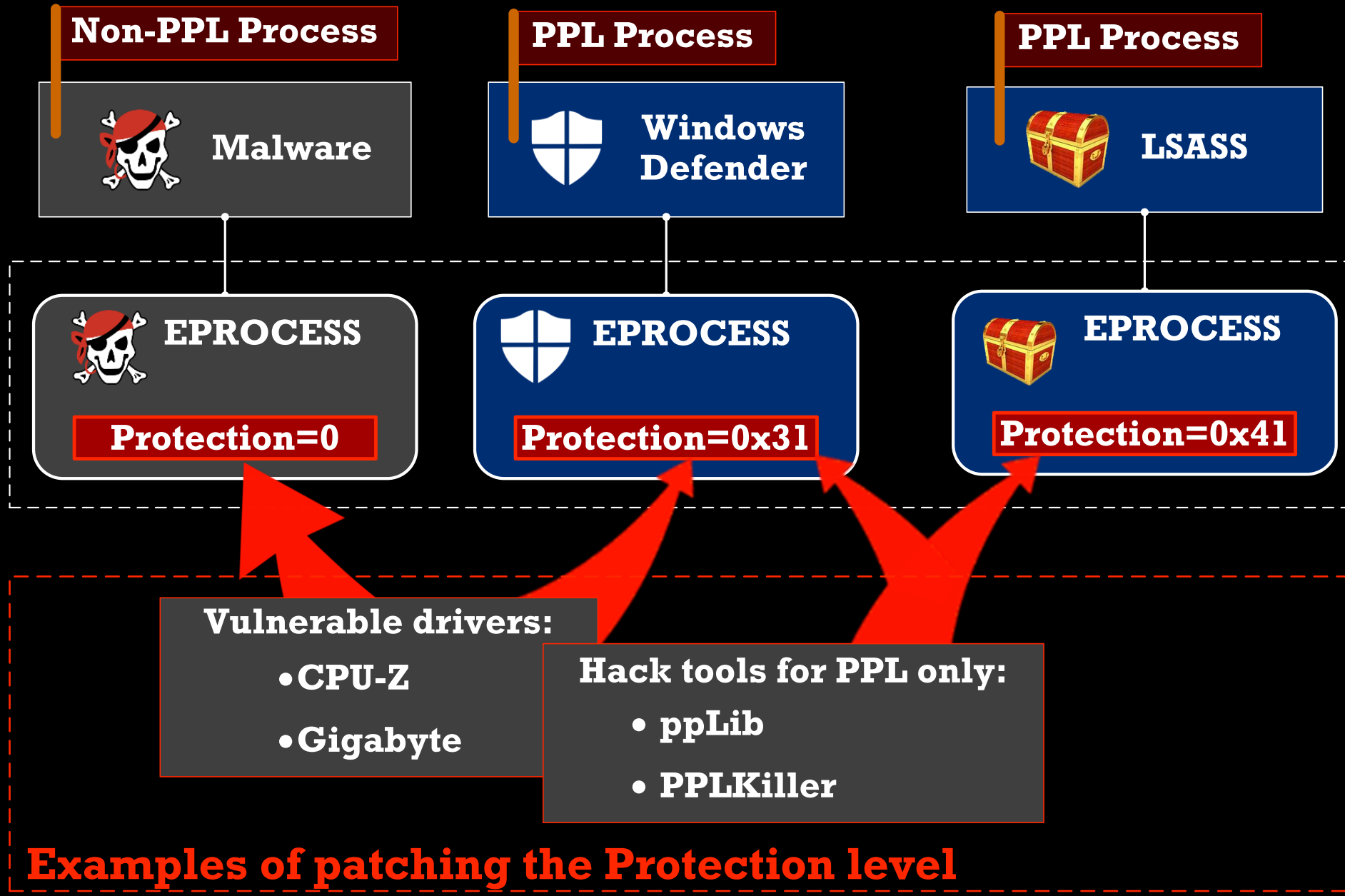


Examples of patching the Protection level

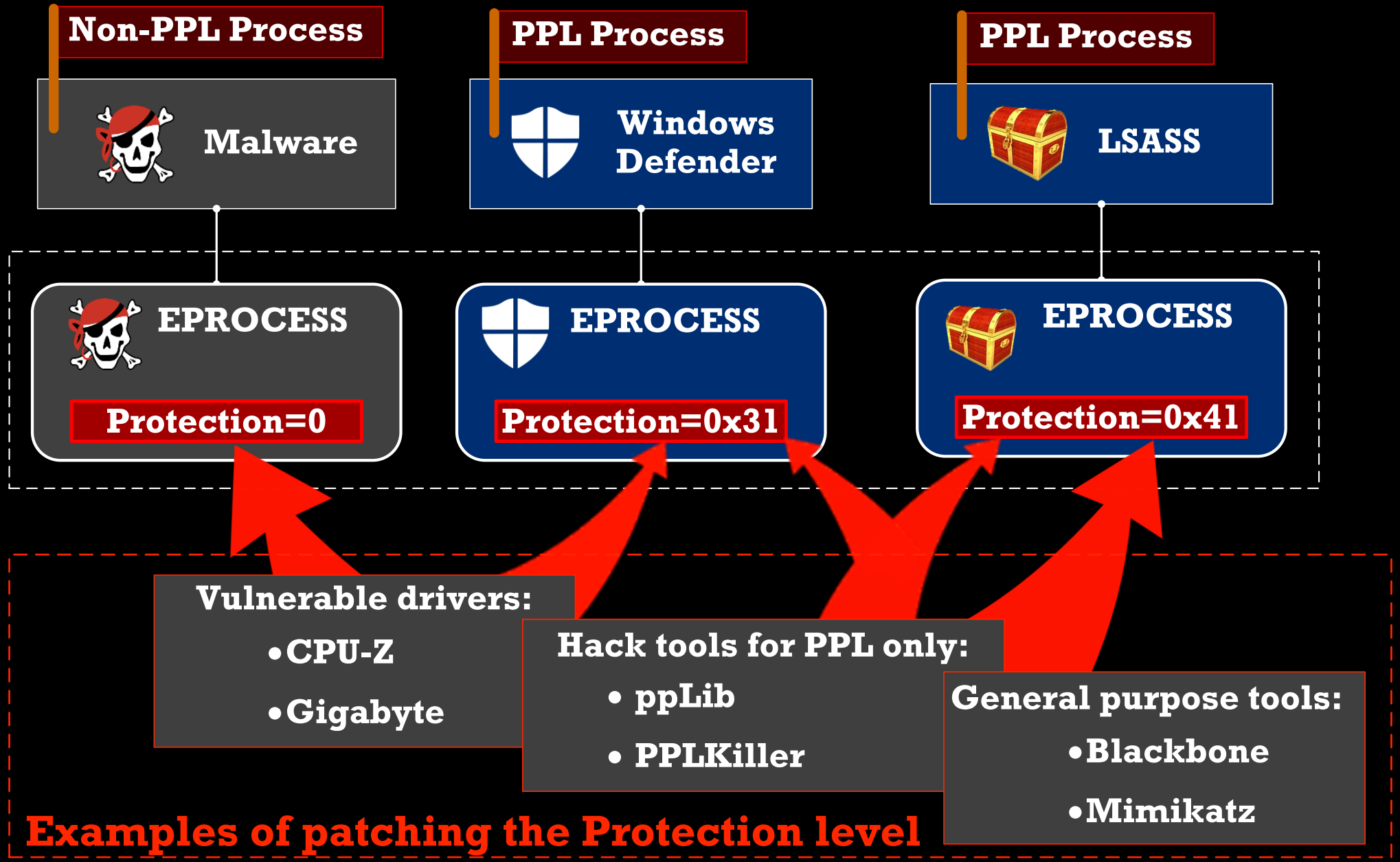
Attacks on PPL: Drivers Can Modify Protection Byte



Attacks on PPL: Drivers Can Modify Protection Byte



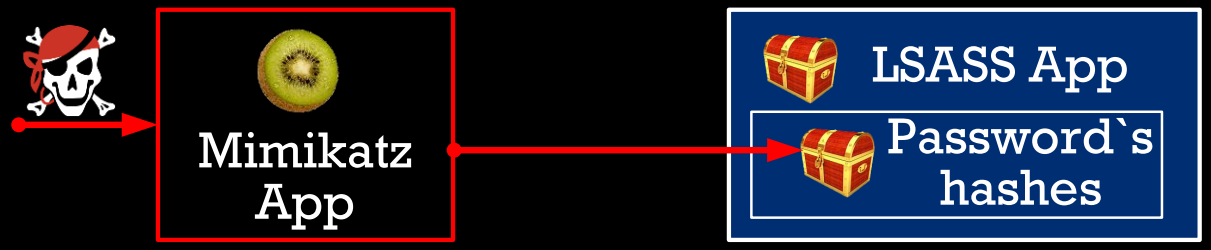
Attacks on PPL: Drivers Can Modify Protection Byte



Episode 8

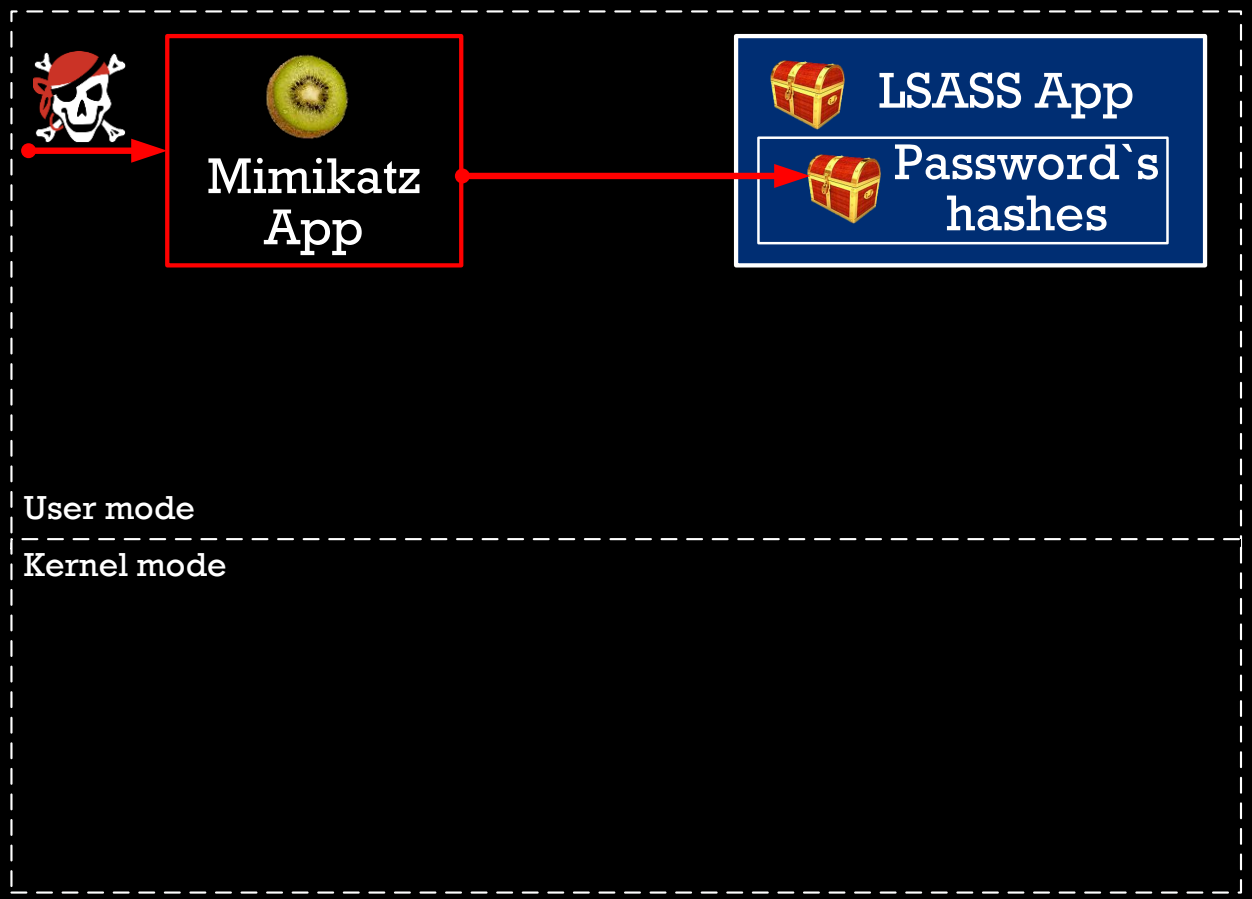
Mimikatz can disable PPL!

