LSASS Shtinkering

Abusing Windows Error Reporting to Dump LSASS

About Us

Asaf Gilboa

Security Researcher

Found & implemented the
LSASS Shtinkering technique

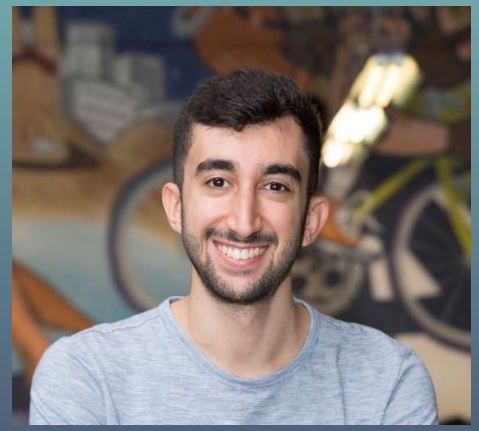
Weightlifter, gamer, traveler, gluten addict

Ron Ben-Yizhak

Security Researcher

Interested in malware campaigns, attack vector and evasion techniques

Enjoys rock climbing and volleyball



Agenda

01

Memory Dumping Techniques
Overview of known techniques and tools

02

LSASS Shtinkering
Reverse Engineering the WER Client Side

03

LSASS Shtinkering
Reverse Engineering the WER Server Side

04

Detection & Prevention
How to stop the attack

Credential Access

- Covers many types of attacks
- This method is for “OS Credential Dumping: LSASS Memory” (T1003.001)
- Actors try to obtain credentials to move laterally through the network
- Credentials allows adversaries to run ransomware remotely
- Effort of exploiting vulnerabilities is saved with valid credentials
- The prime goal is to gain execution on the domain controller

Credential Access
16 techniques

Adversary-in-the-Middle (3)	Multi-Factor Authentication Request Generation
Brute Force (4)	Network Sniffing
Credentials from Password Stores (5)	OS Credential Dumping (8)
Exploitation for Credential Access	Steal Application Access Token
Forced Authentication	Steal or Forge Kerberos Tickets (4)
Forge Web Credentials (2)	Steal Web Session Cookie
Input Capture (4)	Unsecured Credentials (7)
Modify Authentication Process (5)	
Multi-Factor Authentication Interception	

Credential Access in the Wild

Credential and Data Theft

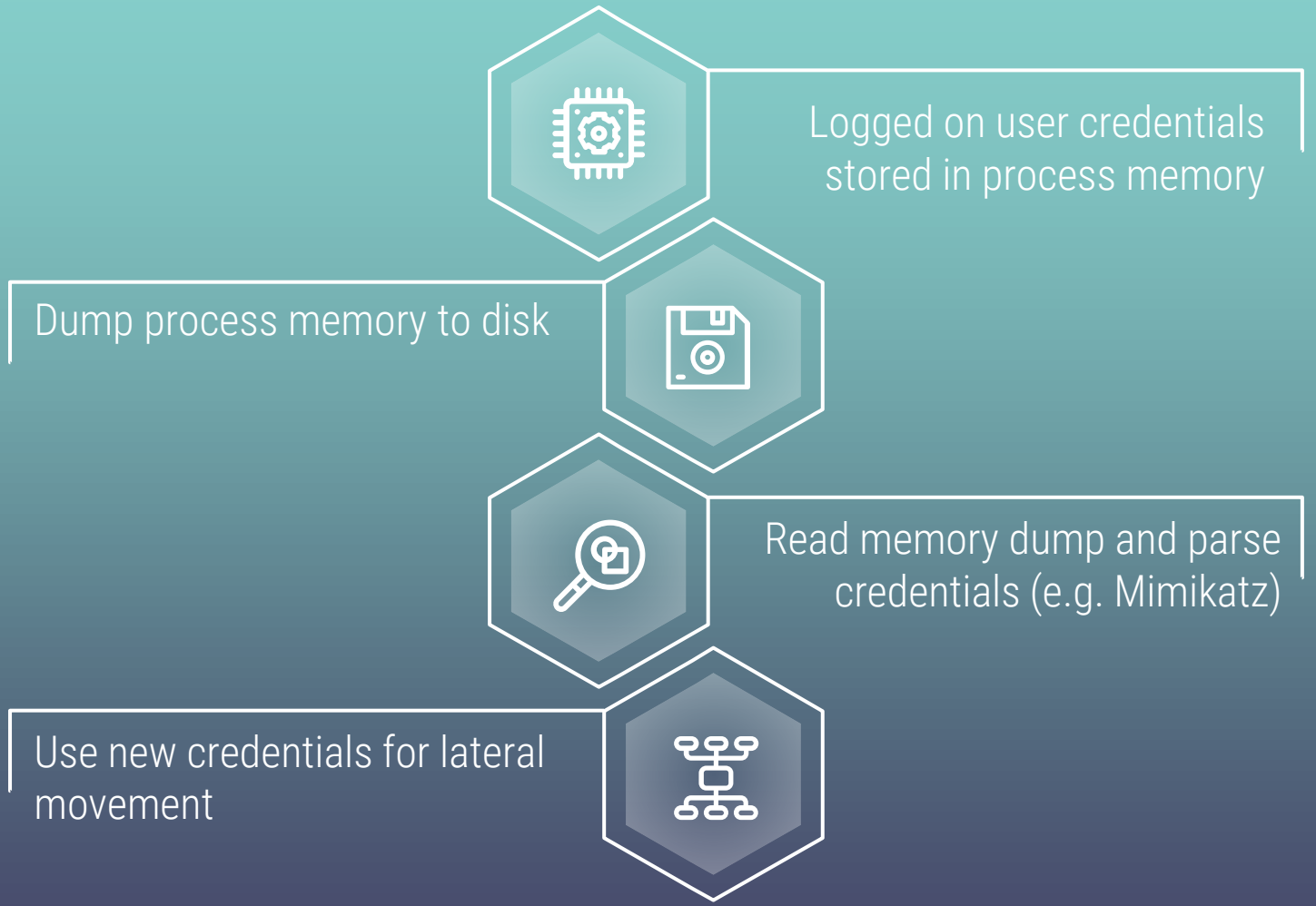
Conti actors steal credentials by dumping the memory of the *Local Security Authority Subsystem Service* (*lsass*) process. Conti actors download PowerShell payload from an attacker-controlled endpoint, such as *httpx://datasecuritytoday[.]com::757/securiday*, which dumps credentials from *lsass*:

Cybereason Global SOC Team:
From Shathak Emails to the Conti Ransomware

	Conti	Pysa	Clop (TA505)	Hive	Ragnar Locker	Lockbit	BlackByte	BlackCat
OS Credential Dumping: LSASS Memory T1003.001	✓	✓	✓	✓	✓	✓	✓	✓

Kaspersky Crimeware Reports:
Common TTPs of modern ransomware groups

Credentials Dumping Flow



Introduction

- **Local Security Authority Subsystem Service**
  - System process for managing the authentication procedure
  - Verifies user logons (local and remote)
  - Forced termination will result in a restart
- **The Problem**
  - The LSASS process has SSO (Single-Sign-On)
  - SSO requires credentials to be stored in memory
  - Any process can extract these credentials from the LSASS process
  - Often done by dumping LSASS to disk

```
NTSTATUS  MiniDumpWriteDump(
    _In_    HANDLE    ProcessHandle,
    ...
    _In_    HANDLE    hFile,
    ...);
```

Introduction

- **Windows Error Reporting service**
  - Comes with all Windows versions
  - Gathers information about software crashes
  - Can dump memory of crashing user-mode processes for further analysis
- **End goal**
  - Find a new stealthy way to perform credentials dumping
  - Force Windows Error Reporting to dump the memory of LSASS
  - Evade EDR solutions

Existing Dumping Techniques

- **ProcDump**
  - Part of SysInternals
  - Signed By Microsoft
  - *procdump.exe -ma lsass.exe lsass.dmp*
  - Command line easy to detect
- **ComSvcs.dll**
  - Native DLL found on all Windows OS versions
  - *rundll32.exe C:\windows\System32\comsvcs.dll, MiniDump <lsass pid> lsass.dmp full*
  - Command line easy to detect
- **Task Manager**
  - Signed Native exe found on all Windows OS versions
  - *Right Click lsass.exe -> Create dump file*
  - Dumping activity still stands out

Existing Dumping Techniques

- **SilentProcessExit**
  - Documented mechanism since Windows 7
  - Activated when a process exits or is terminated by a foreign process
  - Offers one of the three actions:
    - Show message box
    - Launch a new process
    - Create dump file
  - Requires setting the following registry keys:
    - HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\lsass.exe
    - HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit\lsass.exe
  - Triggered by calling RtlReportSilentProcessExit

```
NTSTATUS  RtlReportSilentProcessExit(
    _In_    HANDLE    ProcessHandle,
    _In_    NTSTATUS  ExitStatus,
);
```

https://www.deepinstinct.com/blog/lsass-memory-dumps-are-stealthier-than-ever-before-part-2

Silent Process Exit

Benjamin Delpy Retweeted

/r/netsec
@_r_netsec
Automated

New LSASS Dumping Method via SilentProcessExit
deepinstinct.com/2021/02/16/lsa...