Mimikatz can gather credentials from Windows



Mimikatz can gather credentials from Windows

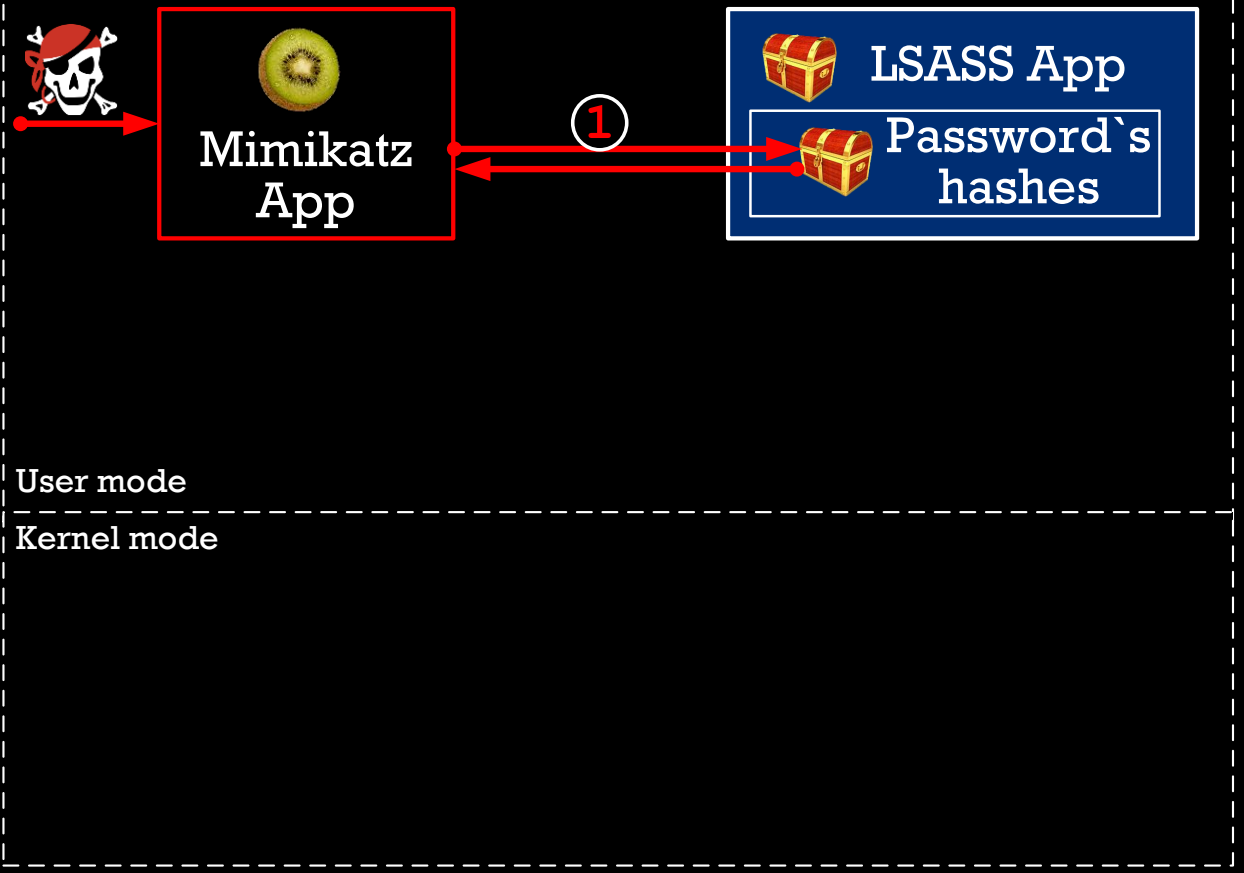
Attempt 1: Without Disabling PPL



Attempt 2: With Disabling PPL for LSASS

Mimikatz can gather credentials from Windows

Attempt 1: Without Disabling PPL

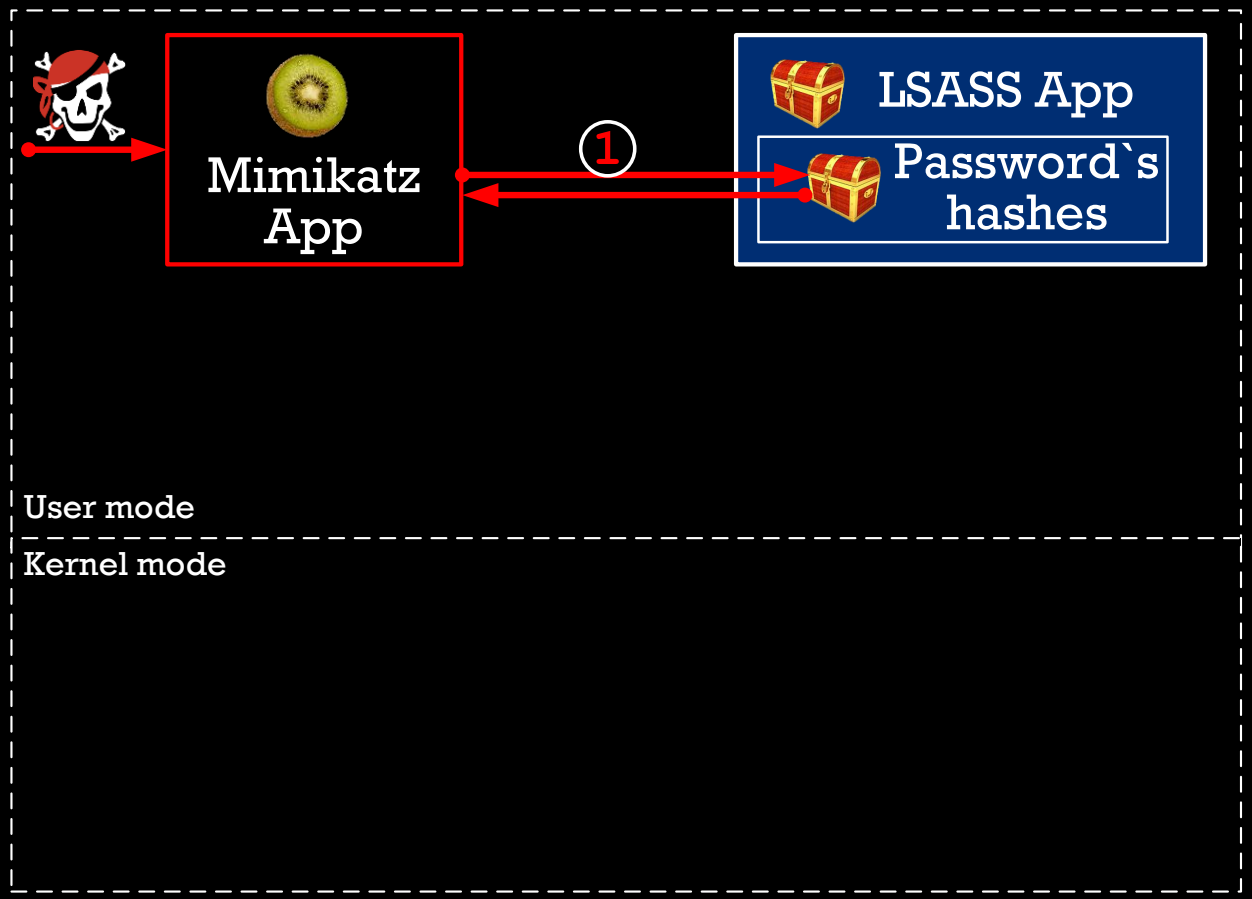


Attempt 2: With Disabling PPL for LSASS

1
- Add debug privilege
- Dump user's hash

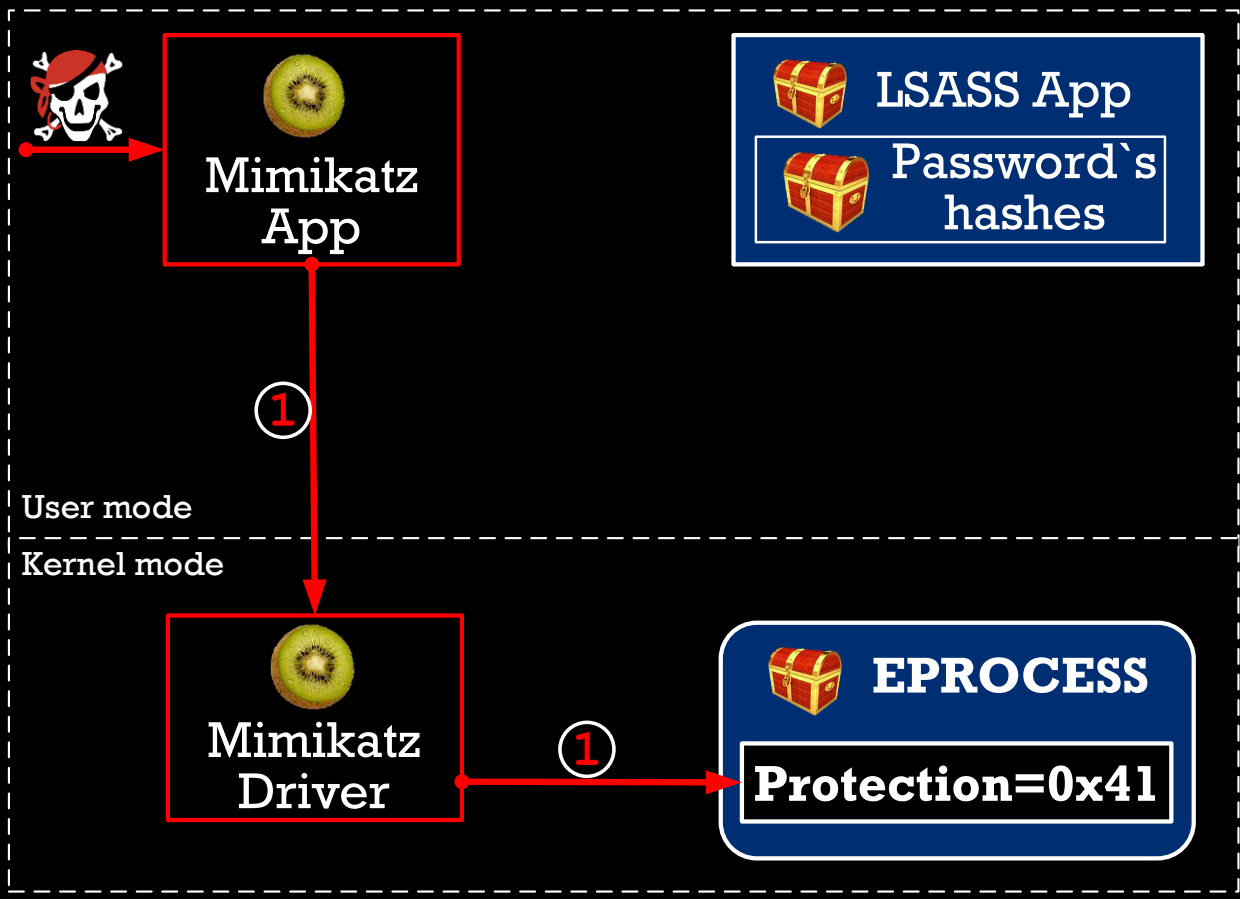
Mimikatz can gather credentials from Windows

Attempt 1: Without Disabling PPL



1
- Add debug privilege
- Dump user's hash

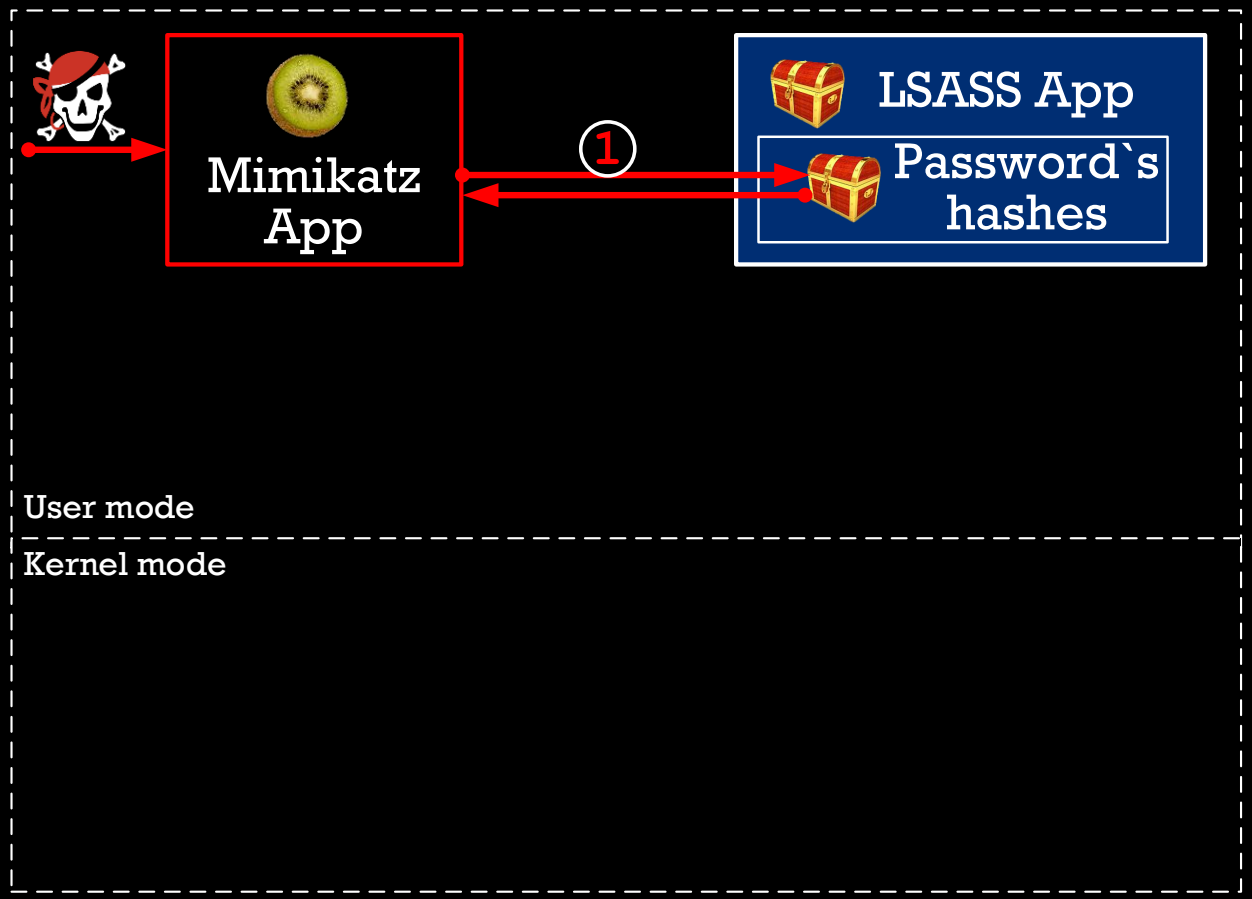
Attempt 2: With Disabling PPL for LSASS



1
- Load Mimikatz driver
- Clear Protection byte

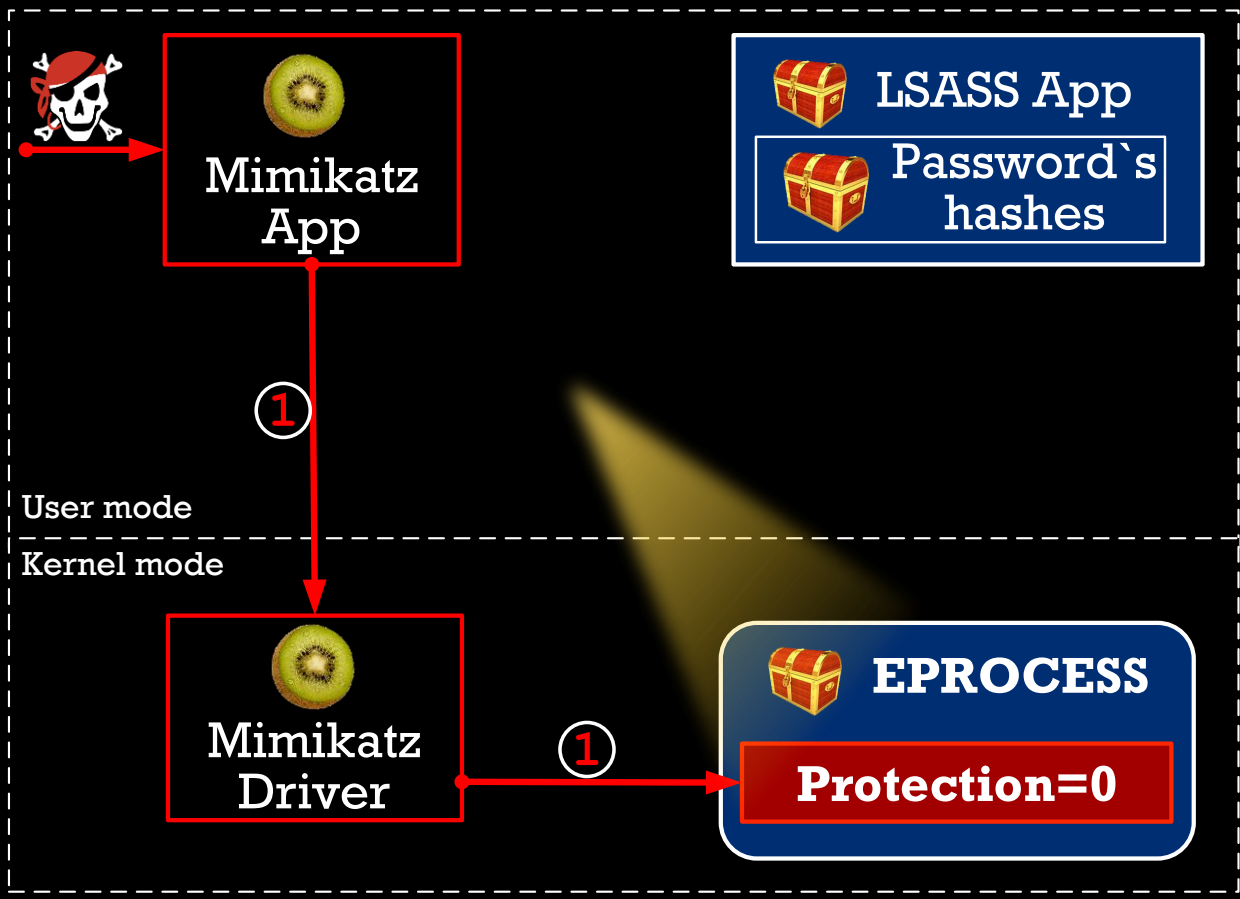
Mimikatz can gather credentials from Windows

Attempt 1: Without Disabling PPL



1
- Add debug privilege
- Dump user's hash

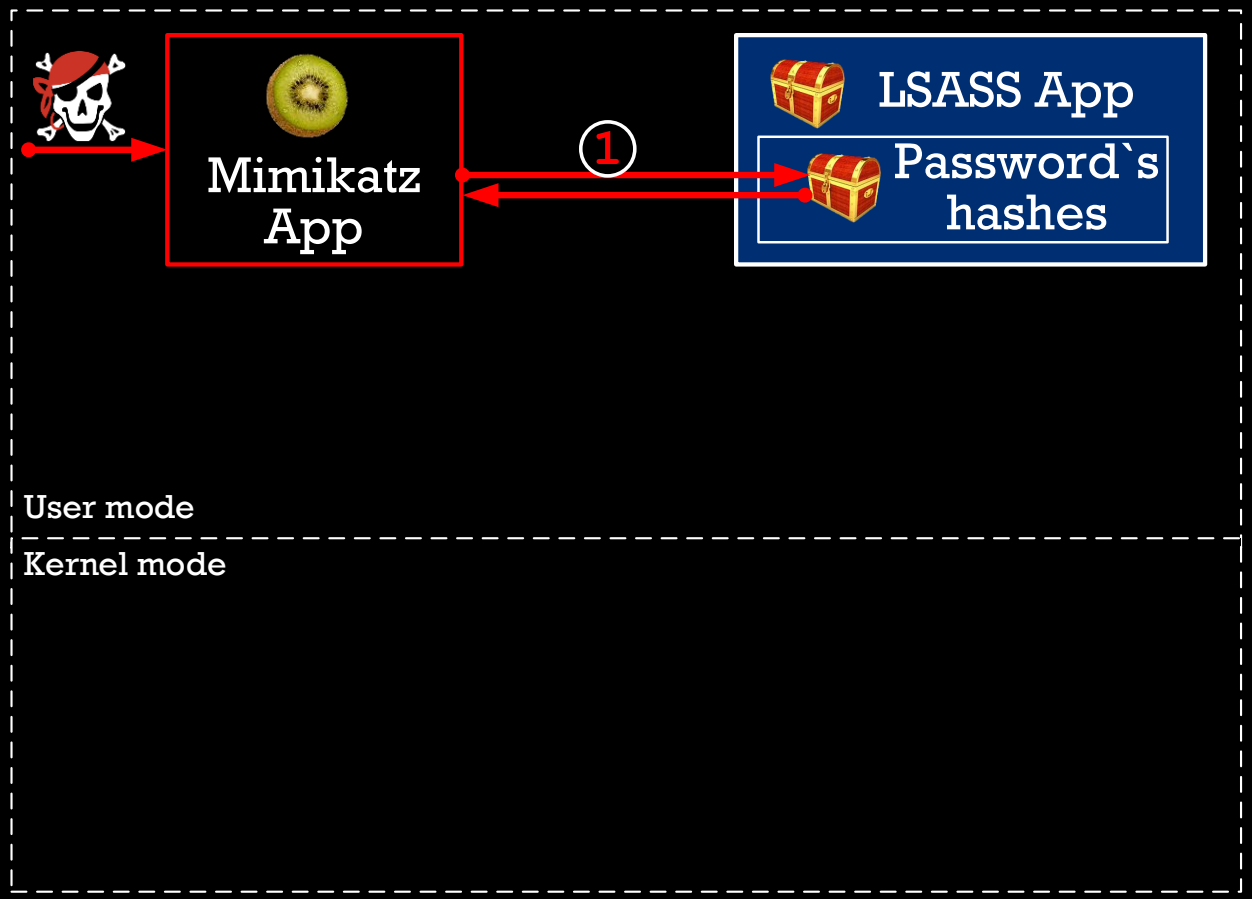
Attempt 2: With Disabling PPL for LSASS



1
- Load Mimikatz driver
- Clear Protection byte

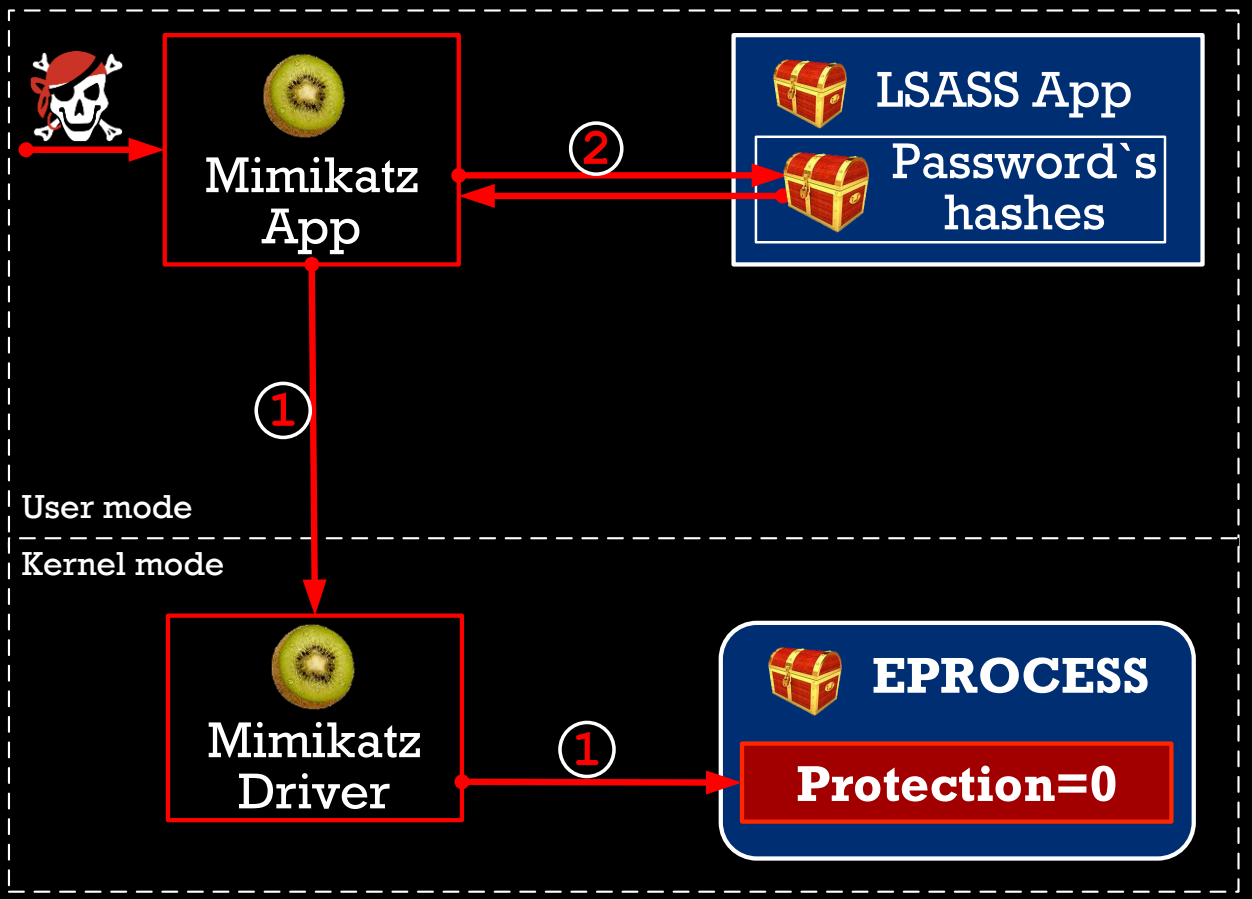
Mimikatz can gather credentials from Windows

Attempt 1: Without Disabling PPL



1
- Add debug privilege
- Dump user's hash

Attempt 2: With Disabling PPL for LSASS

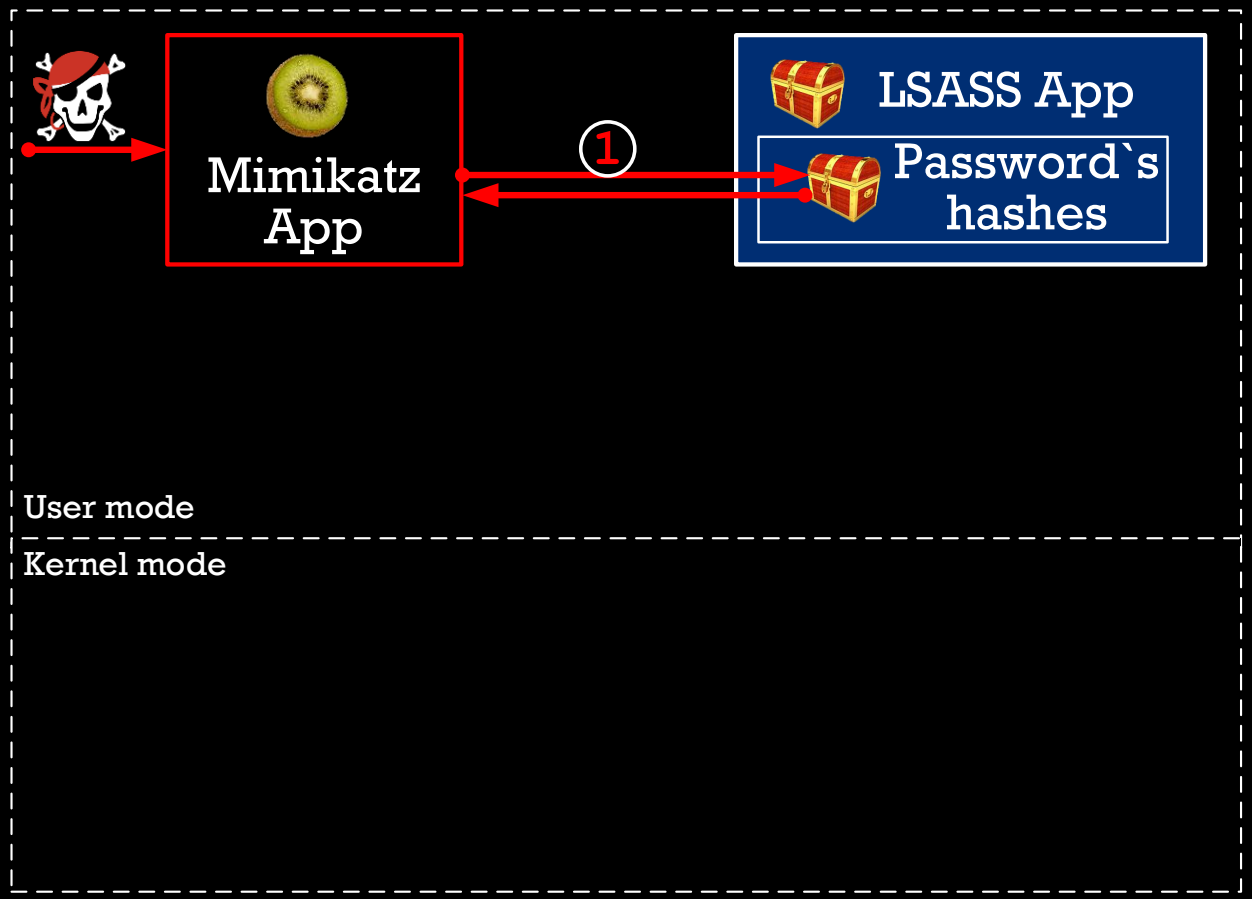


1
- Load Mimikatz driver
- Clear Protection byte

2
- Add debug privilege
- Dump user's hash

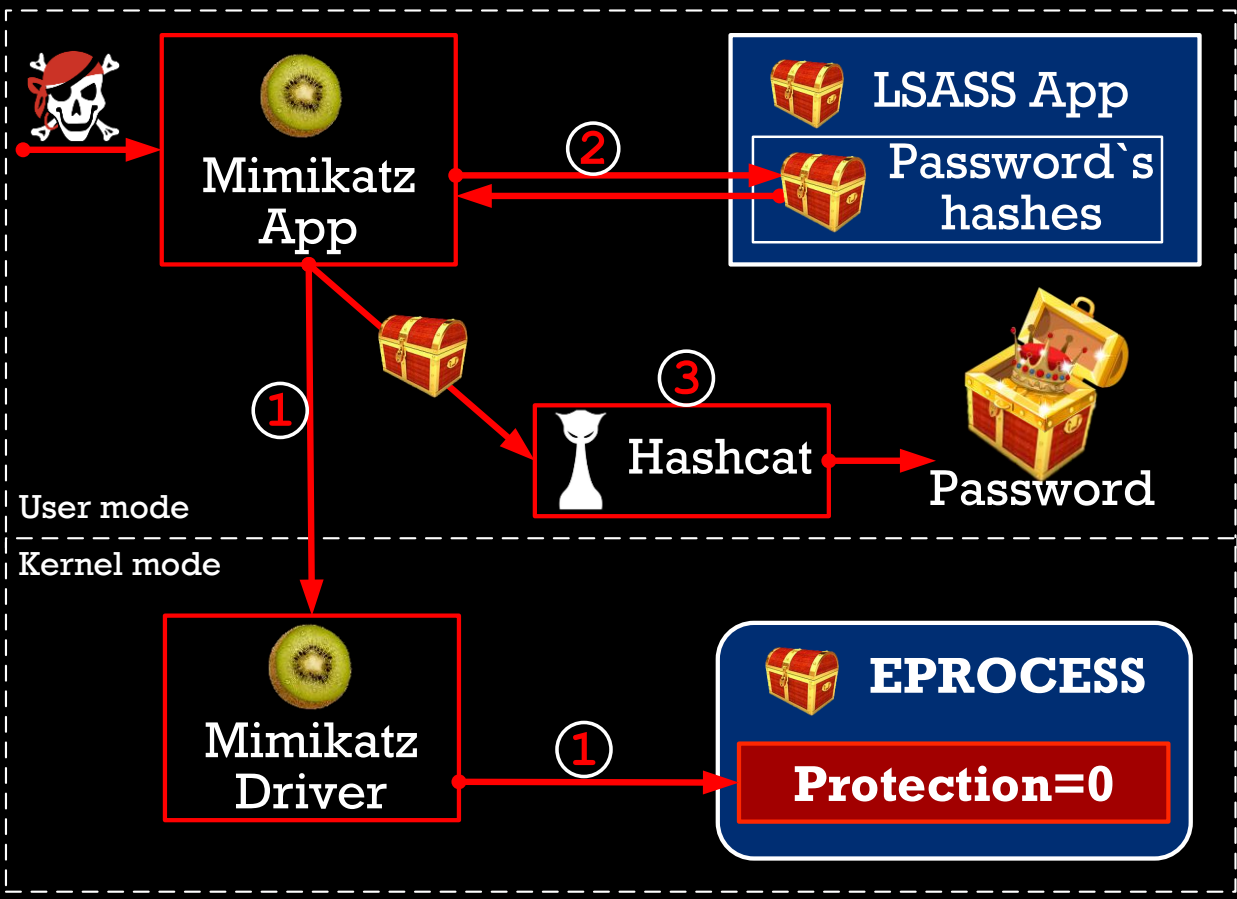
Mimikatz can gather credentials from Windows

Attempt 1: Without Disabling PPL



1
- Add debug privilege
- Dump user's hash

Attempt 2: With Disabling PPL for LSASS

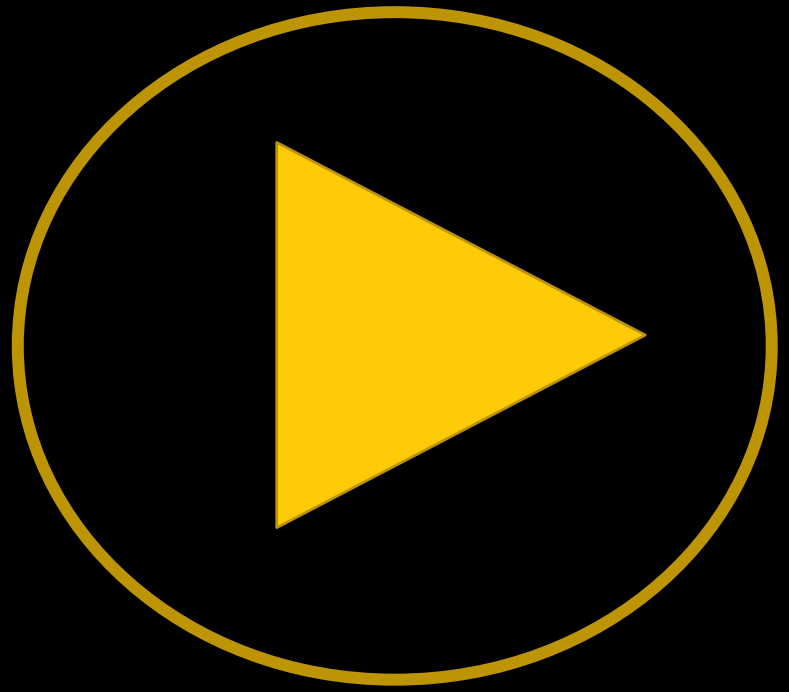


1
- Load Mimikatz driver
- Clear Protection byte

2
- Add debug privilege
- Dump user's hash

3
- Crack the hash to get password

Mimikatz disables PPL to dump NTLM hashes



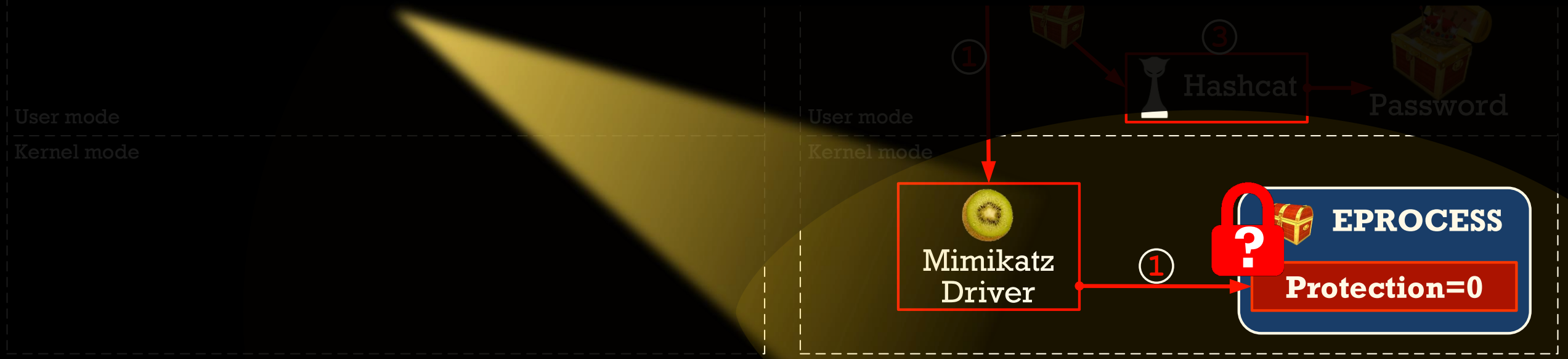
The online version is here –
https://www.youtube.com/embed/66g4PgtuD7c?vq=hd1440

Attempt 1: Without Disabling PPL

Attempt 2: With Disabling PPL for LSASS

– How to prevent PPL disabling?

– We need to restrict access to the Protection field!

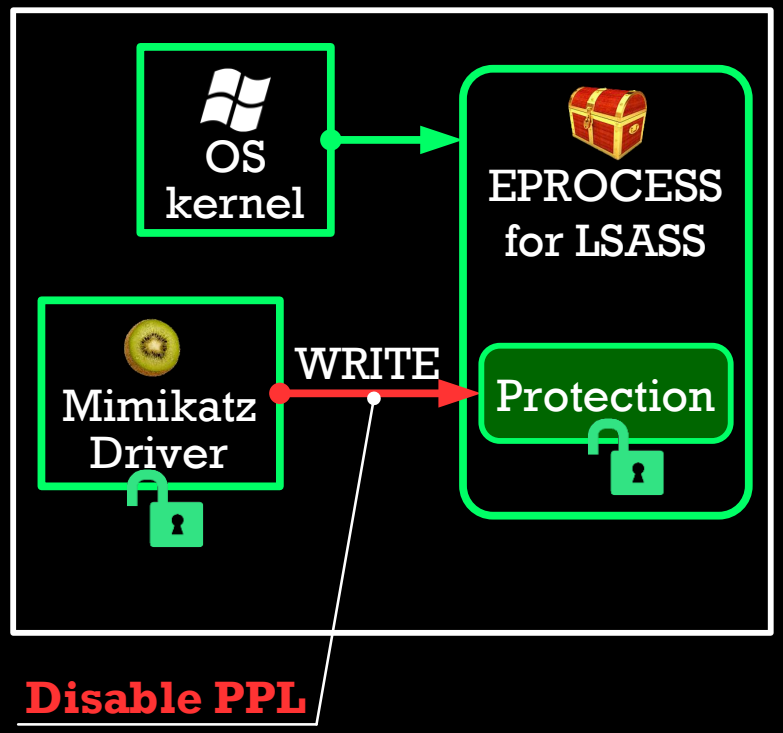


Episode 9

MemoryRanger can block Mimikatz

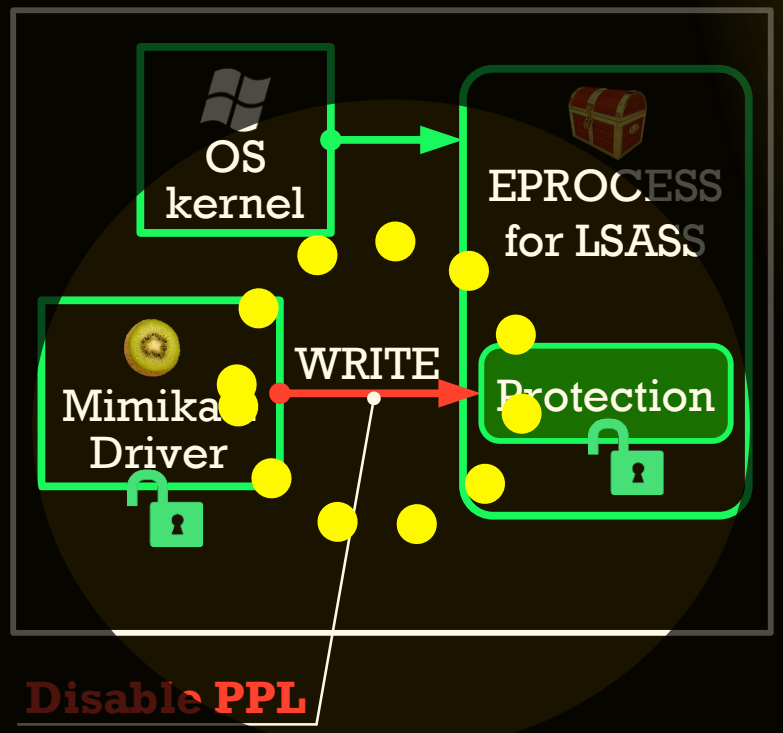
KERNEL DRIVERS SHARE THE SAME MEMORY SPACE

The Current situation

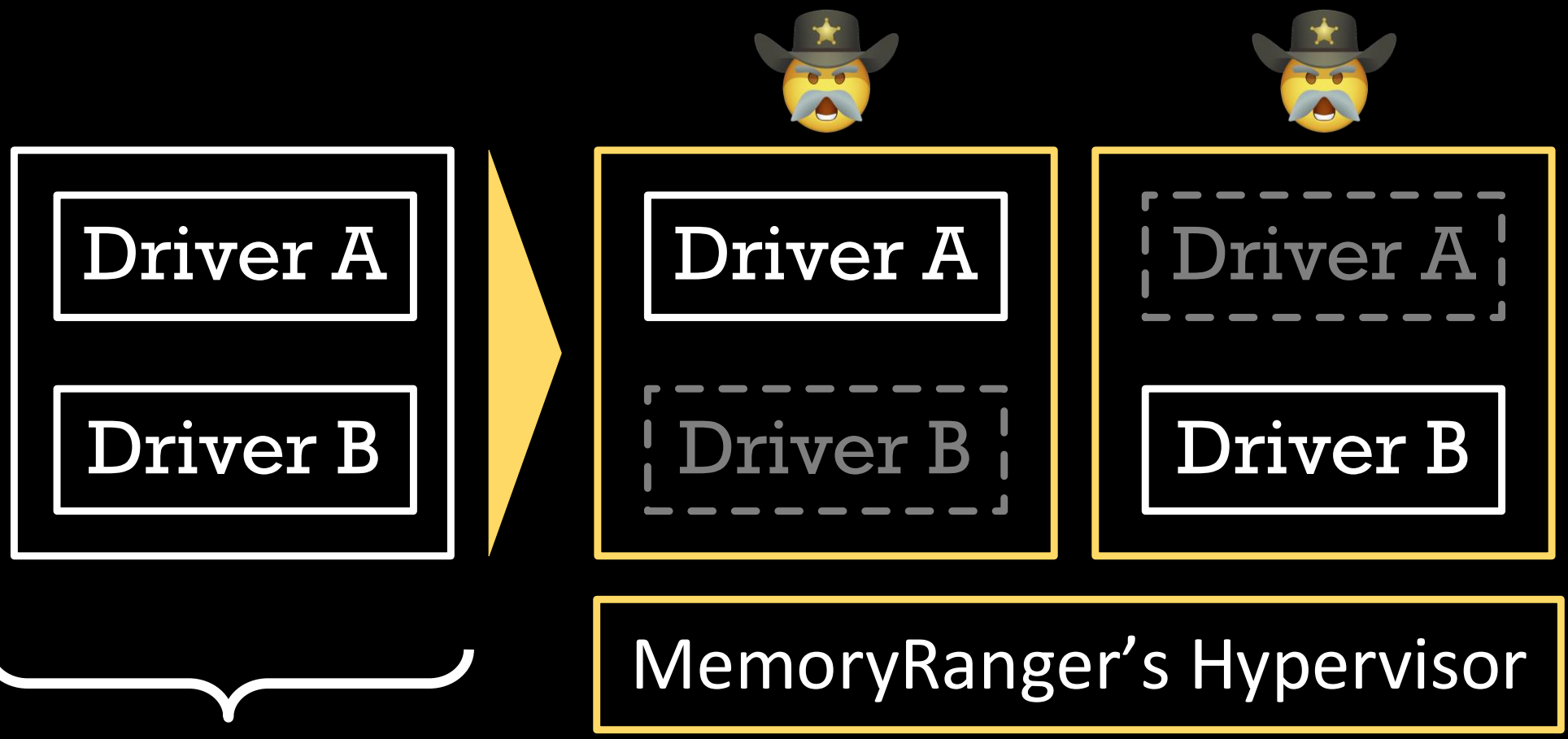


How to restrict access to the Protection field?