6:43 PM · Feb 23, 2021 ·

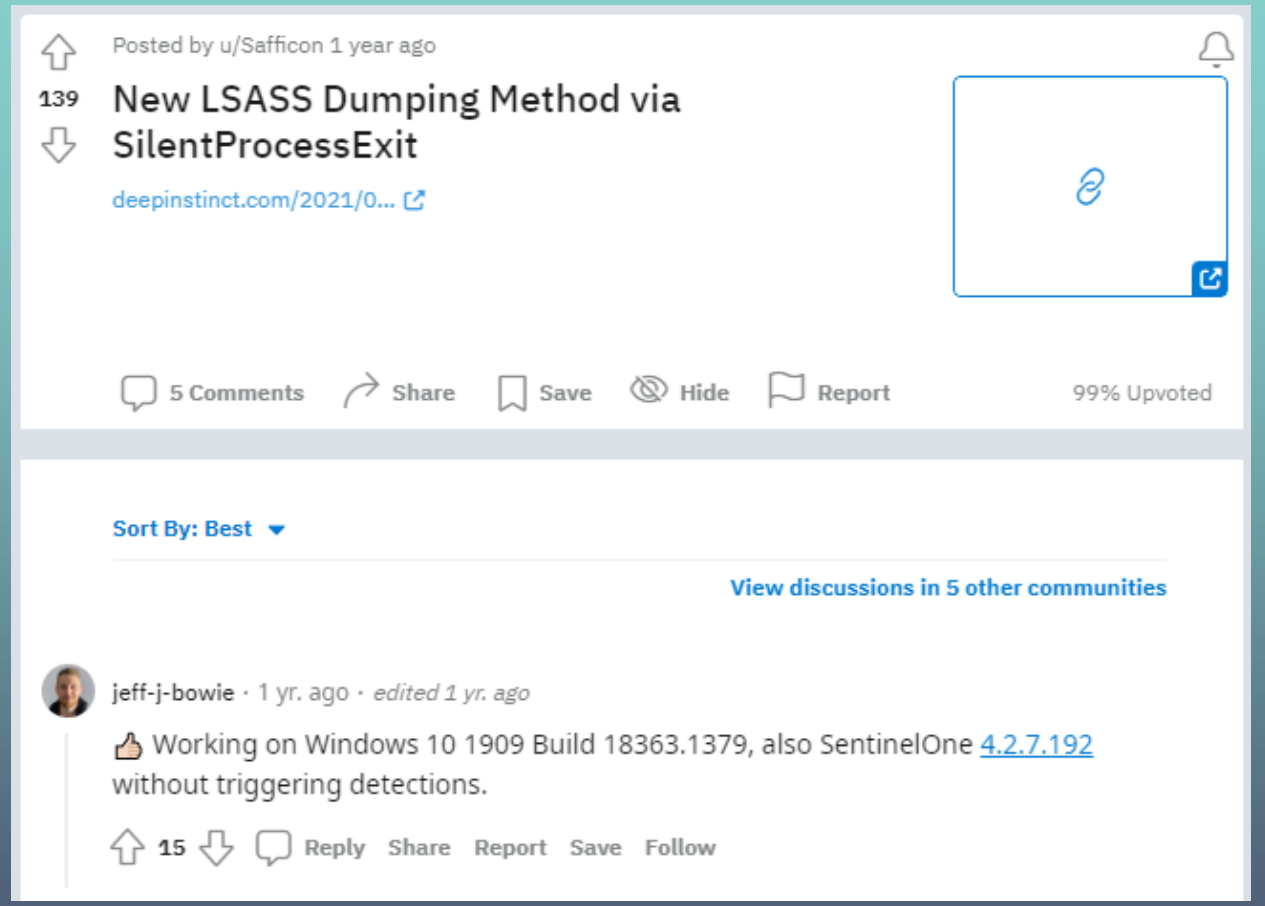
197 Retweets **2** Quote Tweets **395** Likes

Florian Roth ⚡
@cyb3rops

LSASS Memory Dumps are Stealthier than Ever Before Part 2 deepinstinct.com/2021/02/16/lsa...

9:28 AM · Feb 26, 2021 · Twitter for iPhone

111 Retweets **2** Quote Tweets **259** Likes



Protected Process Light

- ✓ LSASS can be launched as a Process Protected Light (PPL)
- ✓ Prevents tampering and termination of specially-signed programs
- ✓ Determined by a field in the EPROCESS that is checked by WinAPI
- ✓ Handle for LSASS opened by a non-PPL process is insufficient for the attacks

- x Setting LSASS as PPL is not applicable for organizations:
  - x Prevents third-party DLLs from loading into LSASS
  - x Benign authentication packages cannot be used

Issues

Easy to Identify

Command lines stand out

Stands Out

MiniDumpWriteDump on LSASS coming from Task Manager isn't normal

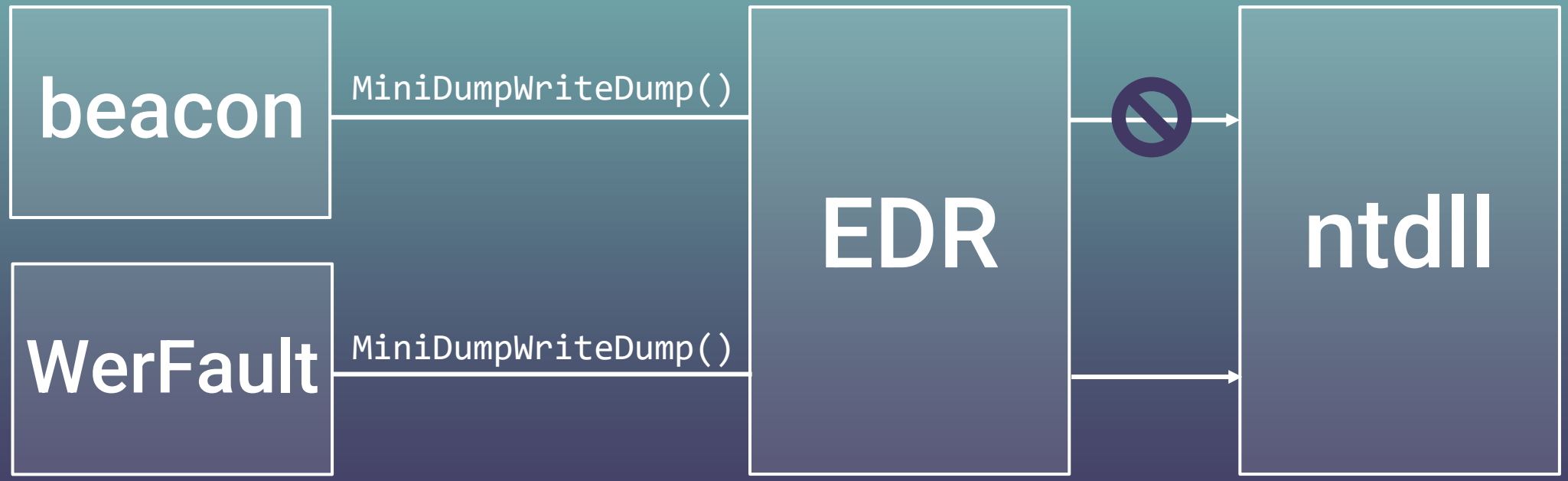
Deny-Listed File

ProcDump could be deny-listed

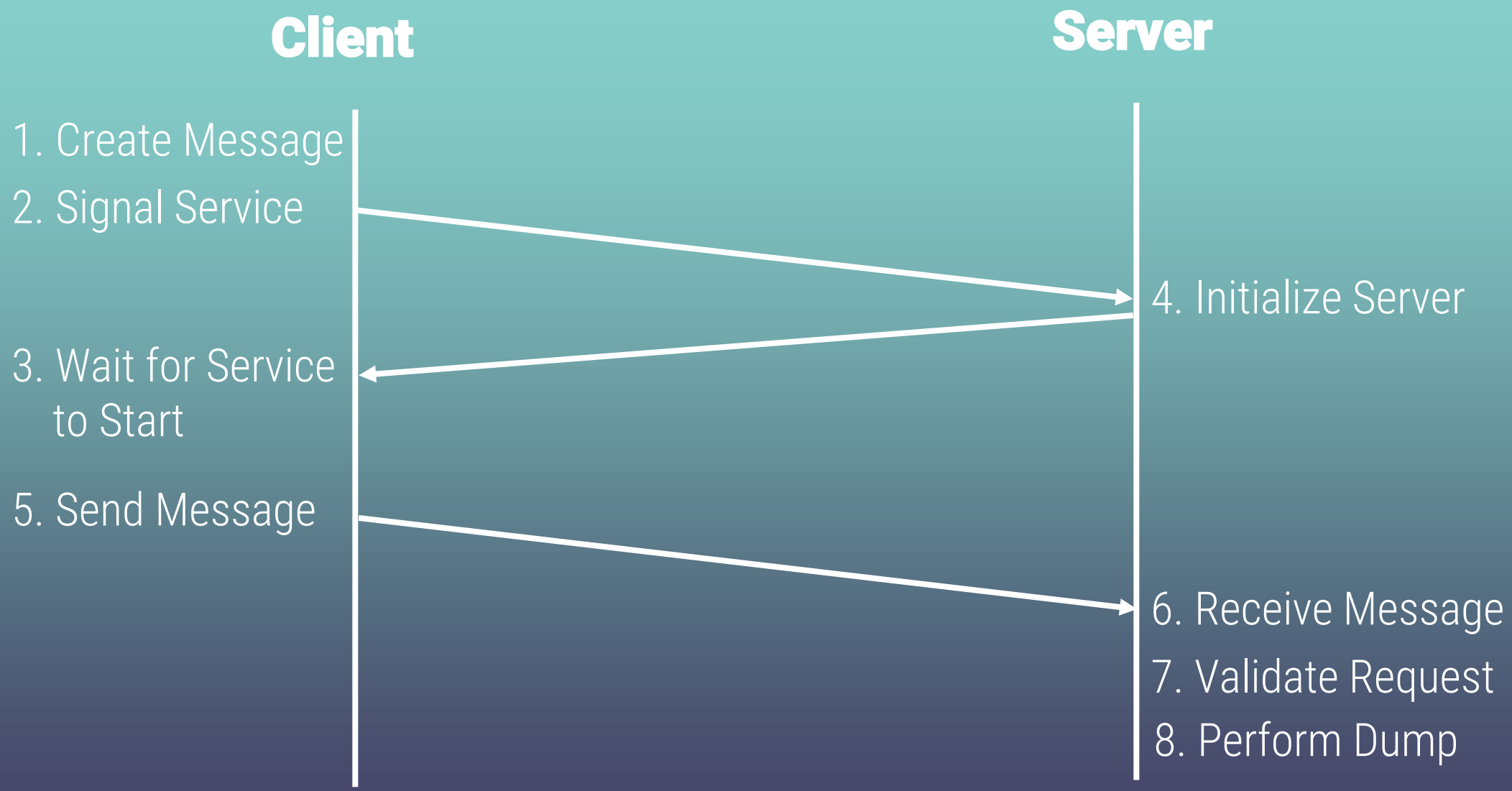
Introducing:
LSASS Shtinkering

LSASS Shtinkering

- New method of dumping LSASS without using a vulnerability
- Abuses the Windows Error Reporting service
- Manually reporting an exception to WER on LSASS will produce a dump without crashing it
- Security products that allow WER to generate memory dumps will be bypassed