The Current situation



MemoryRanger

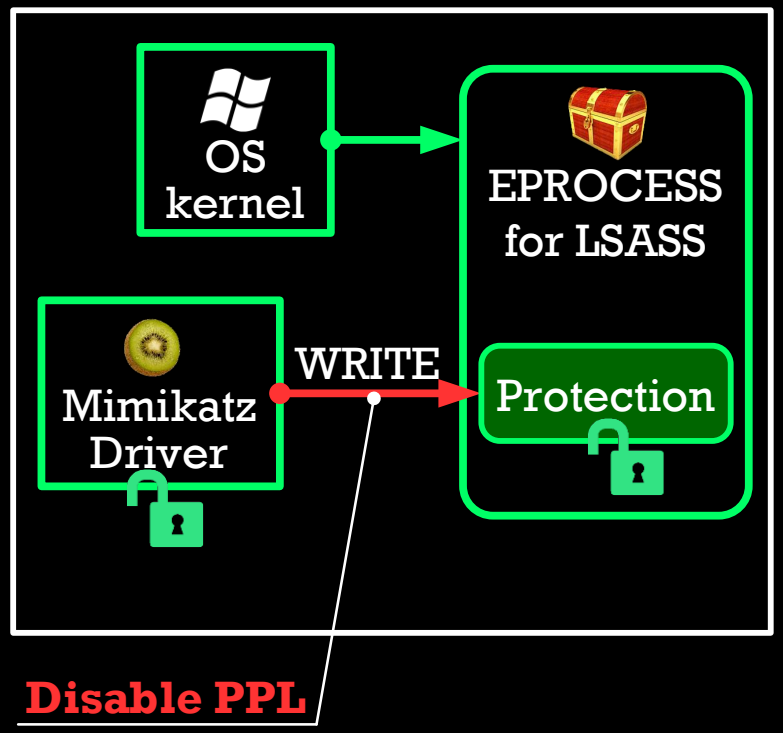


Without MemoryRanger all drivers share the same kernel memory space

MemoryRanger isolates drivers by running them in separate kernel enclaves

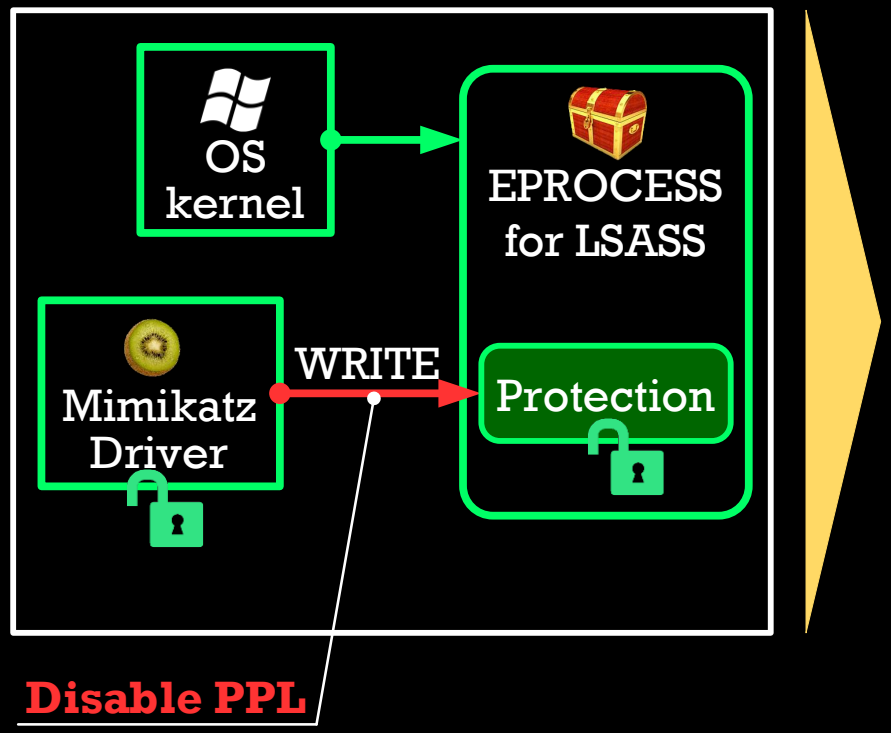
MemoryRanger Prevents Disabling of PPL

The Current situation



MemoryRanger Prevents Disabling of PPL

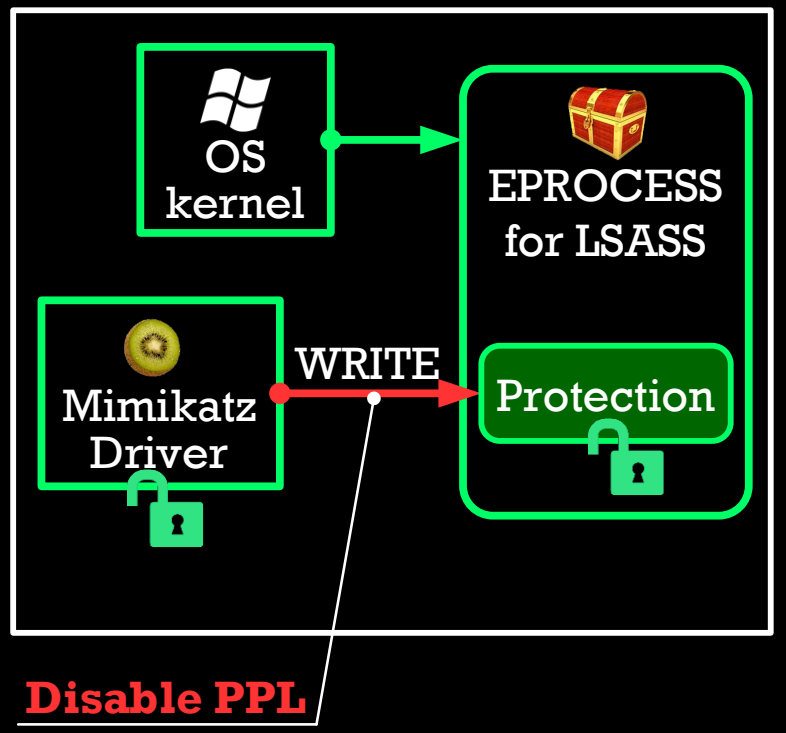
The Current situation



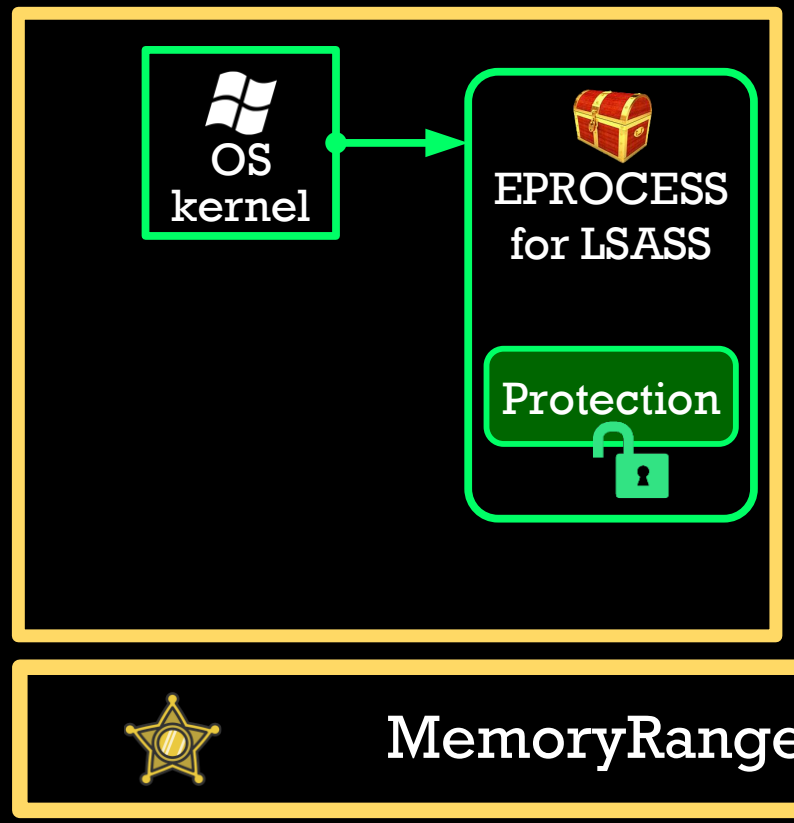
MemoryRanger's Hypervisor

MemoryRanger Prevents Disabling of PPL

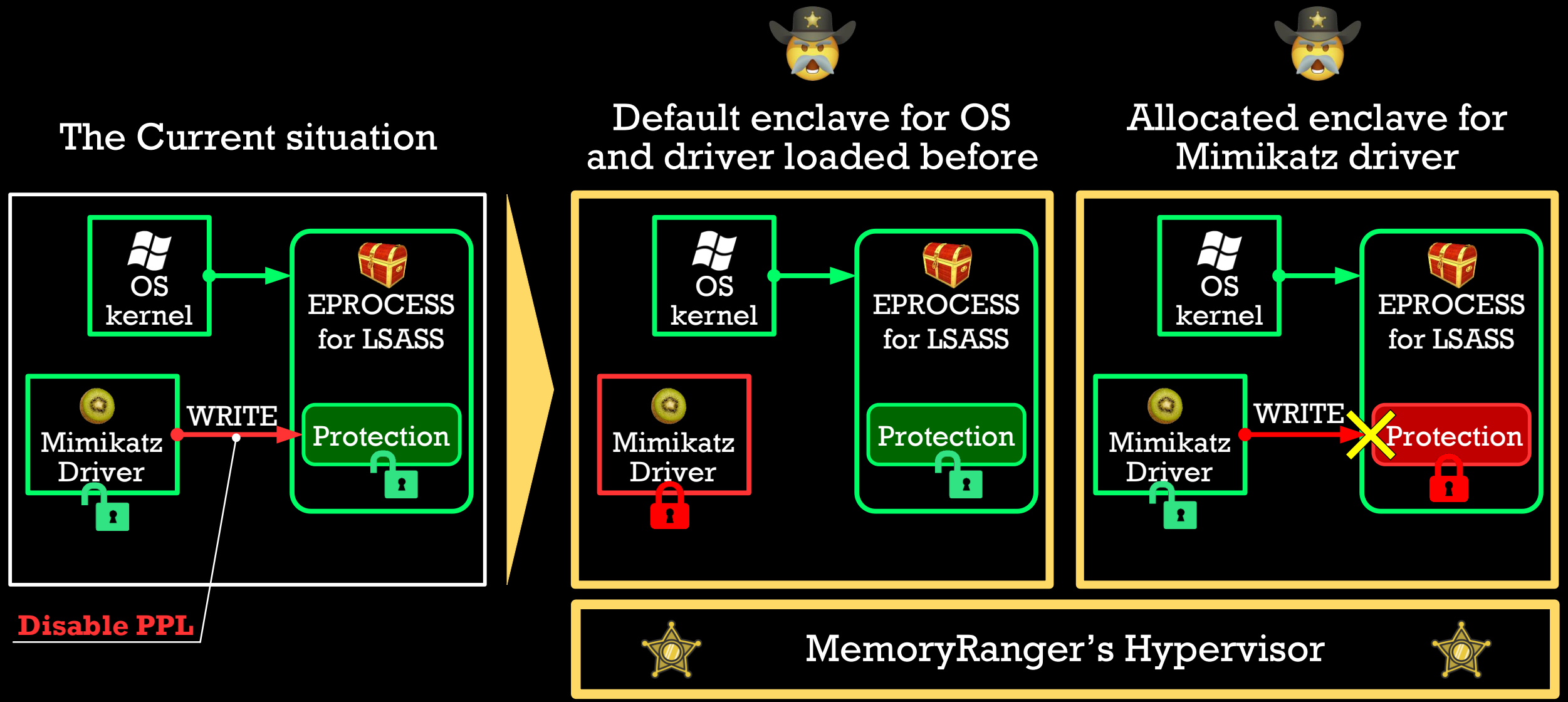
The Current situation



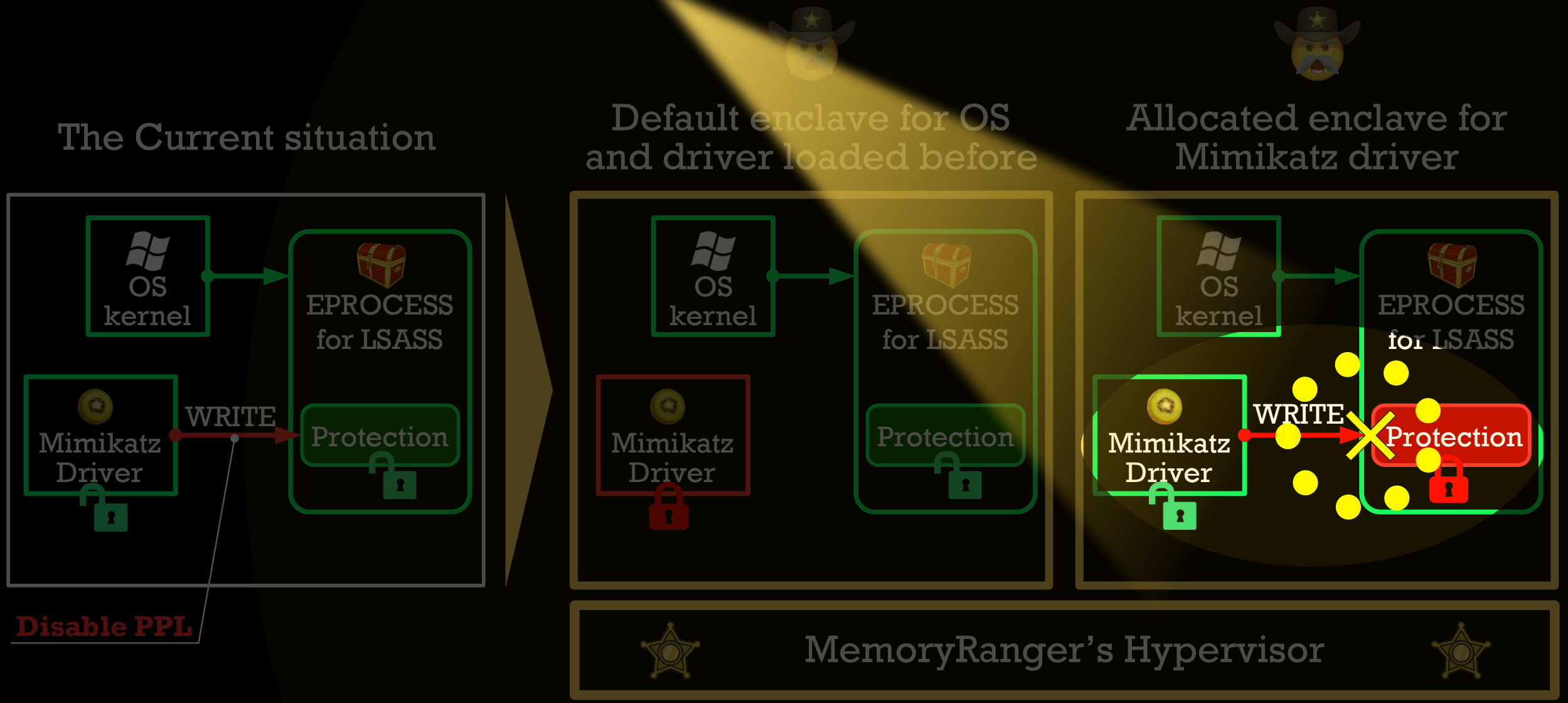
Default enclave for OS and driver loaded before