The Steps of LSASS Shtinkering



Prerequisites

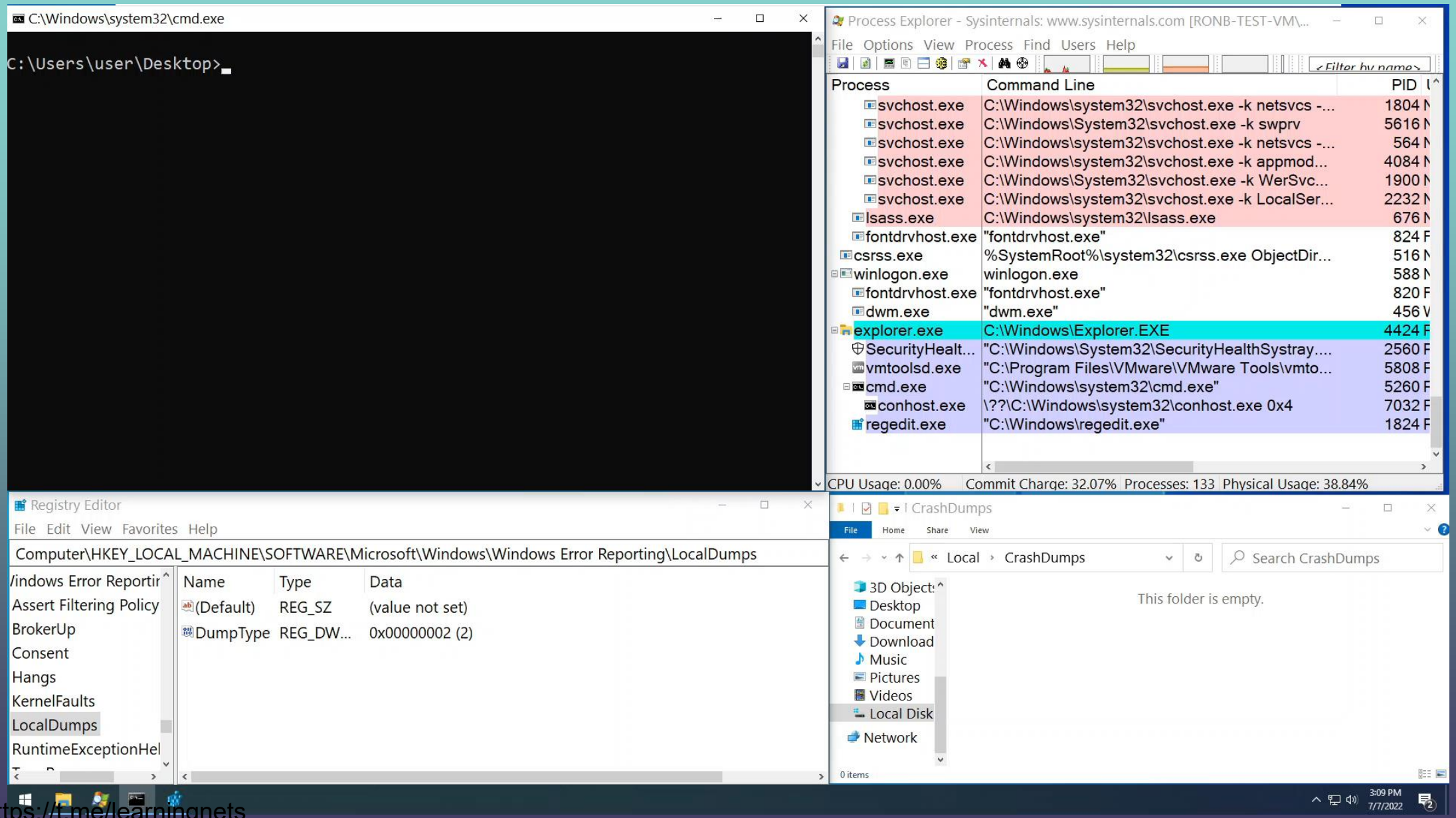
This method requires the following:

- Inheritable process handle to target process with the following access:
  - `PROCESS_VM_READ`
  - `PROCESS_QUERY_LIMITED_INFORMATION`
- Inheritable thread handle a thread in the target process with the following access:
  - `THREAD_QUERY_LIMITED_INFORMATION`
- Registry value “DumpType" set to 2 (Full dump) for the "HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\LocalDumps" key

DumpType	Specify one of the following dump types: • 0: Custom dump • 1: Mini dump • 2: Full dump	REG_DWORD	1

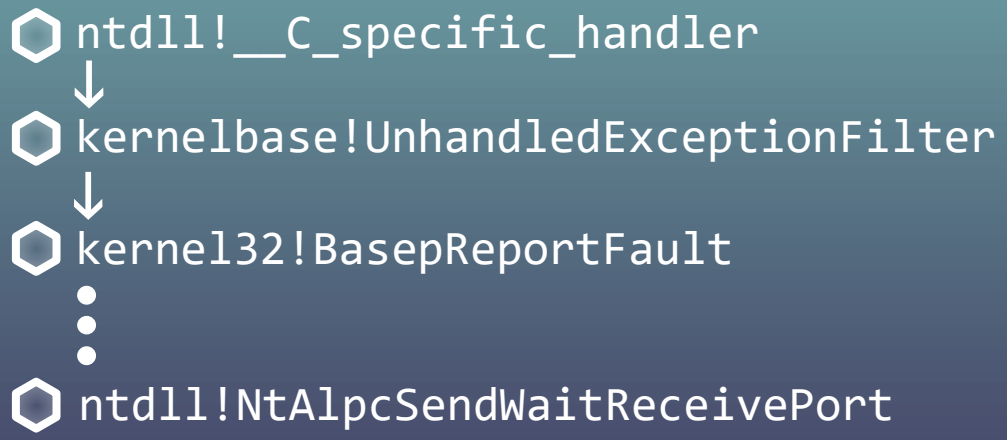
https://docs.microsoft.com/en-us/windows/win32/wer/collecting-user-mode-dumps

Crash Dump Creation



From Exception to Dump File

- The last handler in the Structured Exception Handling stack is `ntdll!__C_specific_handler()`
  - Makes sure that the process exits gracefully instead of hanging
  - Reports the exception details to the WER service
- After reporting an exception to WER, the faulting process will terminate itself
- Exception is reported to the WER service via a call to `ntdll!NtAlpcSendWaitReceivePort()`



01

Memory Dumping Techniques

Overview of known techniques and tools

02

LSASS Shtinkering

Reverse Engineering the WER Client Side

03

LSASS Shtinkering

Reverse Engineering the WER Server Side

04

Detection & Prevention

How to stop the attack

Reverse Engineering WER - Client Side

1. Create a message with WerpReportFaultInternal
2. Send the message with SendMessageToWERService
3. Manually Report an Exception to WER

Creating Message to Send to WER

`WerpReportFaultInternal()` performs the following actions:

1

Create Message

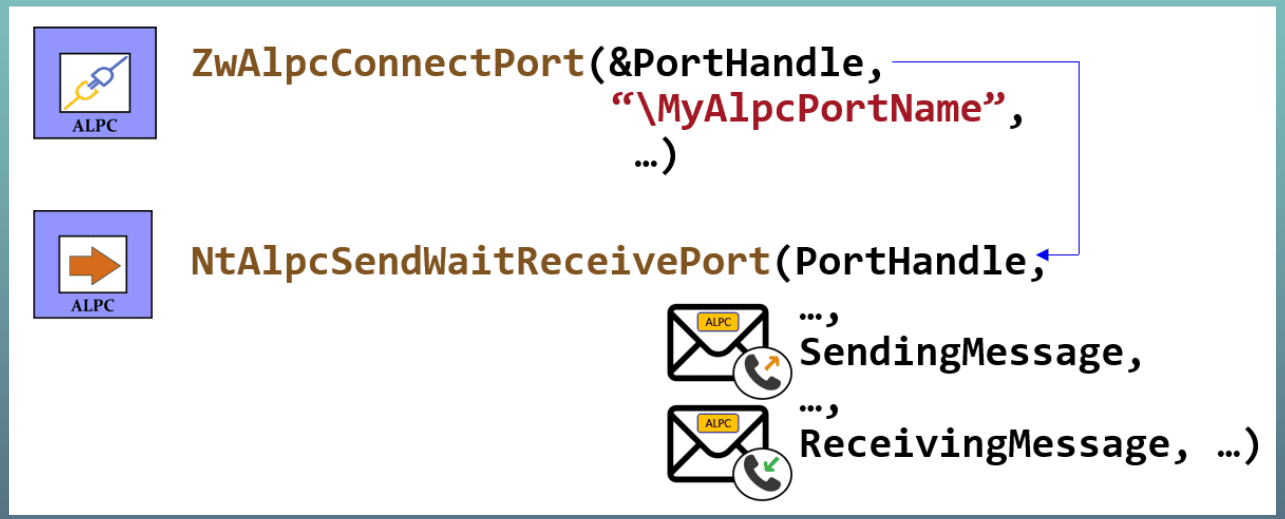
```
hCompletionEvent = CreateEventW(&EventAttributes, 1, 0, 0);
if ( hCompletionEvent )
{
  MappedViewStruct[0] = (int)hCompletionEvent;
  v1 = 1;
  v52 = (void *)1;
}
hRecoveryEvent = CreateEventW(&EventAttributes, 1, 0, 0);
if ( hRecoveryEvent )
{
  MappedViewStruct[v1++] = (int)hRecoveryEvent;
  v52 = (void *)v1;
}
hFileMapping = CreateFileMappingW((HANDLE)0xFFFFFFFF, &EventAttributes, 4u, 0, 0xF8u, 0);
MappedViewStruct[v1] = (int)hFileMapping;
v8 = v1 + 1;
v52 = (void *)v8;
v53 = MapViewOfFile(hFileMapping, 6u, 0, 0, 0);
CurrentProcess = GetCurrentProcess();
if ( DuplicateHandle(CurrentProcess, CurrentProcess, CurrentProcess, &TargeProcesstHandle, 0x1FFFFFu, 1, 0) )
{
  MappedViewStruct[v8++] = (int)TargeProcesstHandle;
  v52 = (void *)v8;
}
v41 = DuplicateHandle(CurrentProcess, CurrentThreadHandle, CurrentProcess, &TargeThreadtHandle, 0x1FFFFFu, 1, 0);
if ( v41 )
{
  MappedViewStruct[v8] = (int)TargeThreadtHandle;
  v52 = (void *)(v8 + 1);
}
CurrentProcessId = GetCurrentProcessId();
v17 = RtlWerpReportException(CurrentProcessId, v29, MappedViewStruct, v32, 0, &v51);
```