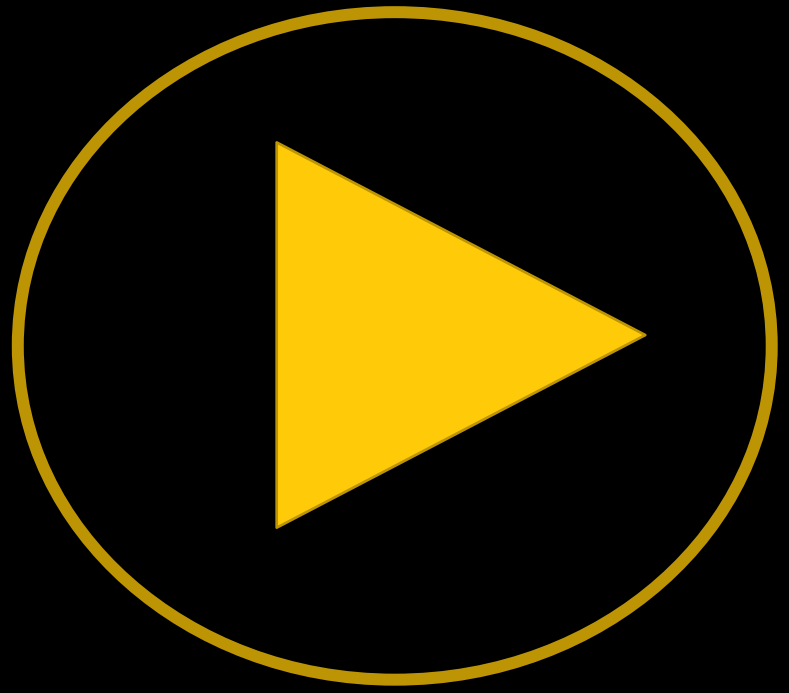
MemoryRanger Prevents Disabling of PPL



MemoryRanger Prevents Disabling of PPL

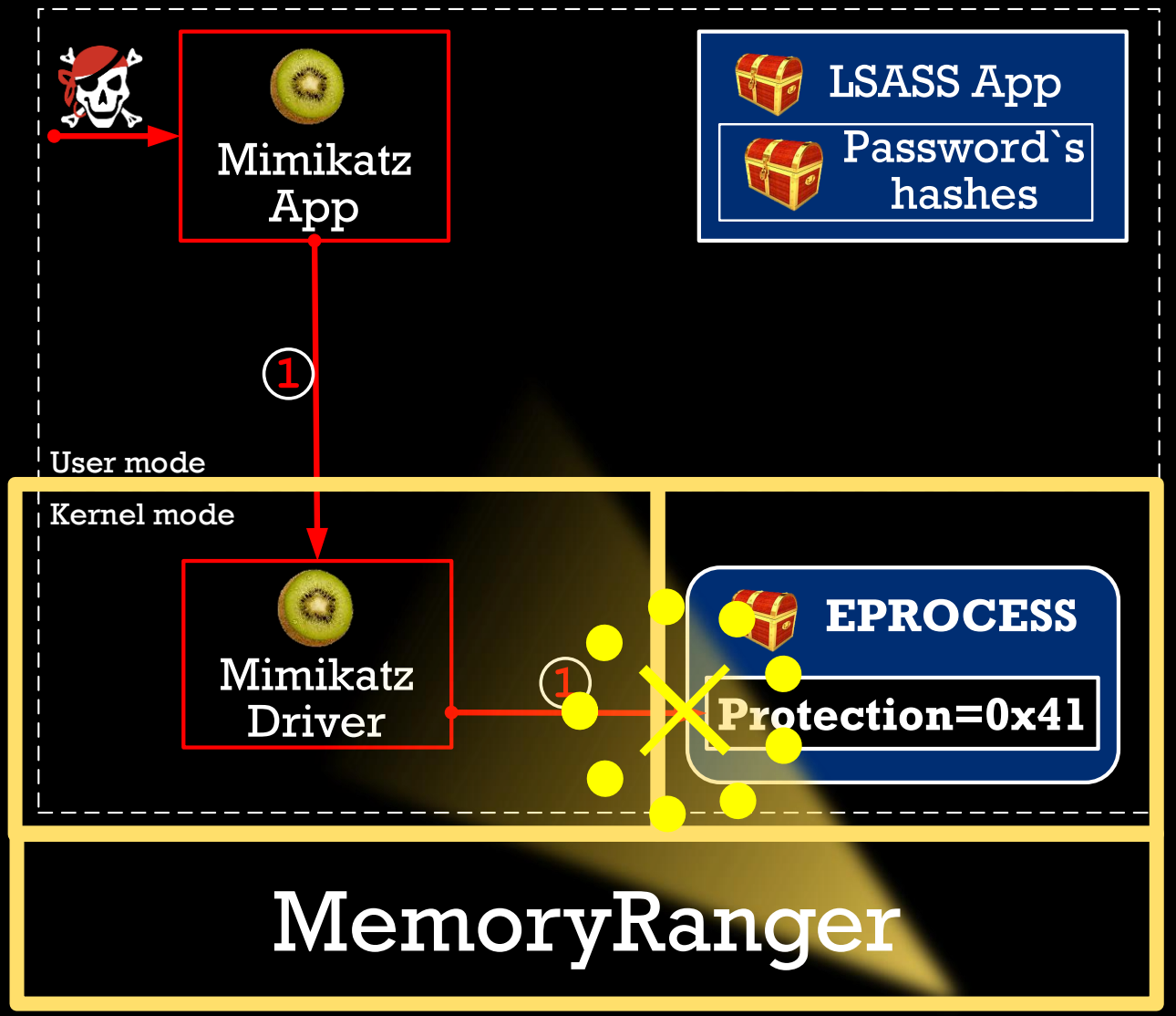


MemoryRanger Blocks Mimikatz and prevents disabling PPL



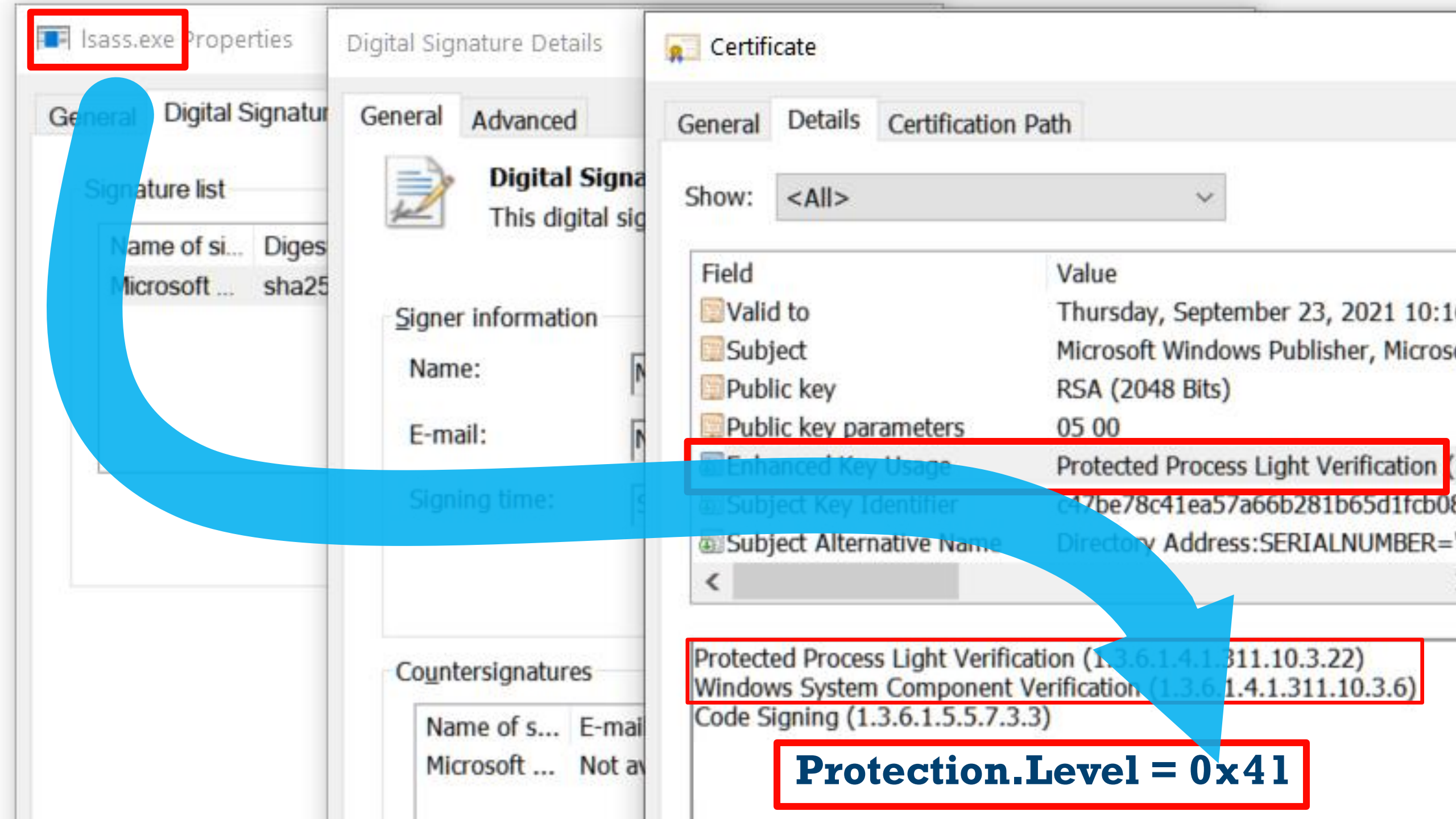
The online version is here –
https://www.youtube.com/embed/66g4PgtuD7c?vq=hd1440

MemoryRanger Prevents Disabling of PPL for LSASS



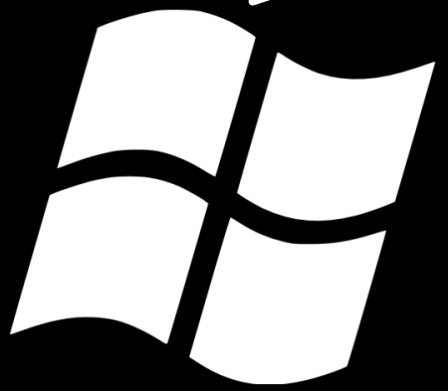
Episode 10

But what if malware can escalate its own PPL?


lsass.exe Properties
Digital Signatur
ature list
Name of si...
Diges
Microsoft ...
sha25
Digital Signature Details
General
Advanced
Digital Signa
This digital sig
Signer information
Name:
E-mail:
Signing time:
Countersignatures
Name of s...
E-mai
Microsoft ...
Not av
Certificate
General
Details
Certification Path
Show:
<All>
Field
Value
Valid to
Thursday, September 23, 2021 10:1
Subject
Microsoft Windows Publisher, Micros
Public key
RSA (2048 Bits)
Public key parameters
05 00
Protected Process Light Verification
Subject Key Identifier
Subject Alternative Name
Directory Address:SERIALNUMBER=
Protected Process Light Verification (1.3.6.1.4.1.311.10.3.22)
Windows System Component Verification (1.3.6.1.4.1.311.10.3.6)
Code Signing (1.3.6.1.5.5.7.3.3)
Protection.Level = 0x41

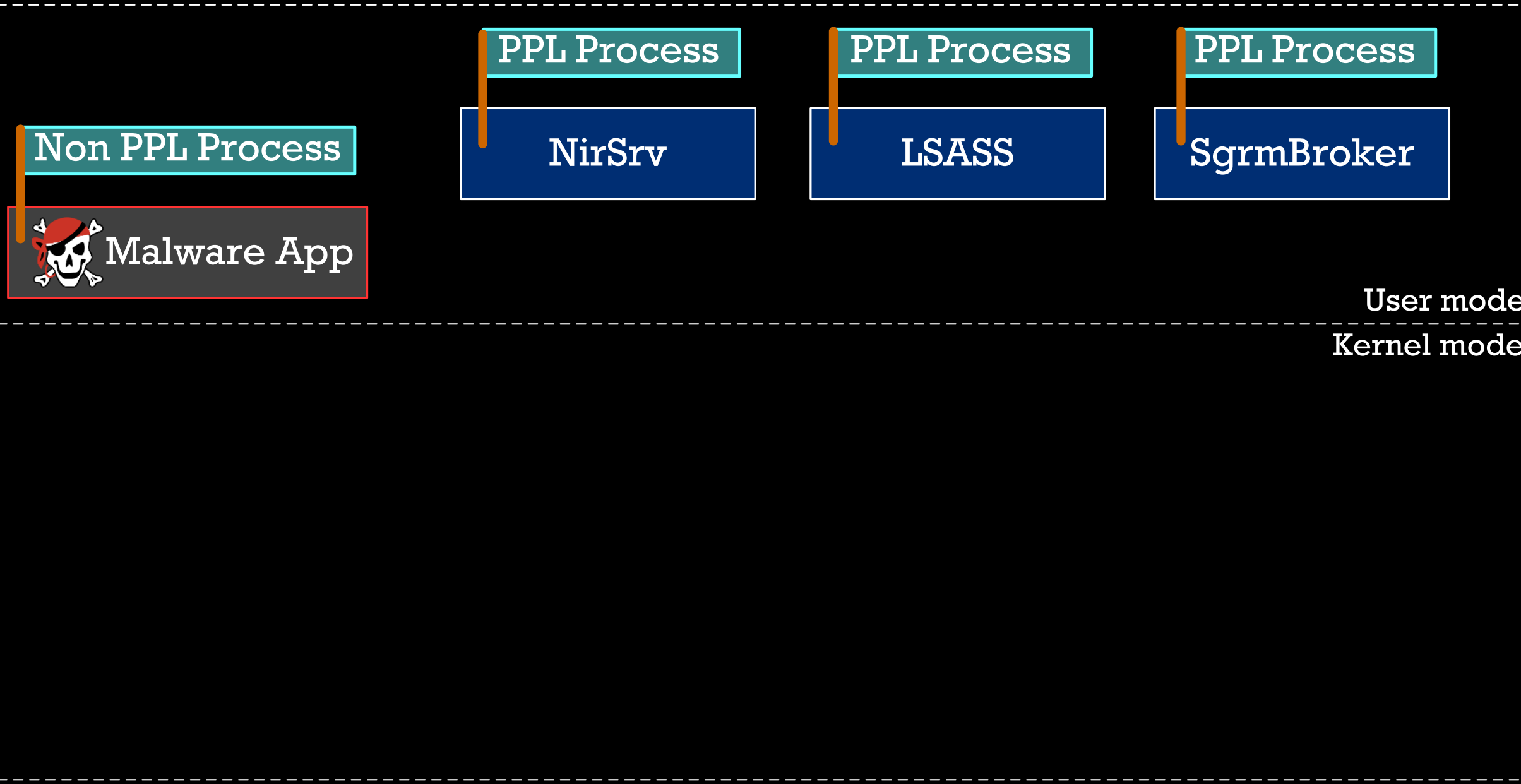
Code-Signing Certificate Determines the Protection Level

Malware does not have this kind of the certificate

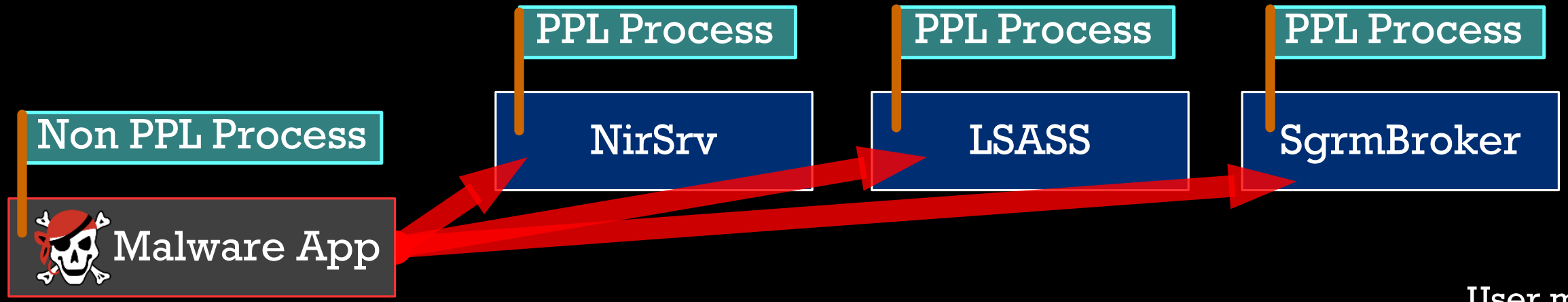


Malware doesn't bother about it!

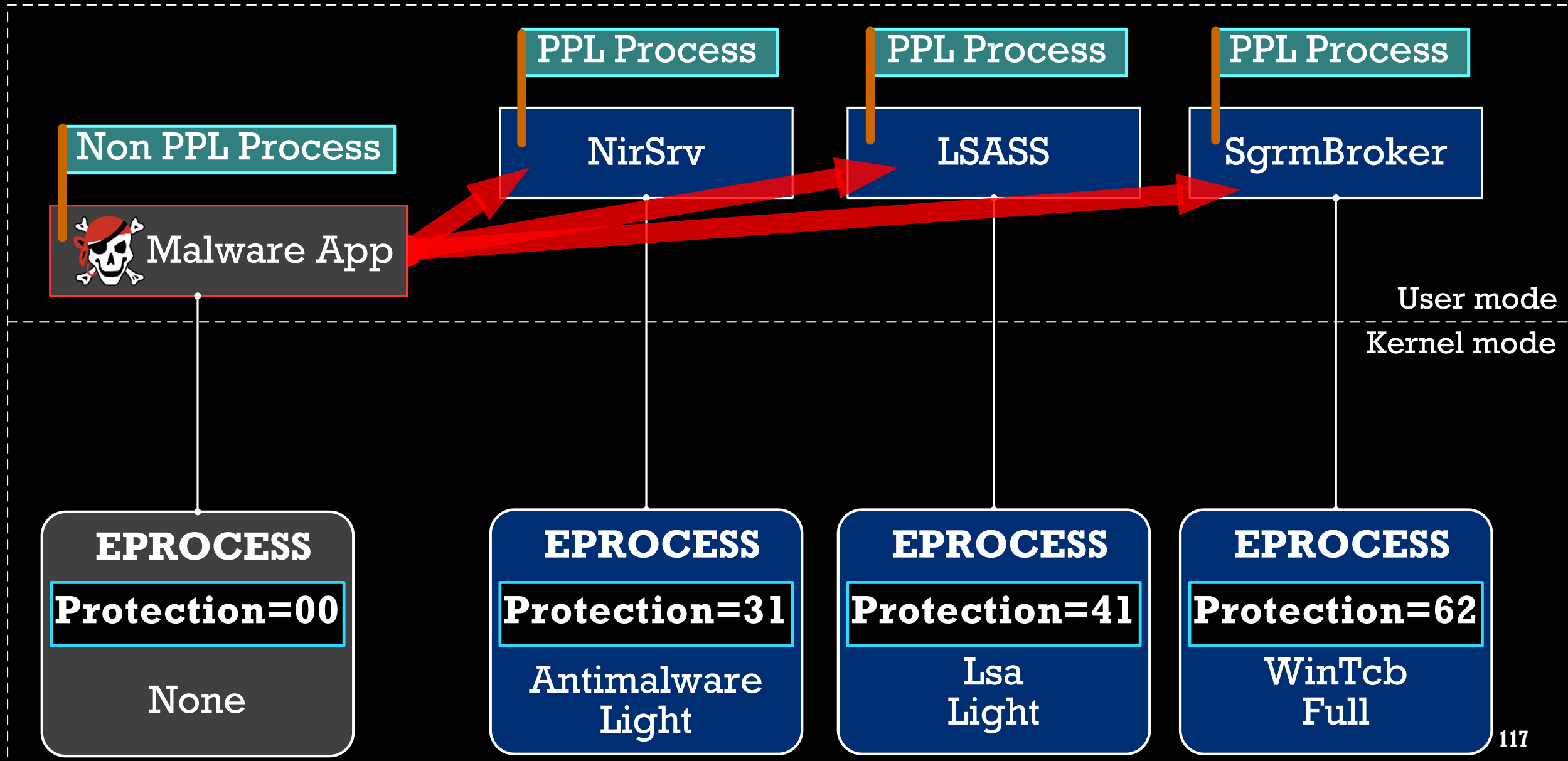
Malware can escalate its PPL to dump Protected Apps



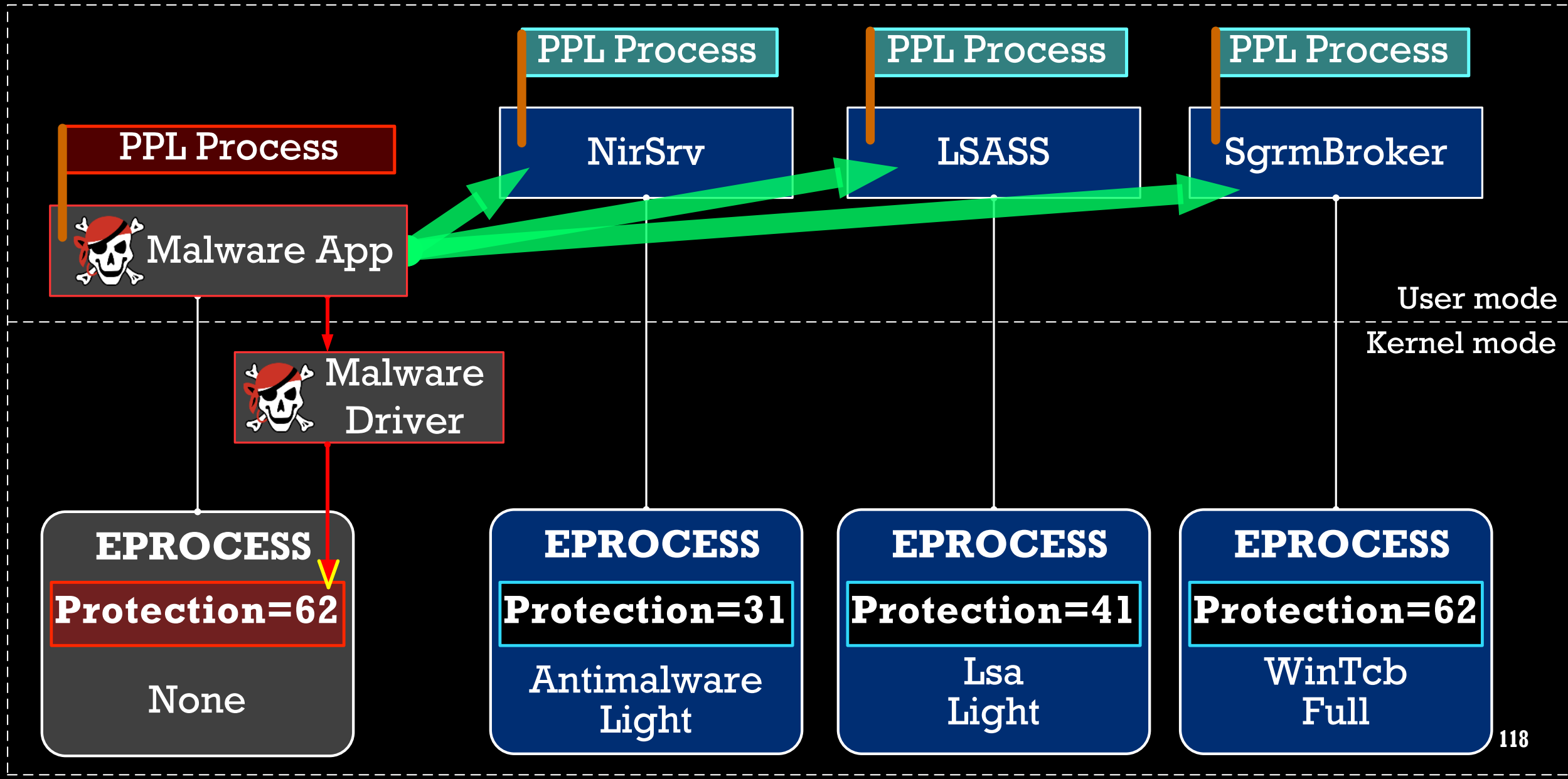
Malware can escalate its PPL to dump Protected Apps



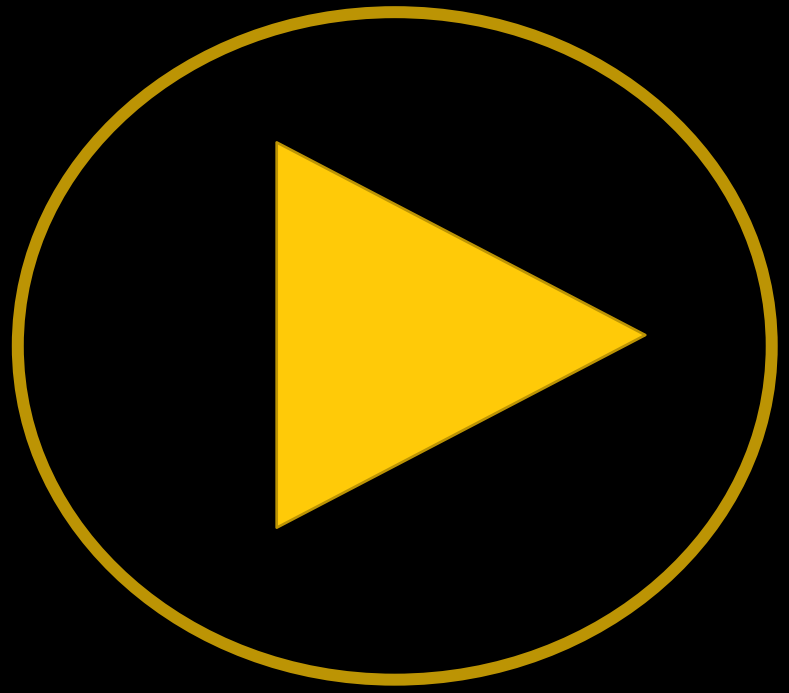
Malware can escalate its PPL to dump Protected Apps



Malware can escalate its PPL to dump Protected Apps

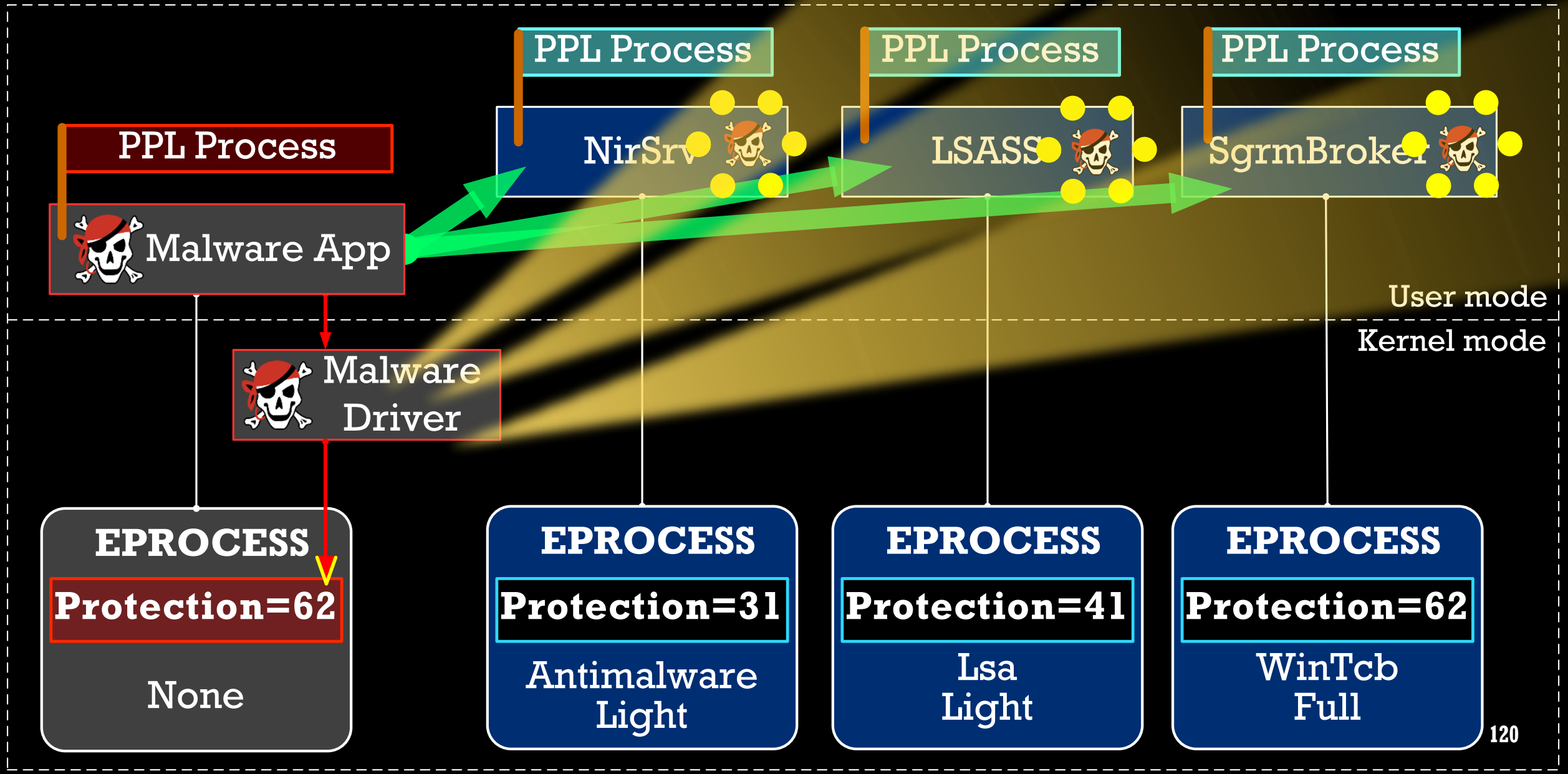


Malware escalates its PPL to dump Protected Processes



The online version is here –
https://www.youtube.com/embed/IELn8mcMZ4Q?vq=hd1440

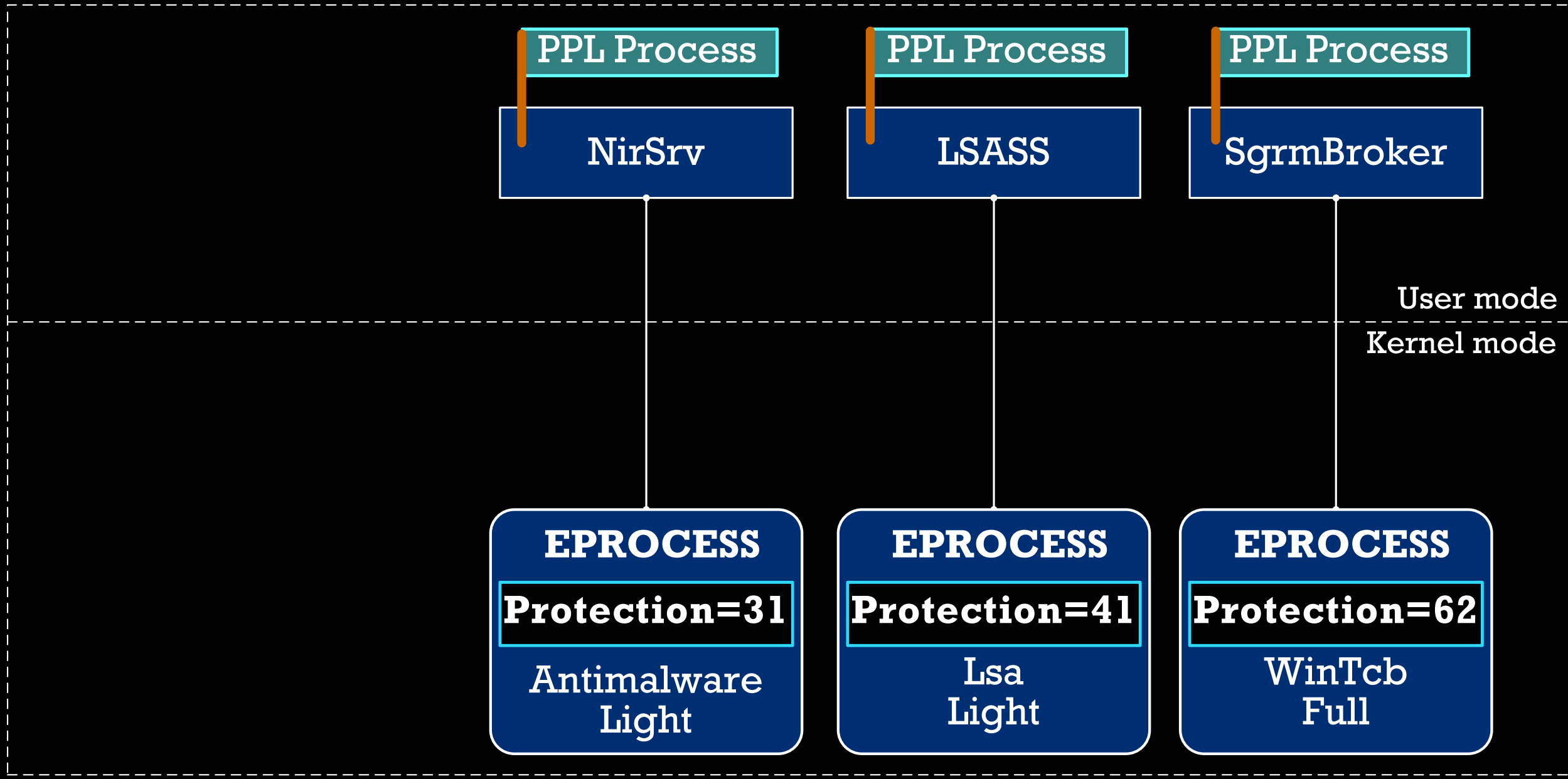
Malware can escalate its PPL to dump Protected Apps