Advanced Local Procedure Call

2

Send Message

- Undocumented IPC mechanism
- Used by RPC under the hood
- Two functions of interest on the client side:



Sending the Message to WER

2

Send Message

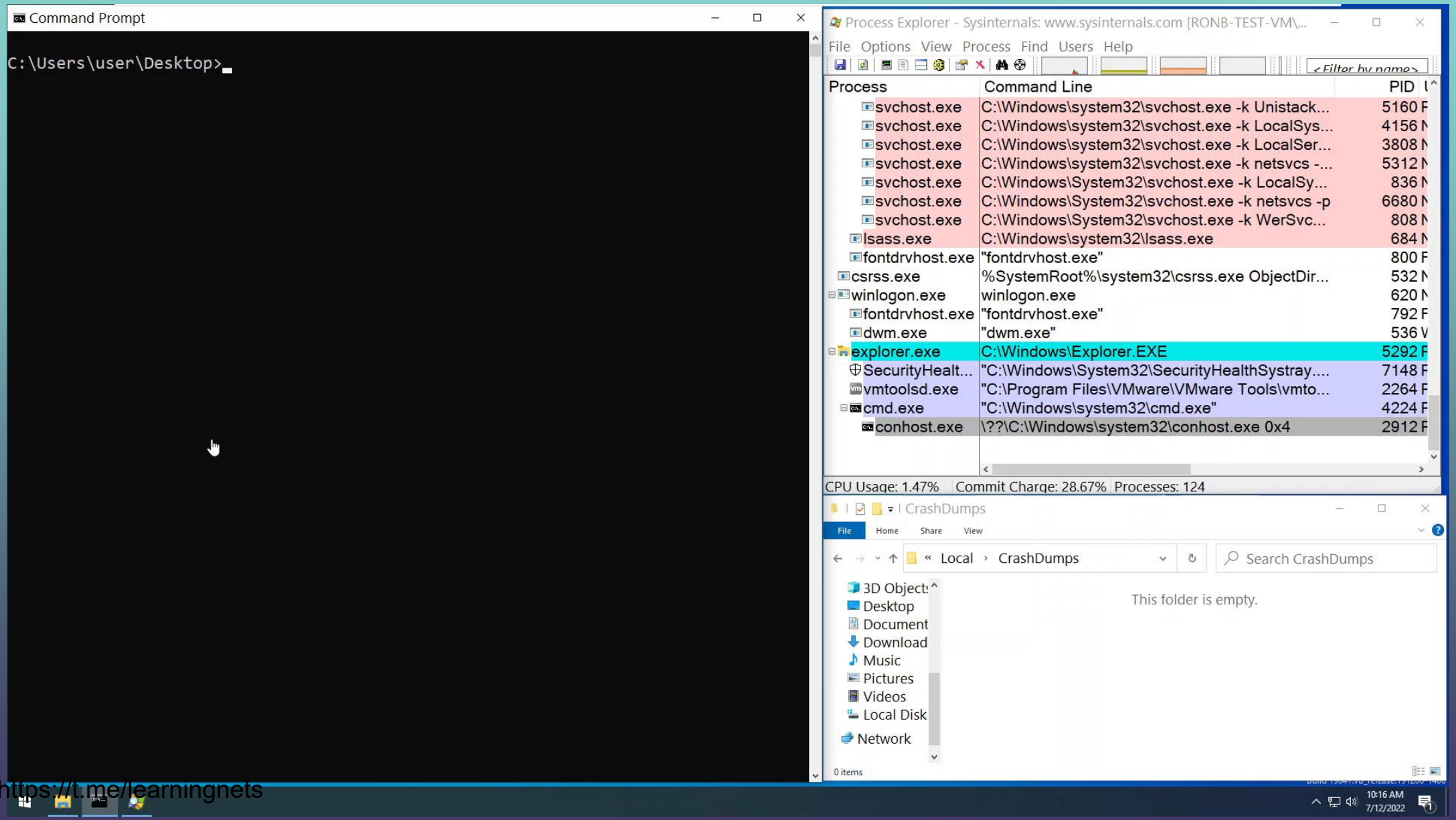
- `SendMessageToWERService()` performs the following actions:

```
ntstatus = SignalStartWerSvc(); // Call NtUpdateWnfStateData with WNF_WER_SERVICE_START
if ( ntstatus >= 0 )
{
  ntstatus = NtQuerySystemInformation(NtQuerySystemInformation, &Systeminformation, 8u, 0);
  if ( ntstatus >= 0 )
  {
    ntstatus = WaitForWerSvc(Systeminformation); // Wait for the event "\\KernelObjects\\SystemErrorPortReady"
    if ( ntstatus >= 0 && ntstatus != STATUS_TIMEOUT)
    {
      RtlInitUnicodeString(&DestinationString, L"\\WindowsErrorReportingServicePort");
      ntstatus = ZwAlpcConnectPort(&Handle, &DestinationString, objectAttributes, portAttributes, 0x20000, v29, 0, 0, 0, 0, v5);
      if ( ntstatus >= 0 && ntstatus != STATUS_TIMEOUT)
      {
        NtAlpcSendWaitReceivePort((int)Handle, 0x20000, v24, 0, v25 , (int)v26 , 0, (int)v27);
      }
    }
  }
}
return ntstatus;
```

Manually Report an Exception to WER

2

Send Message



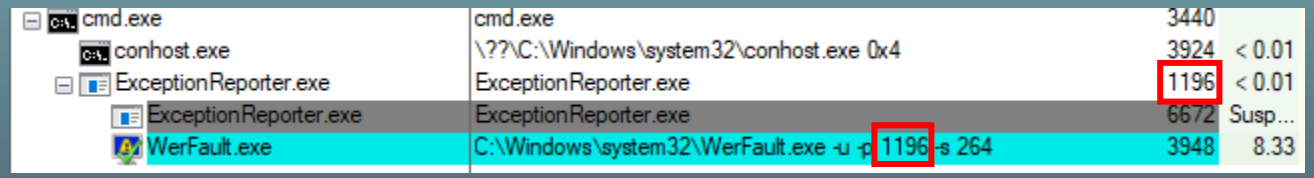
Manually Report an Exception to WER

2

Send Message

Upon a request for a crash dump, WER service performs the following

- Duplicate the file mapping handle into itself and map the view
- Spawn WerFault.exe under WerSvc service with the following parameters:
  WerFault.exe -pss -s <file mapping handle> -p <target process> -ip <source process>
- Spawn WerFault.exe as a child of the sending process via `CreateProcessAsUserW()`
  WerFault.exe -u -p <target process> -s <file mapping handle>
  - Calls `MiniDumpWriteDump()`
  - Report exception to event log