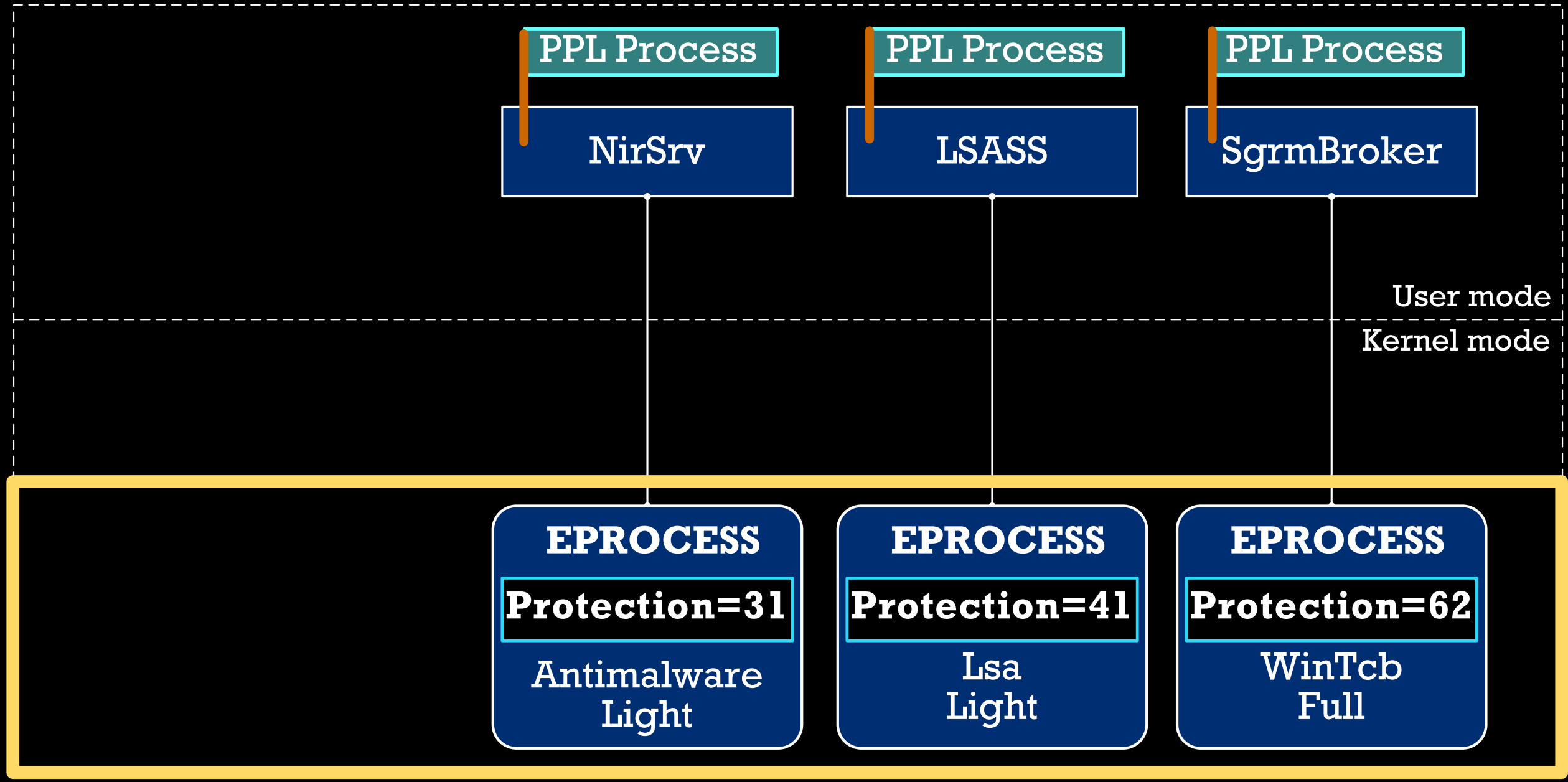
Episode 11

MemoryRanger can block malware

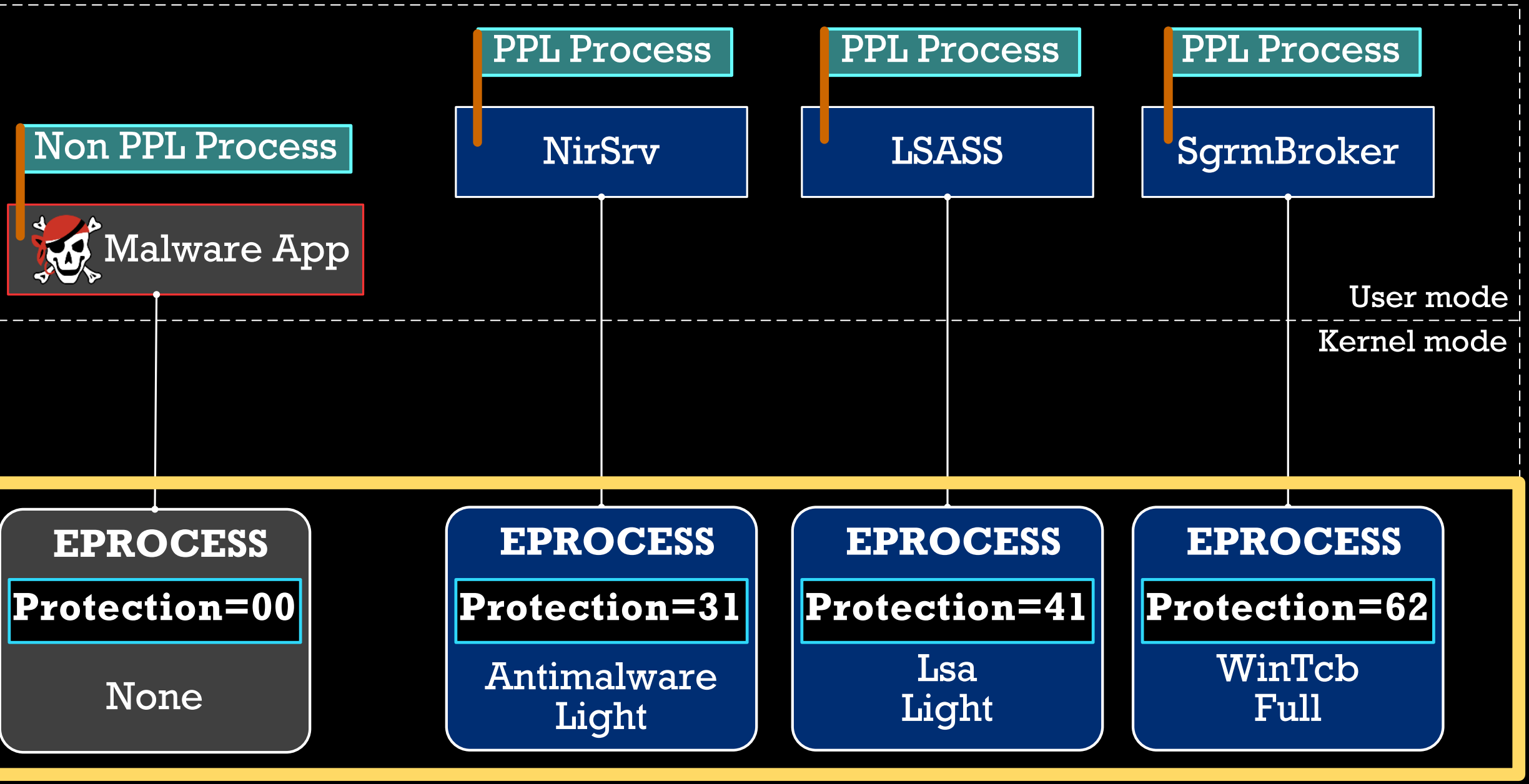
MemoryRanger blocks modifying PPL



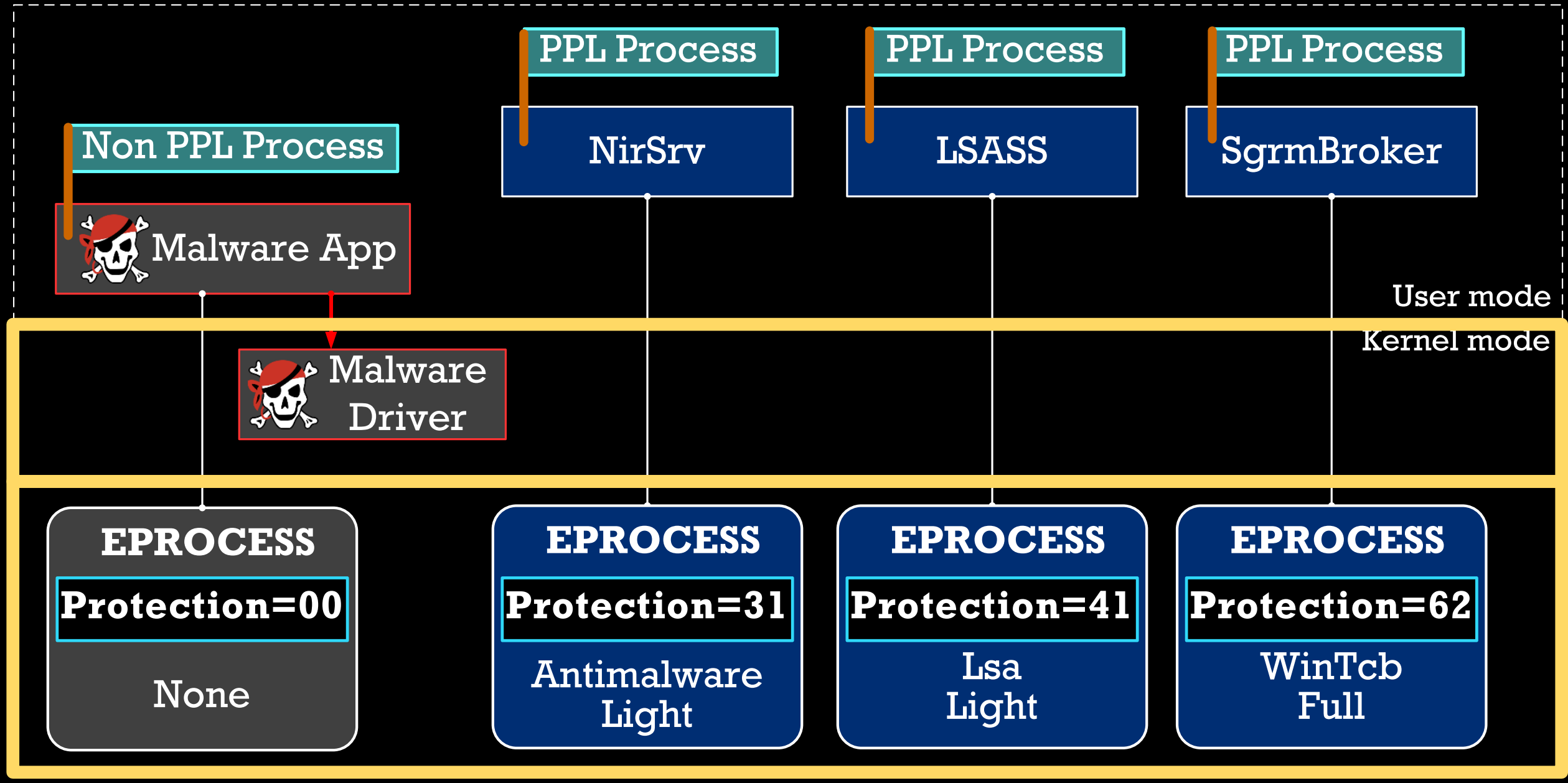
MemoryRanger blocks modifying PPL



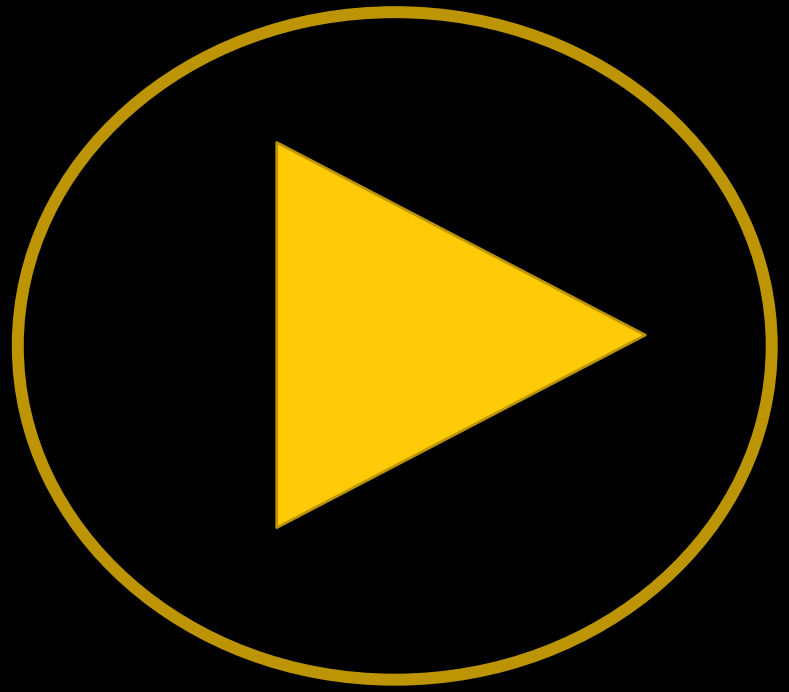
MemoryRanger blocks modifying PPL



MemoryRanger blocks modifying PPL

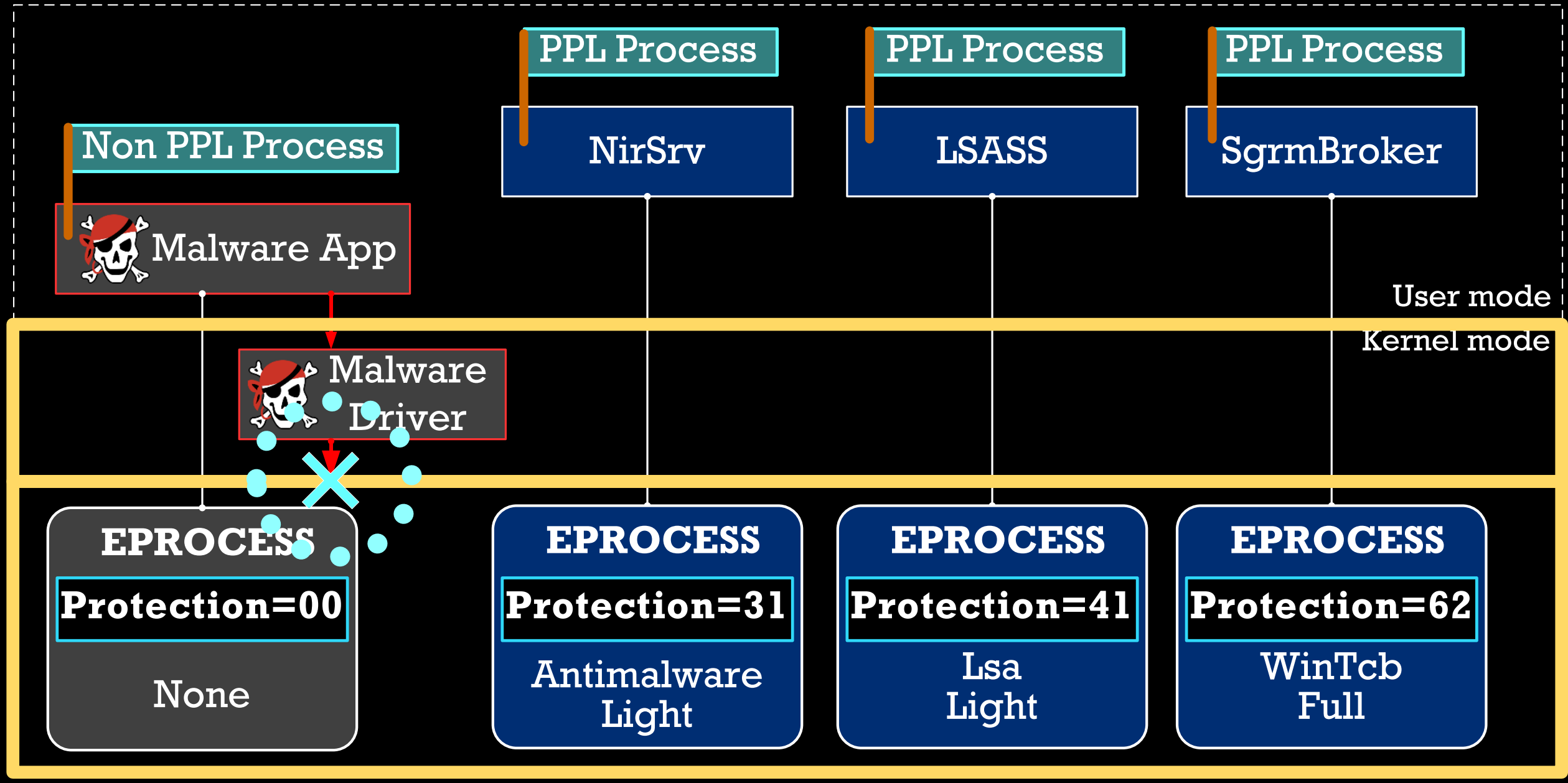


MemoryRanger prevents escalation of PPL



The online version is here –
https://www.youtube.com/embed/GEc-GOWtm8M?vq=hd1440

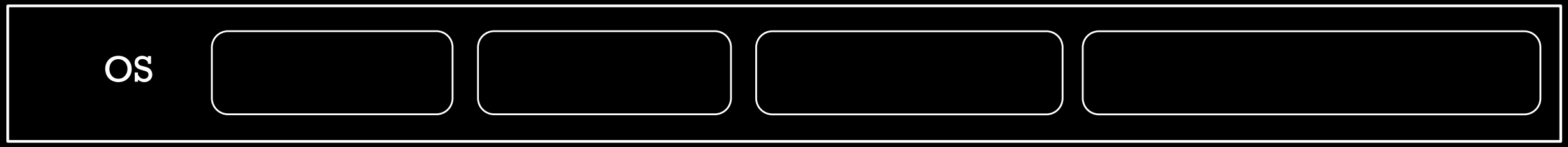
MemoryRanger blocks modifying PPL



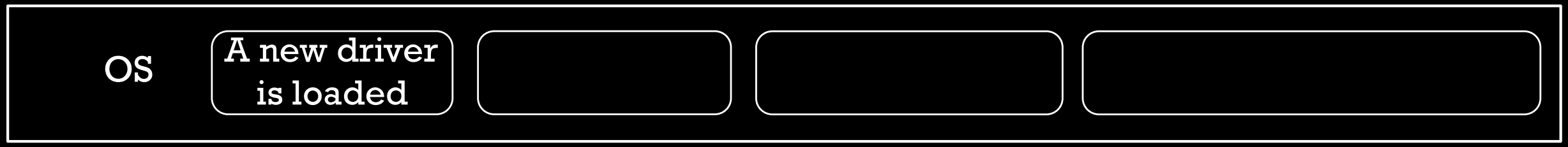
Episode 12

Architecture and Customization of MemoryRanger

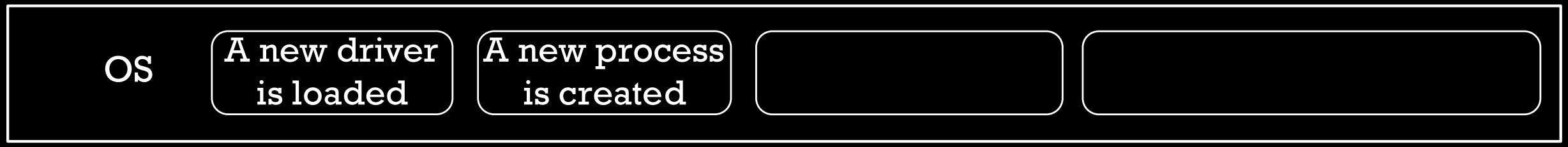
MEMORY RANGER ARCHITECTURE



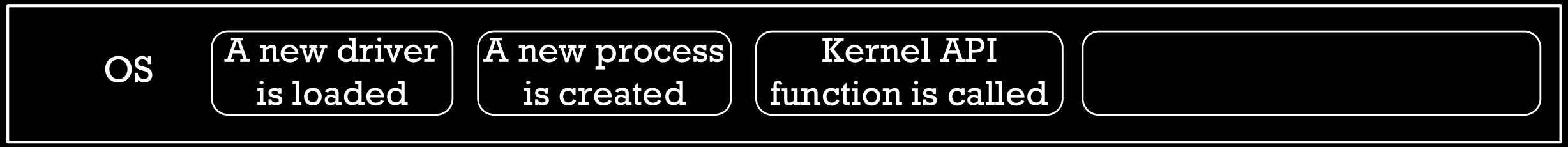
MEMORY RANGER ARCHITECTURE



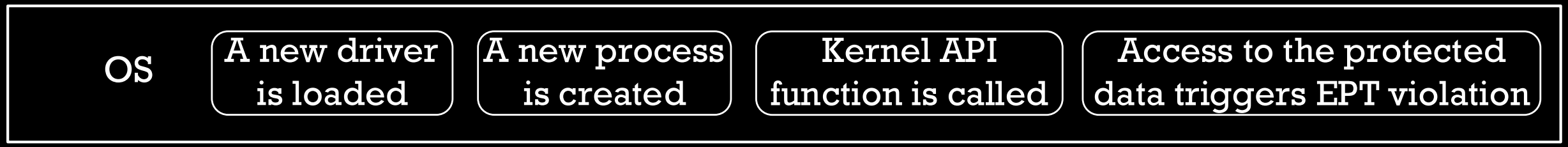
MEMORY RANGER ARCHITECTURE



MEMORY RANGER ARCHITECTURE



MEMORY RANGER ARCHITECTURE



MEMORY RANGER ARCHITECTURE

OS

A new driver is loaded

A new process is created

Kernel API function is called

Access to the protected data triggers EPT violation

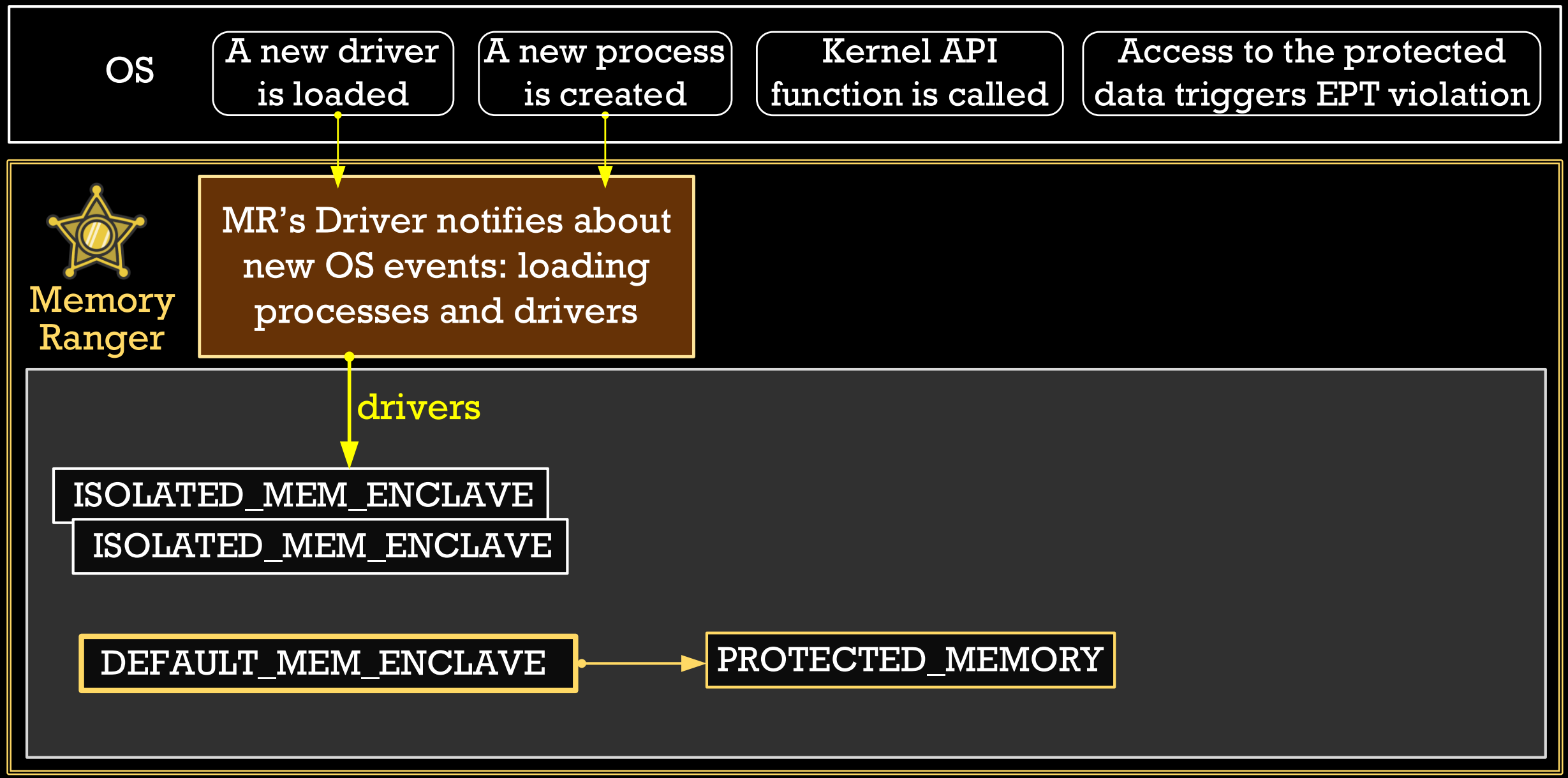
Memory Ranger

DEFAULT_MEM_ENCLAVE → PROTECTED_MEMORY

(Further details in the paper)