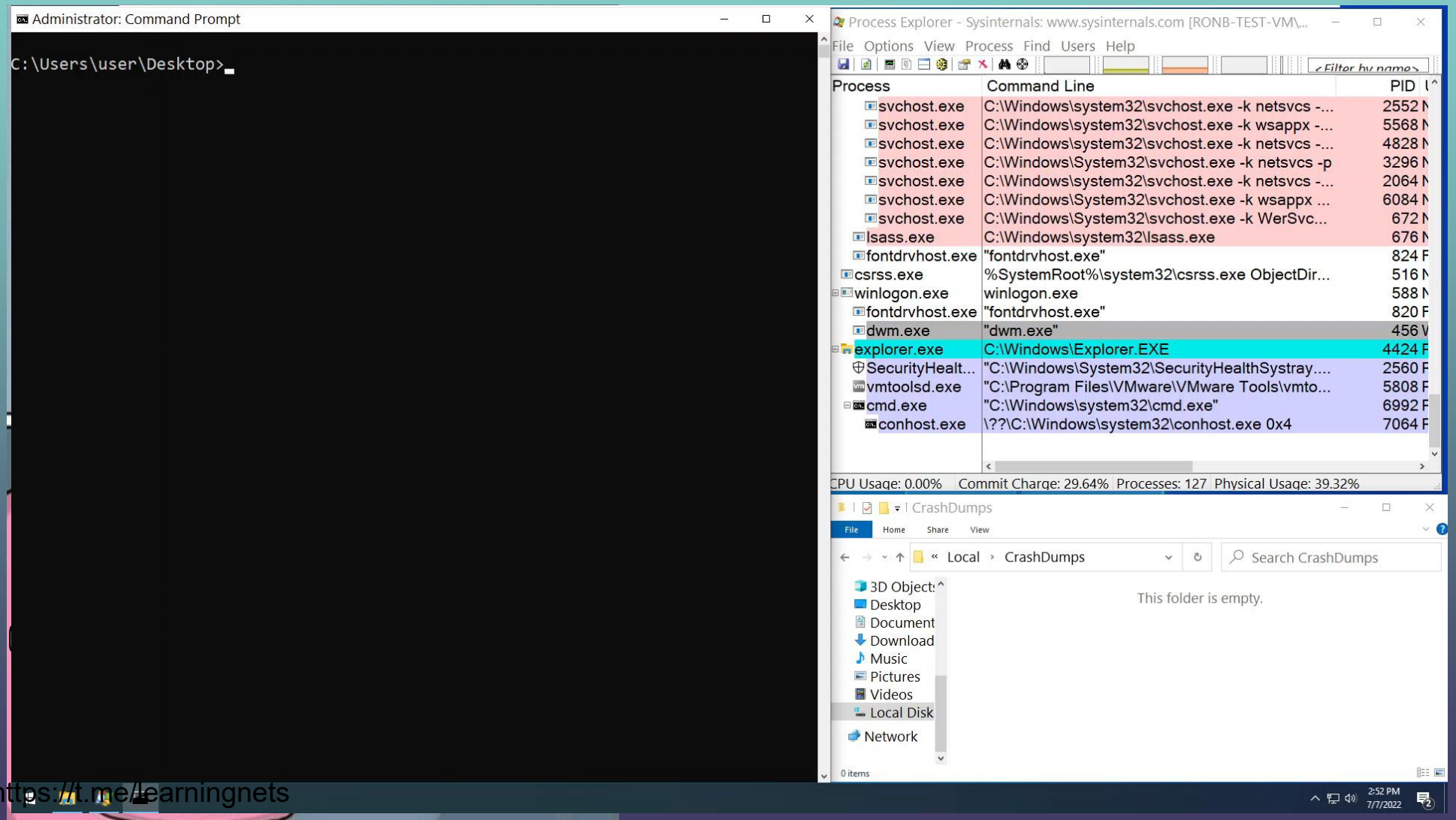
Manually Report an Exception to WER

2

Send Message

The ALPC reply message from WER returns NTSTATUS value of 0x80070005

To understand why, reverse engineering of WerSvc is required



01

Memory Dumping Techniques
Overview of known techniques and tools

02

LSASS Shtinkering
Reverse Engineering the WER Client Side

03

LSASS Shtinkering
Reverse Engineering the WER Server Side

04

Detection & Prevention
How to stop the attack

Reverse Engineering WER - Server Side

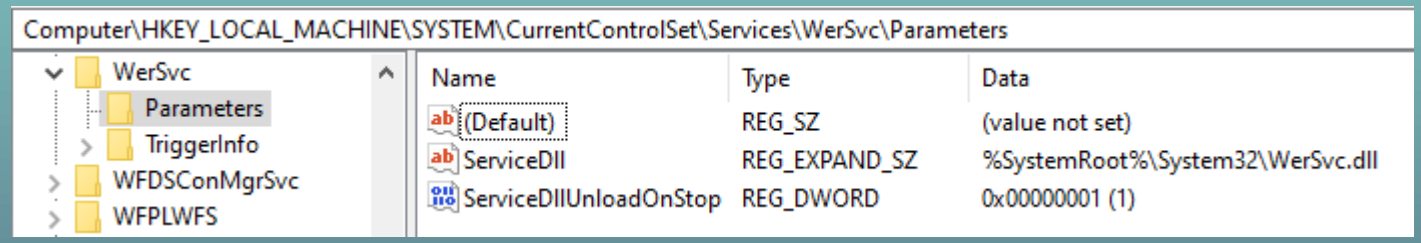
1. WER Service Overview
2. Find Error Code Origin
3. Pass Validation Checks

The WER Service

3

Initialize Service

- Implemented by WerSvc.dll and executed inside svchost.exe
- Service is set to manual start
- Allows errors to be reported when programs stop working
- Allows logs to be generated for diagnostic and repair services



WerSvc ALPC Port Initialization

3

Initialize Service

- `CWerService::_StartLpcServer()`

```
RtlInitUnicodeString(&DestinationString, L"\\WindowsErrorReportingServicePort");
if ( !ConvertStringSecurityDescriptorToSecurityDescriptorW(
        L"D:P(D;OICI;GA;;;NU)(A;OICI;GR;;;AU)(A;OICI;GR;;;BG)(A;OICI;GA;;;S-1-5-80-3299868208-4286319593-1091140620-"
         "3583751967-1732444380)(A;OICI;GR;;;WD)(A;OICI;GR;;;S-1-15-2-1)(A;OICI;GR;;;S-1-15-3-1024-3153509613-96066"
         "6767-3724611135-2725662640-12138253-543910227-1950414635-4190290187)",
        1u,
        &hMem,
        0i64) )
  goto LABEL_23;
ObjectAttributes.Length = 48;
ObjectAttributes.RootDirectory = 0i64;
ObjectAttributes.ObjectName = &DestinationString;
ObjectAttributes.Attributes = 0;
ObjectAttributes.SecurityDescriptor = hMem;
ObjectAttributes.SecurityQualityOfService = 0i64;
memset_0(v23, 0, 0x48ui64);
v23[0] = 0x20000;
v24 = 1400i64;
v25 = 0i64;
v26 = 89600i64;
v16 = NtAlpcCreatePort((char *)lpCriticalSection + 368, &ObjectAttributes, v23);
if ( v16 >= 0 )
{
  if ( *((_QWORD *)lpCriticalSection + 47) )
    MicrosoftTelemetryAssertTriggeredNoArgs(v15);
  v17 = CreateThread(
          0i64,
          0i64,
          (LPTHREAD_START_ROUTINE)CWerService::StaticLpcServerThread,
          lpCriticalSection,
          0,
          &ThreadId);
```

Find Error Code Origin in WerSvc.dll

4

Validate Request

- References for the error code “80070005” where found in WerSvc.dll:

Occurrences of: 80070005

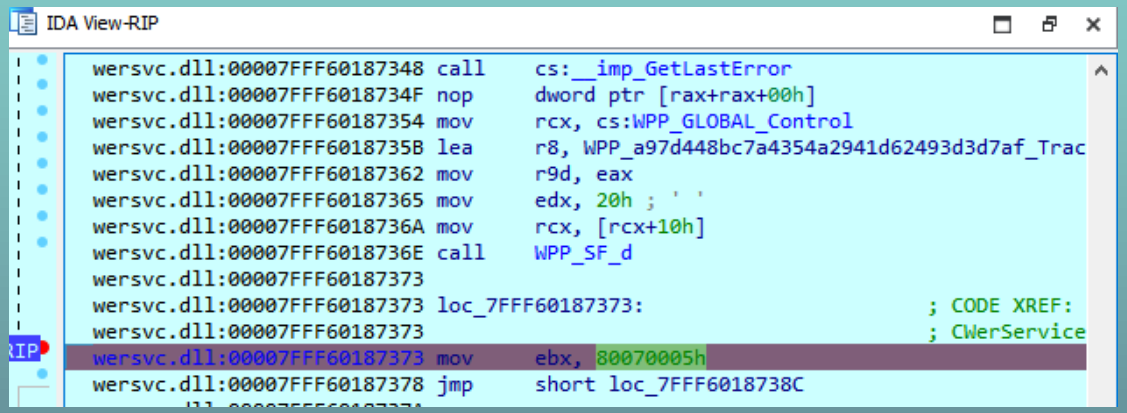
Address	Function	Instruction
.text:00007FFE06C87393	?CheckIfSystemConnectingToPort@CWerService@@AEAAJPEAU_WERSVC_MSG@@@Z	; CWerService::CheckIfSystemConnectingToPort(_WERSVC_MSG *)+246↑j
.text:00007FFE06C8765F	?CheckIfValidPortMessage@CWerService@@AEAAJPEAU_WERSVC_MSG@@@Z	mov eax, 80070005h
.text:00007FFE06C8A953	?SvcReportHang@CWerService@@AEAAJPEAU_WERSVC_MSG@@0@Z	; CWerService::SvcReportHang(_WERSVC_MSG *,_WERSVC_MSG *)+B6↑j
.text:00007FFE06C8AD0B	?SvcReportCrash@CWerService@@AEAAJPEAU_WERSVC_MSG@@0@Z	mov dword ptr [rbx+2Ch], 80070005h
.text:00007FFE06C8F9CD	?NonElevatedProcessStart@@YAJPEAX0PEAPEAX@Z	mov edi, 80070005h
.text:00007FFE06C94BAD	?_CheckIfOKToReport@CHangrepServer@@AEAAJPEAX0KKPEAPEAX1@Z	mov eax, 80070005h
.text:00007FFE06C958DB	?Cancel@CHangrepServer@@QEAAJPEAXK@Z	mov ebx, 80070005h
.text:00007FFE06CA20DB	?UtilVerifyFilePath@@YAJPEBGPEAX@Z	cmp eax, 80070005h
.text:00007FFE06CA96D7	?GetProcessAppId@CallerIdentity@@YAJPEAXPEAPEAG@Z	cmp edi, 80070005h

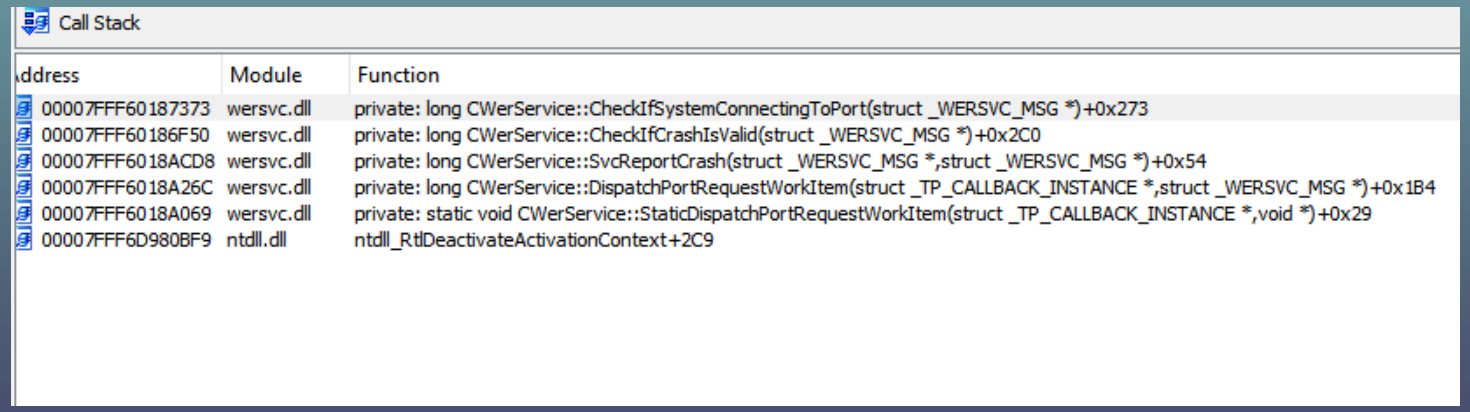
Find Error Code Origin in WerSvc.dll

4

Validate Request

- Placed breakpoint in each reference
- The code stopped inside `CheckIfSystemConnectingToPort()`





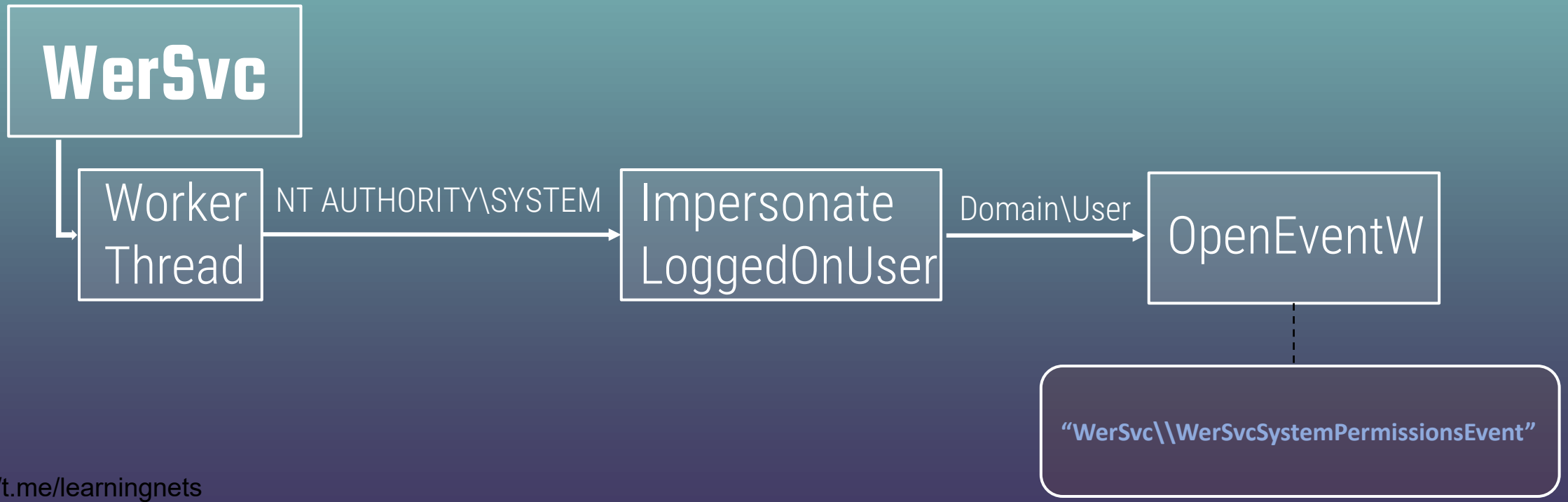
Opening the Event

4

Validate Request

`CheckIfSystemConnectingToPort()`

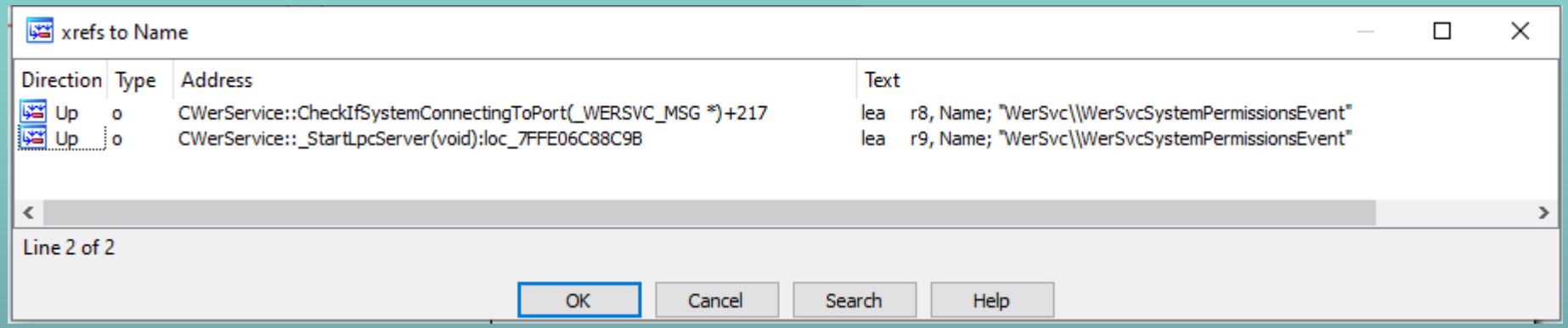
- Impersonates the process that sent the request via `ImpersonateLoggedOnUser()`
- Attempt is made to open the event "WerSvc\WerSvcSystemPermissionsEvent"
- `OpenEvent()` fails with ERROR_ACCESS_DENIED
- Function returns **0x80070005**



Tracing Back Event Creation

4

Validate Request




```
    if ( !ConvertStringSecurityDescriptorToSecurityDescriptorW(
            L"D:(A;OICI;GR;;;SY)",              // Allow "NT AUTHORITY\SYSTEM" GENERIC_READ
            1u,
            &SecurityDescriptor.lpSecurityDescriptor,
            0i64) )
    {
LABEL_23:
      LastError = GetLastError();
      v6 = (unsigned __int16)LastError | 0x80070000;
      if ( LastError <= 0 )
        v6 = LastError;
      if ( v6 >= 0 )
        v6 = -2147467259;
      goto LABEL_51;
    }
    SecurityDescriptor.nLength = 24;
    SecurityDescriptor.bInheritHandle = 0;
    v12 = CreateEventW(&SecurityDescriptor, 0, 0, L"WerSvc\\WerSvcSystemPermissionsEvent");
```

?

Tracing Back Event Creation

4

Validate Request

Address	Length	Type	String
.rdata:00007FF...	0000000E	C (16 bits) - UTF-16LE	wersvc
.rdata:00007FF...	0000000E	C (16 bits) - UTF-16LE	WerSvc
.rdata:00007FF...	00000048	C (16 bits) - UTF-16LE	WerSvc\\WerSvcSystemPermissionsEvent
.rdata:00007FF...	00000030	C (16 bits) - UTF-16LE	WerSvcNameSpaceBoundary
.rdata:00007FF...	00000028	C (16 bits) - UTF-16LE	WerSvcKernelMsgDone
.rdata:00007FF...	0000000B	C	wersvc.dll
.rdata:00007FF...	00000056	C (16 bits) - UTF-16LE	WerSvc\\WerSvcNonElevationInfoSectionName%d
.rdata:00007FF...	0000003E	C	onecore\\windows\\feedback\\core\\wersvc\\lib\\reflectionserver.cpp

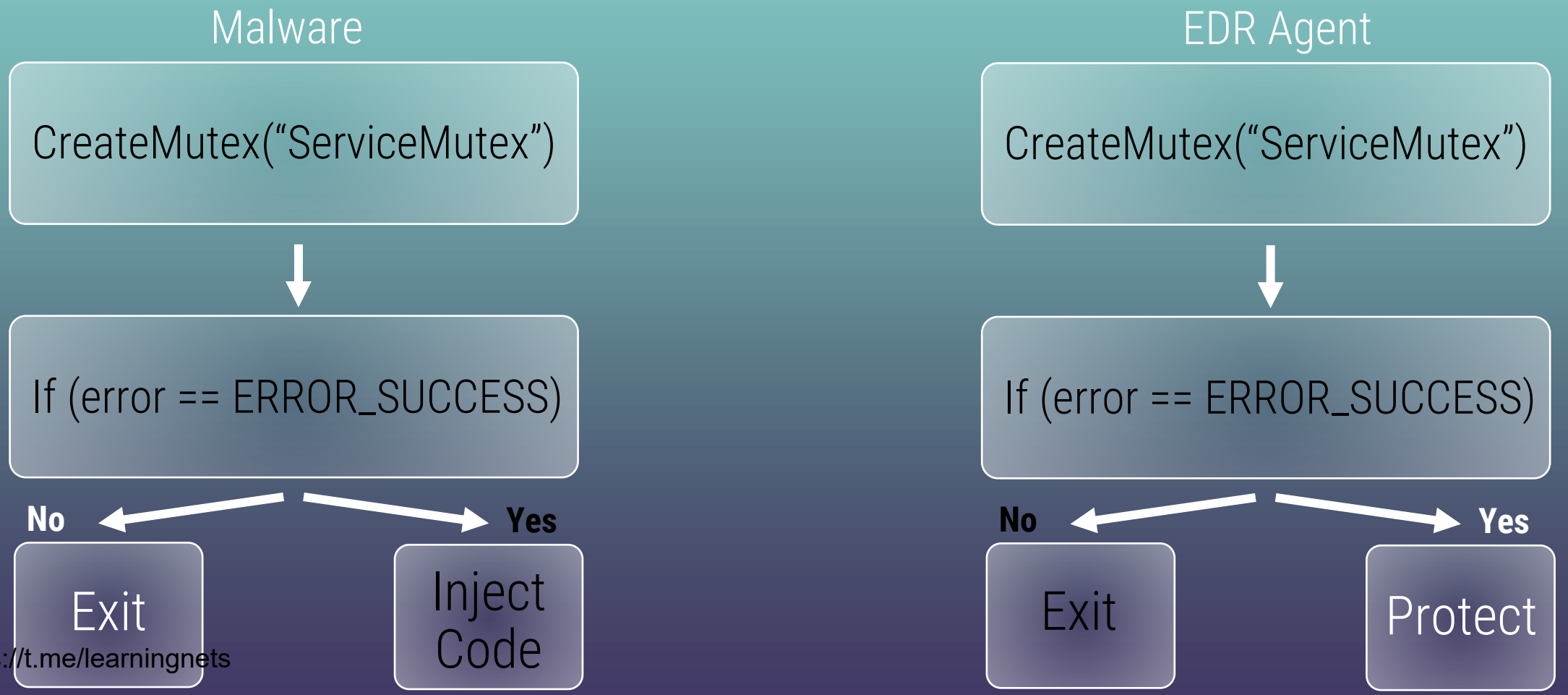
```
AliasPrefix:                            ; DATA XREF: CWerService::QueryServiceS
                                        ; CWerService::_CreatePrivateNamespace(
        text "UTF-16LE", 'WerSvc',0
        align 8
```

xrefs to AliasPrefix

Directio	Type	Address	Text
Up	o	CWerService::QueryServiceStartType(ulong *)+6D	lea rdx, AliasPrefix; "WerSvc"
Up	o	CWerService::_CreatePrivateNamespace(void):loc_7FFE0...	lea rdx, AliasPrefix; "WerSvc"
Up	o	CWerService::_CreatePrivateNamespace(void)+27C	lea r8, AliasPrefix; "WerSvc"
Up	o	CWerService::_CreatePrivateNamespace(void)+2C1	lea rdx, AliasPrefix; "WerSvc"