https://github.com/IgorKorkin/MemoryRanger

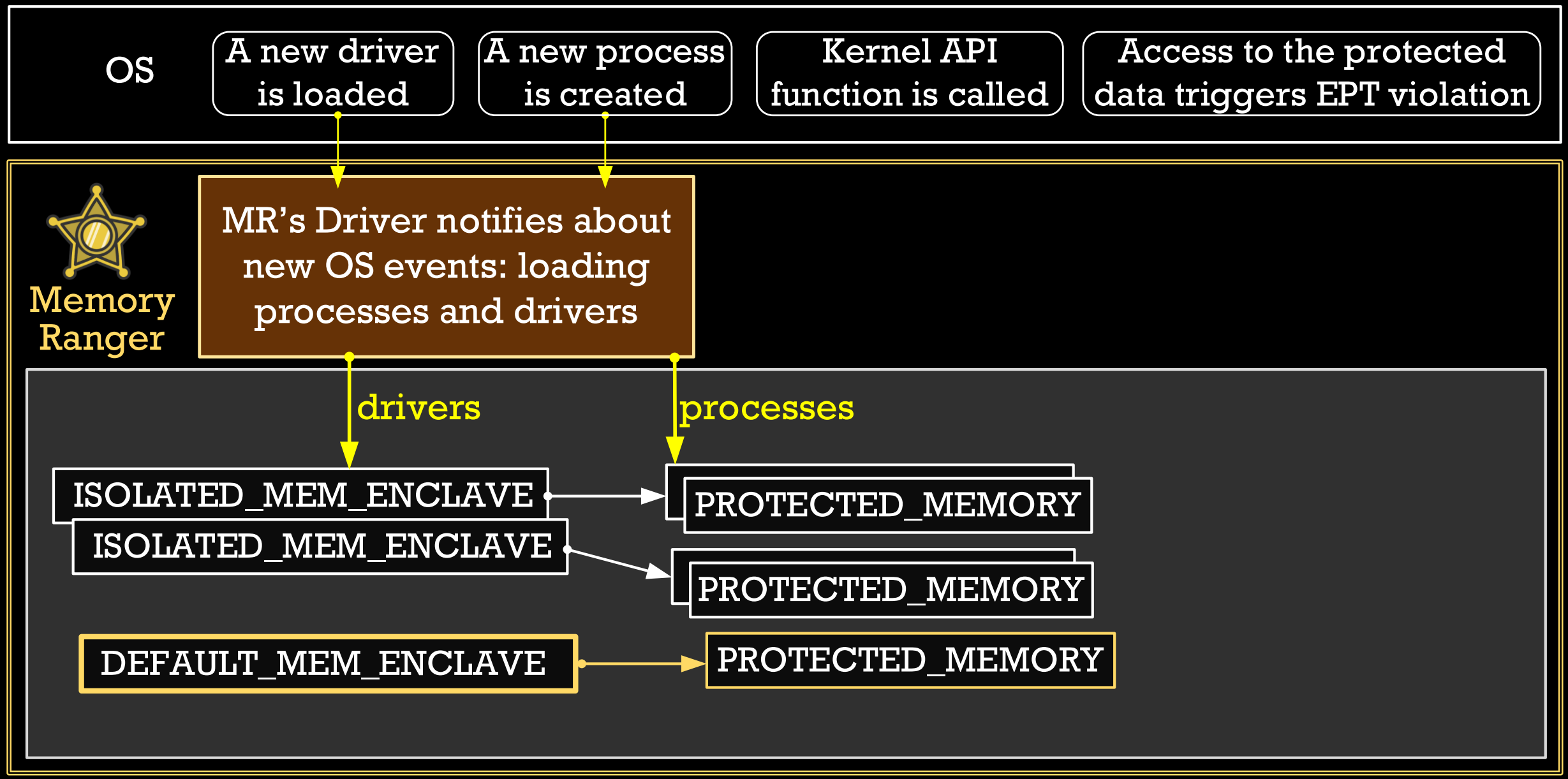
MEMORY RANGER ARCHITECTURE



(Further details in the paper)

https://github.com/IgorKorkin/MemoryRanger

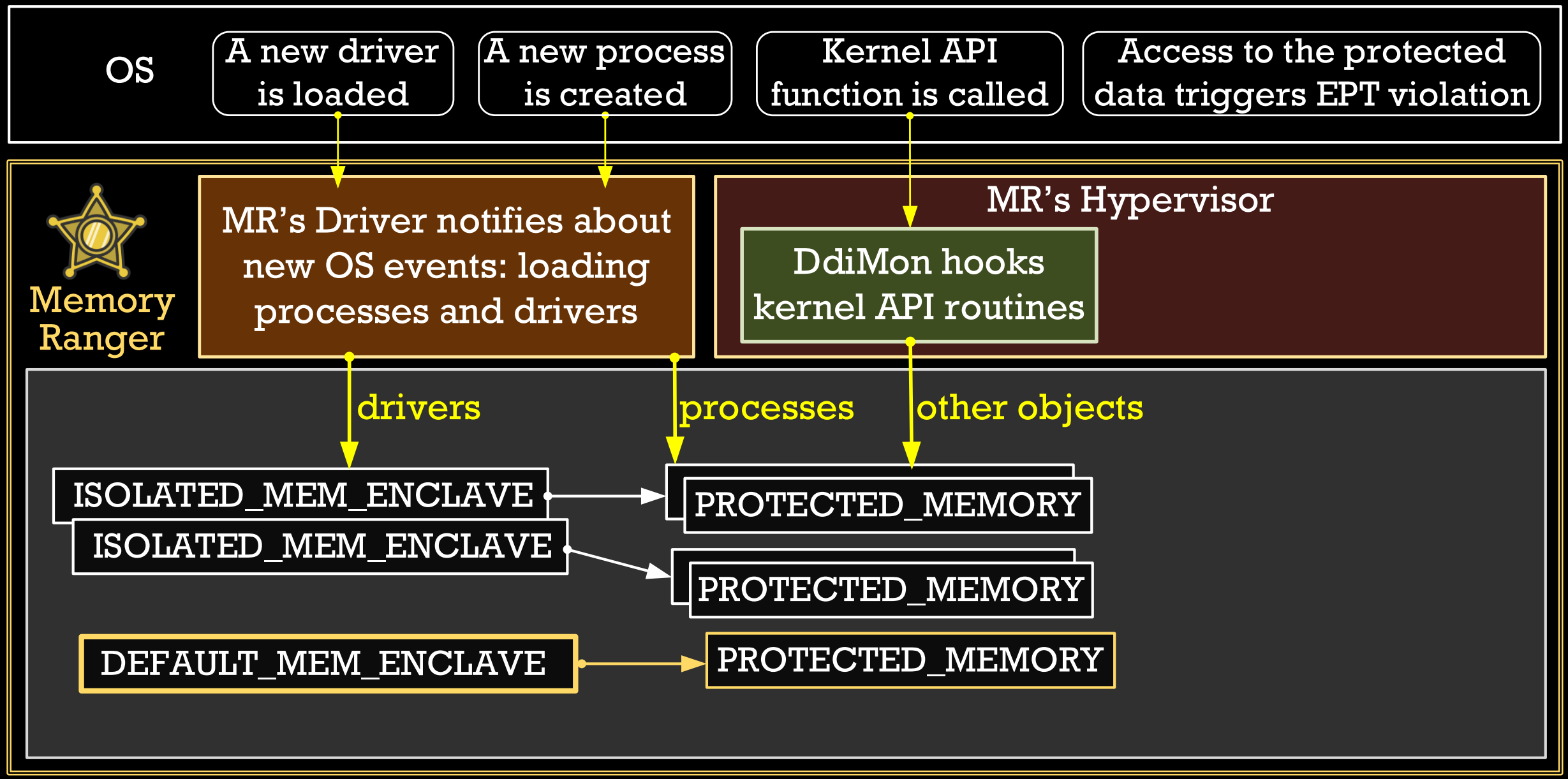
MEMORY RANGER ARCHITECTURE



(Further details in the paper)

https://github.com/IgorKorkin/MemoryRanger

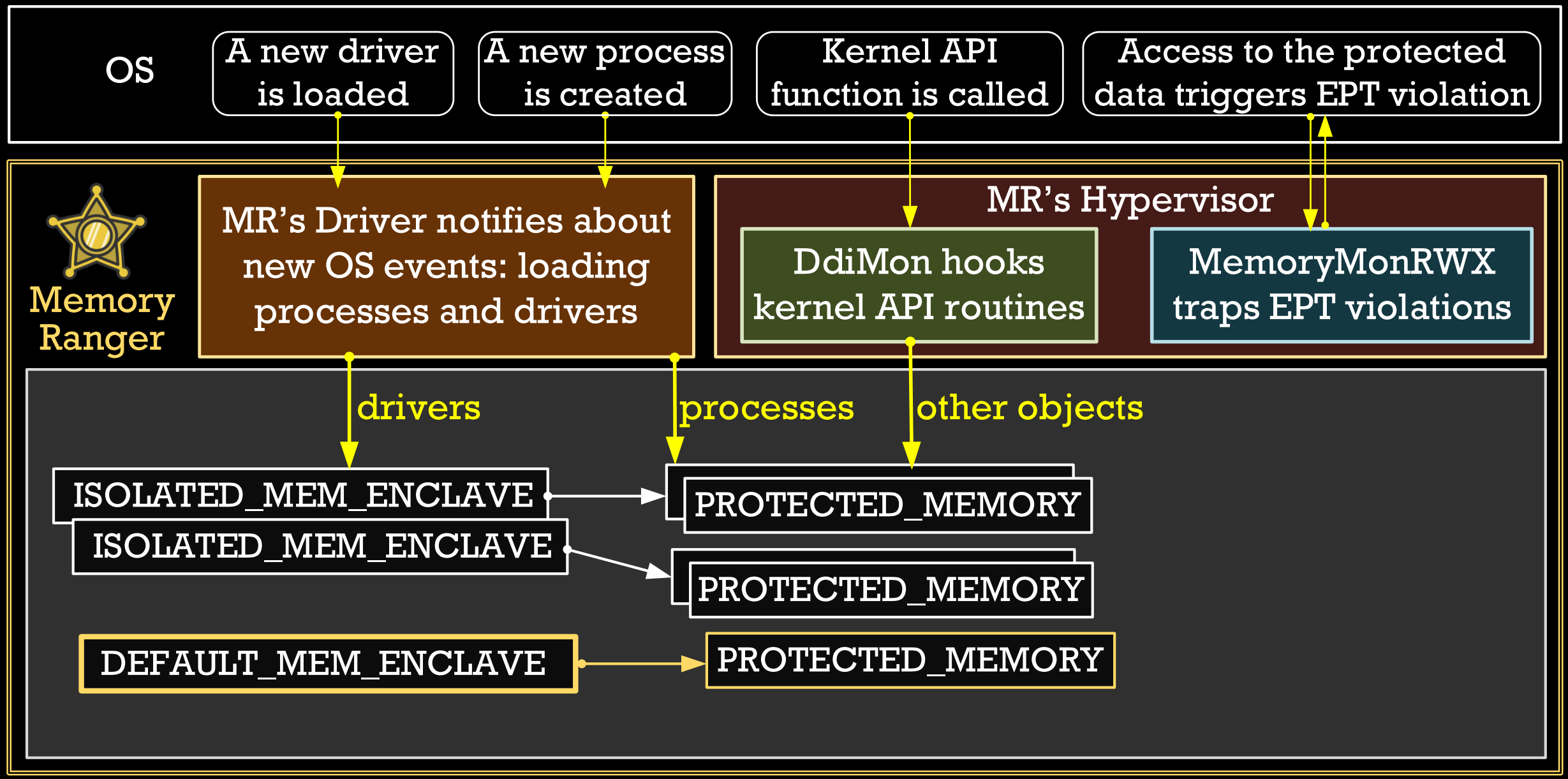
MEMORY RANGER ARCHITECTURE



(Further details in the paper)

https://github.com/IgorKorkin/MemoryRanger

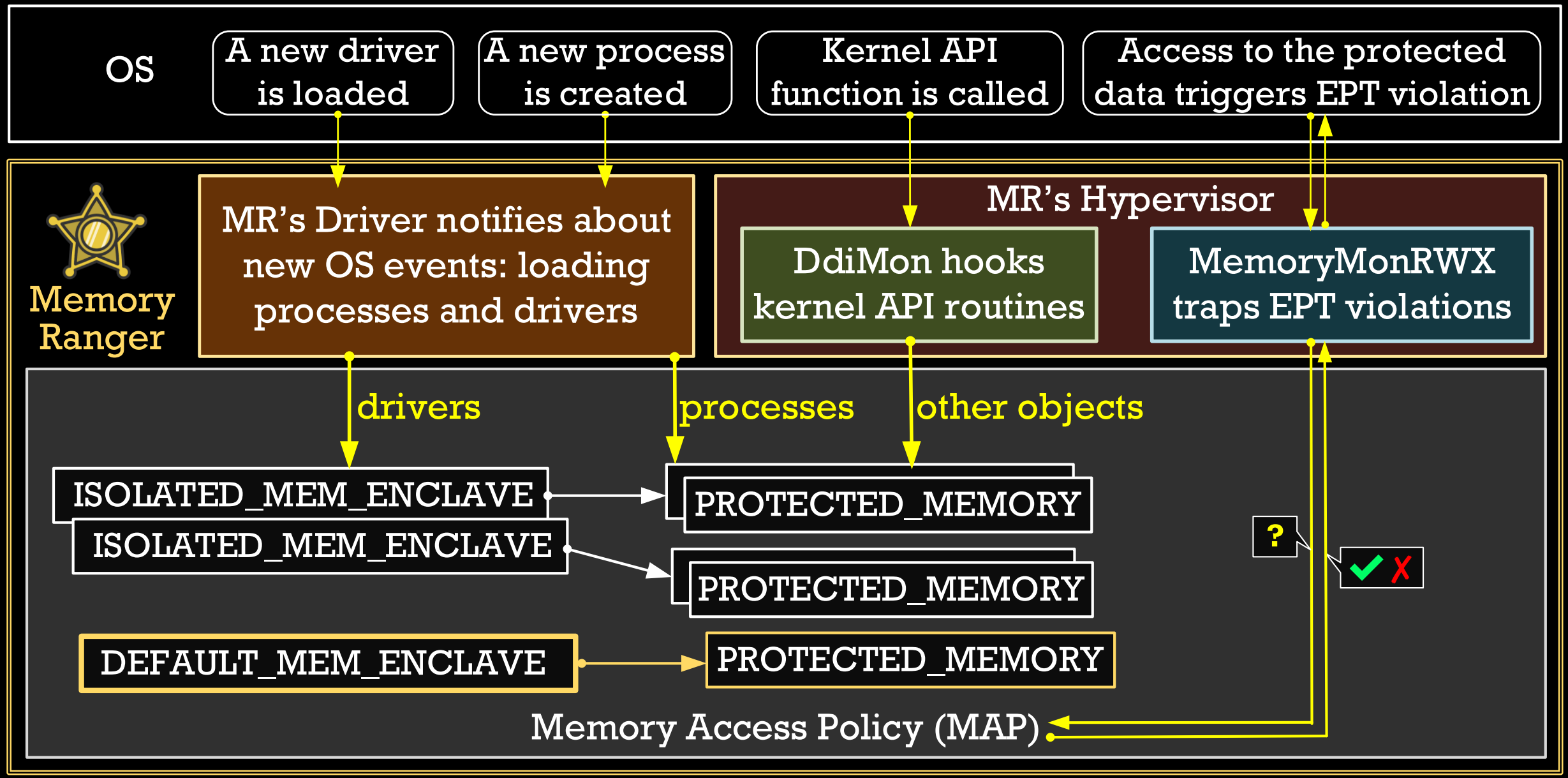
MEMORY RANGER ARCHITECTURE



(Further details in the paper)

https://github.com/IgorKorkin/MemoryRanger

MEMORY RANGER ARCHITECTURE



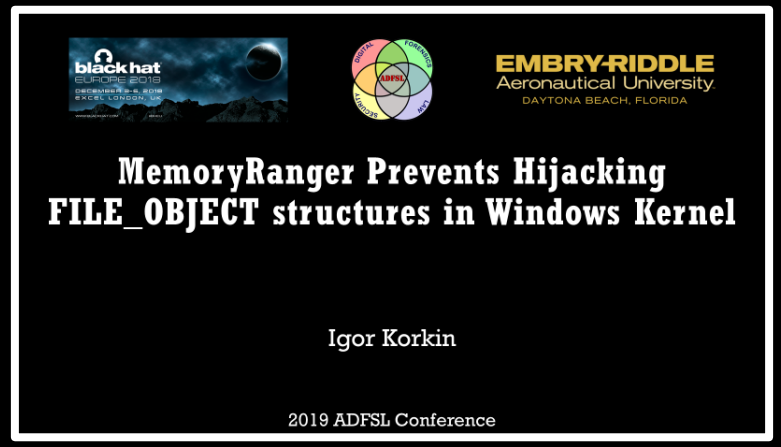
(Further details in the paper)

https://github.com/IgorKorkin/MemoryRanger

MemoryRanger: Previous Research

(2018) Divide et Impera: MemoryRanger Runs Drivers in Isolated Kernel Spaces

https://igorkorkin.blogspot.com/2018/12/divide-et-impera-memoryranger-runs.html



(2019) MemoryRanger Prevents Hijacking FILE_OBJECT Structures in Windows Kernel

https://igorkorkin.blogspot.com/2019/04/memoryranger-prevents-hijacking.html

(2020) Kernel Hijacking Is Not an Option: MemoryRanger Comes to the Rescue Again

https://conference.hitb.org/hitb-lockdown002/sessions/kernel-hijacking-is-not-an-option-memoryranger-comes-to-rescue-again/

MemoryRanger can Protect PPL

- MR's Driver:

- MR's Hypervisor

MemoryRanger can Protect PPL

- MR's Driver:
  - locates an address of Protection field for OS processes and newly created apps

- MR's Hypervisor

MemoryRanger can Protect PPL

- MR's Driver:
  - locates an address of Protection field for OS processes and newly created apps
  - traps loading of kernel drivers
- MR's Hypervisor

MemoryRanger can Protect PPL

- MR's Driver:
  - locates an address of Protection field for OS processes and newly created apps
  - traps loading of kernel drivers
- MR's Hypervisor provides discretionary access control mechanism:

	Default enclave
EPROCESS.Protection	**Read Write**
Mimikatz driver	**No Access**
Malware driver	**No Access**

MemoryRanger can Protect PPL

- MR's Driver:
  - locates an address of Protection field for OS processes and newly created apps
  - traps loading of kernel drivers
- MR's Hypervisor provides discretionary access control mechanism:

	Default enclave	Mimikatz enclave	Malware enclave
EPROCESS.Protection	**Read Write**	**No Access**	**No Access**
Mimikatz driver	**No Access**	**Read Write Execute**	**No Access**
Malware driver	**No Access**	**No Access**	**Read Write Execute**

MemoryRanger can Protect PPL

- MR's Driver:
  - locates an address of Protection field for OS processes and newly created apps
  - traps loading of kernel drivers
- MR's Hypervisor provides discretionary access control mechanism:
- Any access to the restricted memory causes EPT violations
- MR's Hypervisor:
  - For execute violation → switches an enclave;
  - For read\write violation → blocks an access by redirecting to the fake page

CONCLUSION

1. Windows Security Model does not restrict apps running with debug privilege.
2. Protected Process Light (PPL) protects memory of OS and AV processes.
3. Attackers can abuse PPL in the following ways:
   - protect their malware by illegally enabling PPL
   - steal and modify data of protected processes by illegally disabling PPL
4. MemoryRanger blocks attacks on kernel data including attacks on PPL.

Thank you!

Igor Korkin igor.korkin@gmail.com

All the details are here igorkorkin.blogspot.com