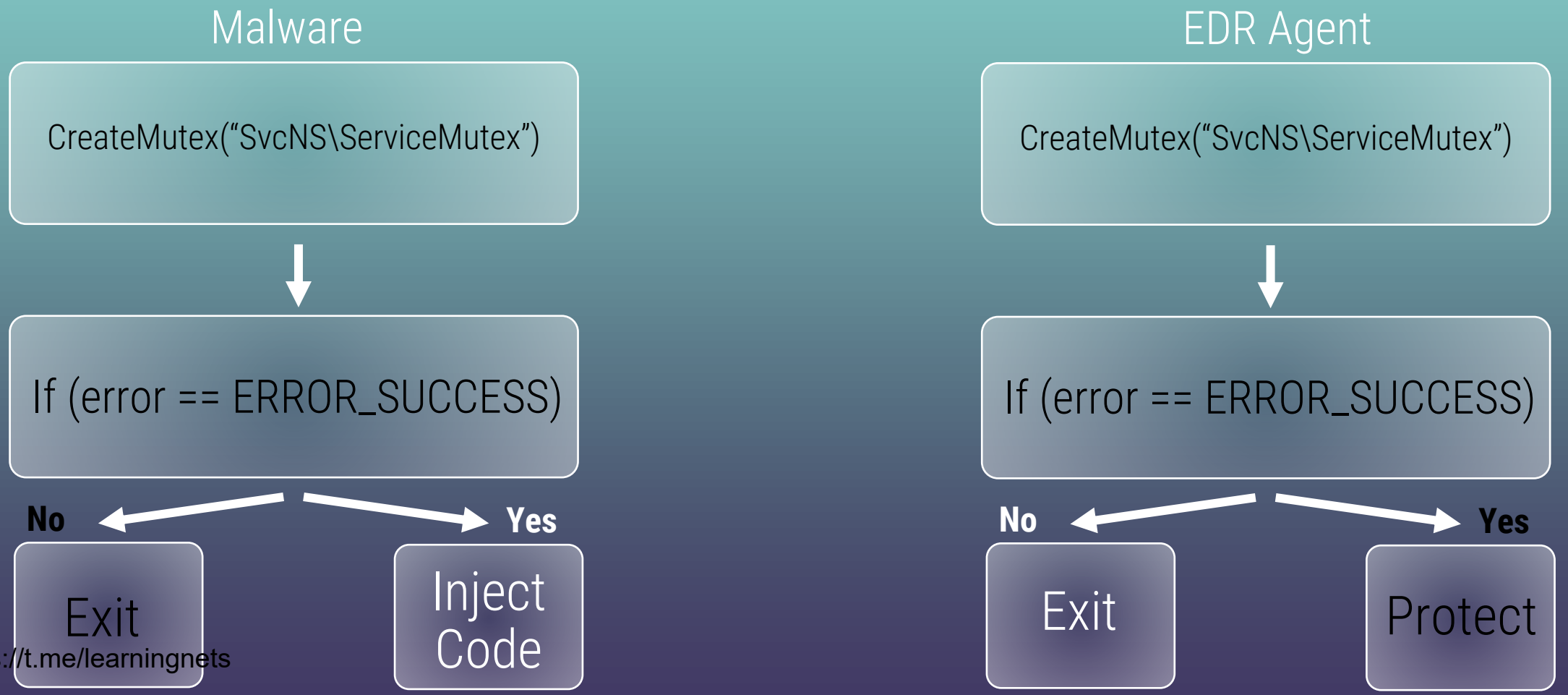
Private Namespaces and Boundaries

- Private namespaces and boundary descriptors protect from a *squatting attack:*
- “DoS attack where a program interferes with another program through the use of shared synchronization objects in an unwanted or unexpected way”



Private Namespaces and Boundaries

- Private namespace is like a directory for kernel objects that is protected by a boundary descriptor
- Descriptors contain SIDs describing which users and groups are allowed to create objects in the directory
- Namespace is identified by both its name and boundary descriptor
- Different namespaces can have identical names if they have differing boundary descriptors



Private Namespaces and Boundaries

- Private namespaces protect named objects from access by non-approved SIDs
- Approved SIDs are set for boundary descriptor
- Boundary Descriptor is created with `CreateBoundaryDescriptor()`
- Approved SIDs are added to boundry descriptor via `AddSIDToBoundaryDescriptor()`
- Namespace is created via `CreatePrivateNamespace()`
  The boundary descriptor is sent as a parameter

WerSvc Initialization

- The following actions are performed upon service initialization:
- `CWerService::_CreatePrivateNamespace()`
  - Creates a boundary descriptor with the SID of the service
  - Creates the "WerSvc" private namepace with the boundary descriptor
  - Events can be created under this namespace only with the WerSvc SID
- Event "WerSvc\WerSvcSystemPermissionsEvent" is created
  - "WerSvcSystemPermissionsEvent" exists under the namespace "WerSvc"
  - Can only be accessed by SYSTEM due to the security descriptor

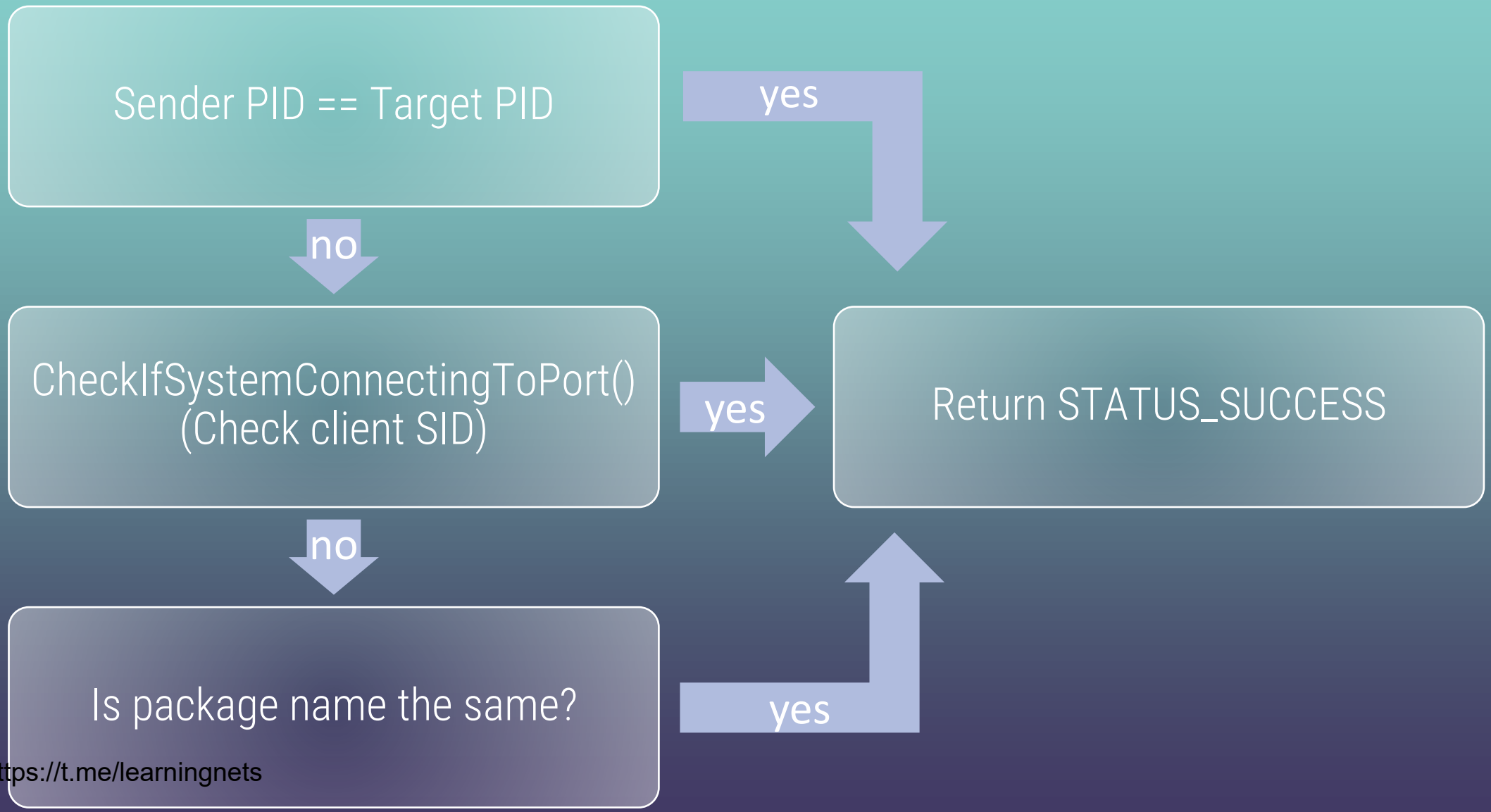
```
HANDLE hBoundaryDescriptor = RtlCreateBoundaryDescriptor(L"WerSvcNameSpaceBoundary", 0);
RtlCreateServiceSid("WerSvc", SidBuffer, BufferSize);
RtlAddSIDToBoundaryDescriptor(SidBuffer, hBoundaryDescriptor);
CreatePrivateNamespaceW(hBoundaryDescriptor, L"WerSvc");
…
// Allow GENERIC_READ to “NT AUTHORITY\SYSTEM”
ConvertStringSecurityDescriptorToSecurityDescriptorW("D:(A;OICI;GR;;;SY)", &SecurityDescriptor);
CreateEventW(&SecurityDescriptor, L"WerSvc\\WerSvcSystemPermissionsEvent");
```

Passing Validation Checks

4

Validate Request

Checks performed by `CWerService::CheckIfCrashIsValid()`

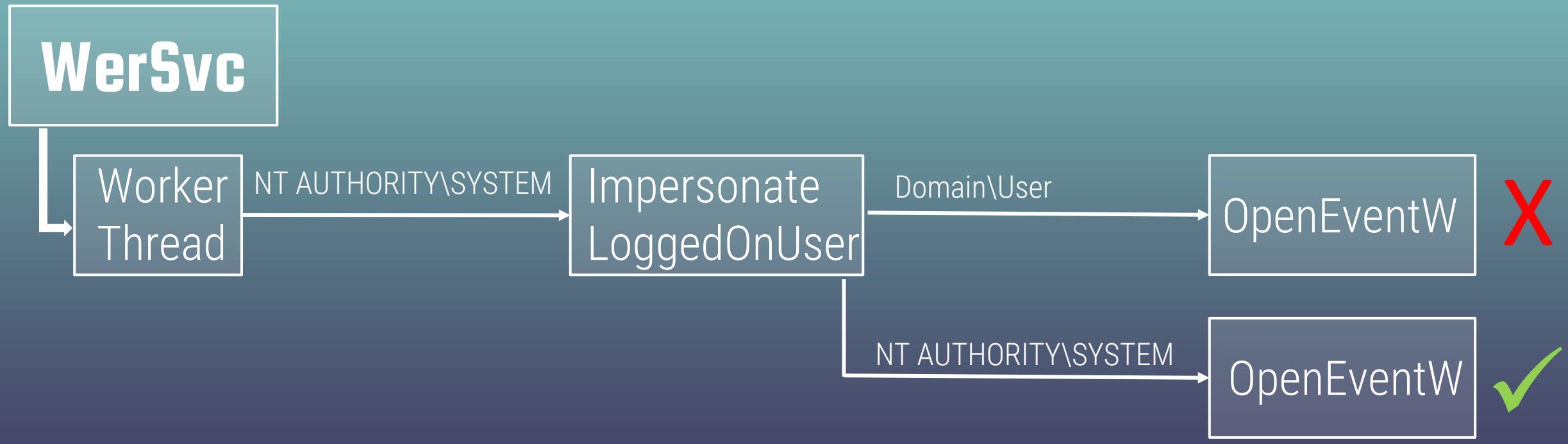


Opening the Event

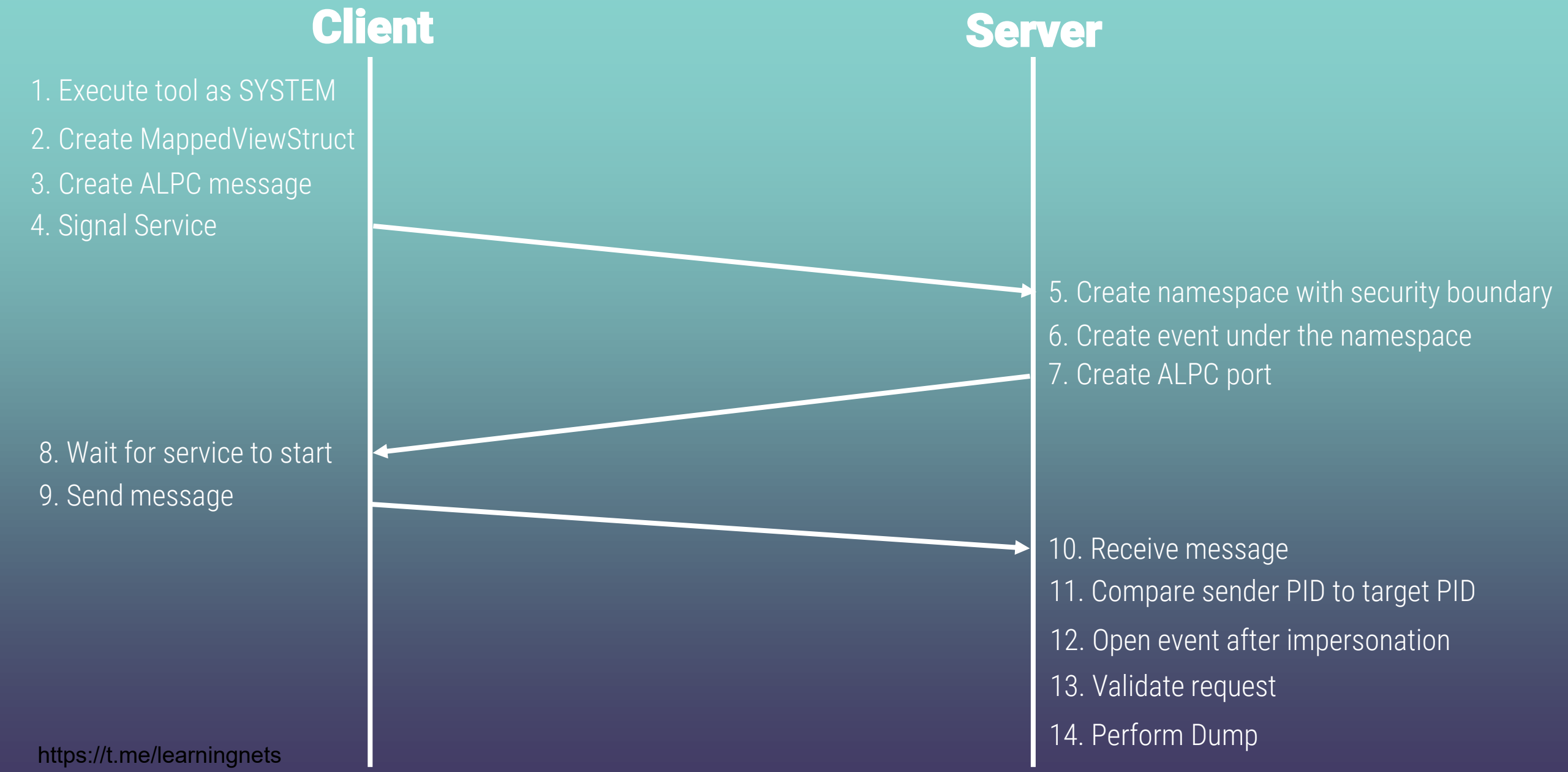
4

Validate Request

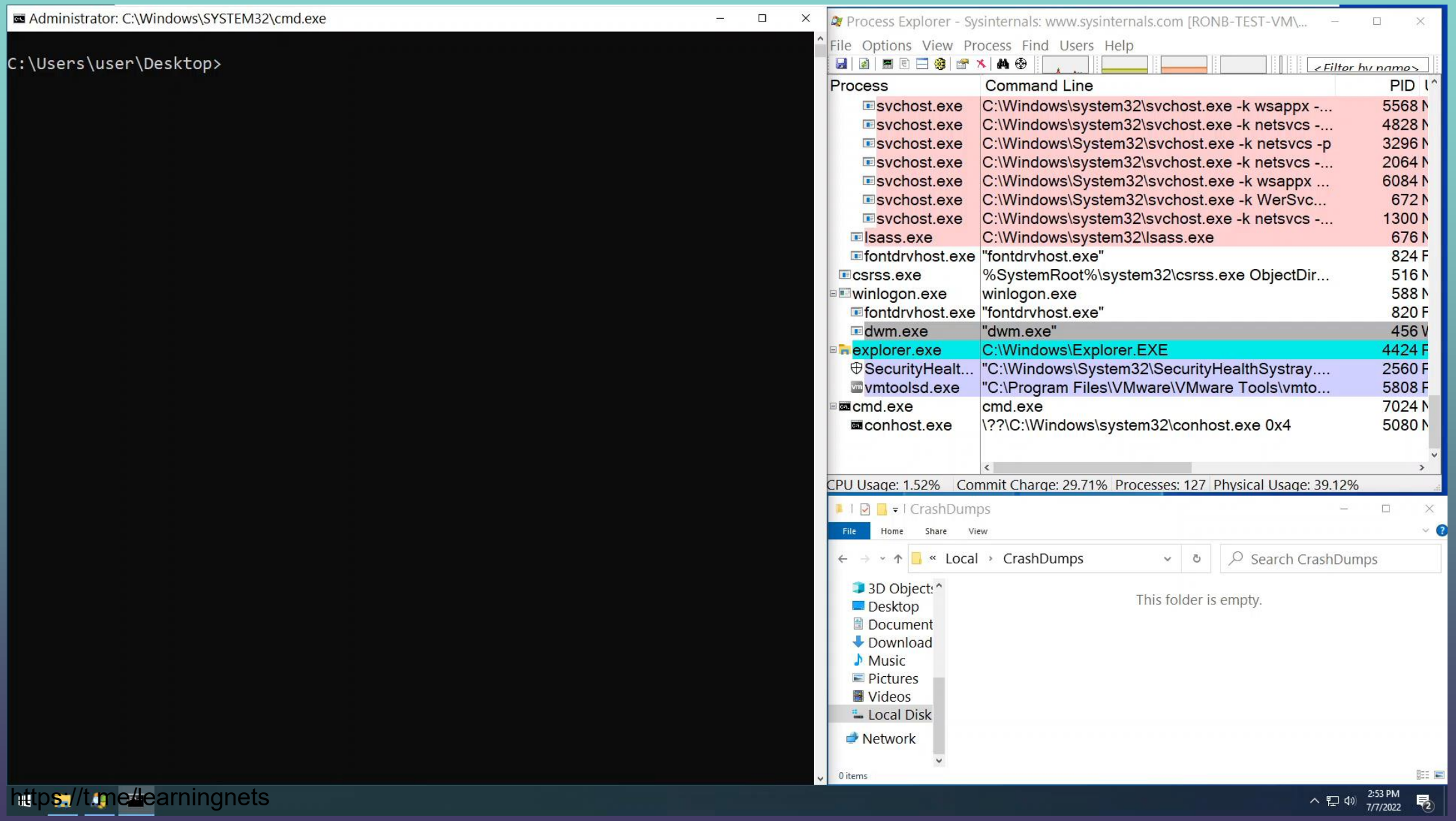
- `CheckIfSystemConnectingToPort()` checks if the sender runs as “NT AUTHORITY\SYSTEM”
- Sender doesn’t have same SID as WER
- The event fails to open
- Solution - execute the sender as “NT AUTHORITY\SYSTEM”



Recap



Demonstration



https://t.me/learningnets

01

Memory Dumping Techniques

Overview of known techniques and tools

02

LSASS Shtinkering

Reverse Engineering the WER Client Side

03

LSASS Shtinkering

Reverse Engineering the WER Server Side

04

Detection & Prevention

How to stop the attack

Remaining Artifacts

Event log

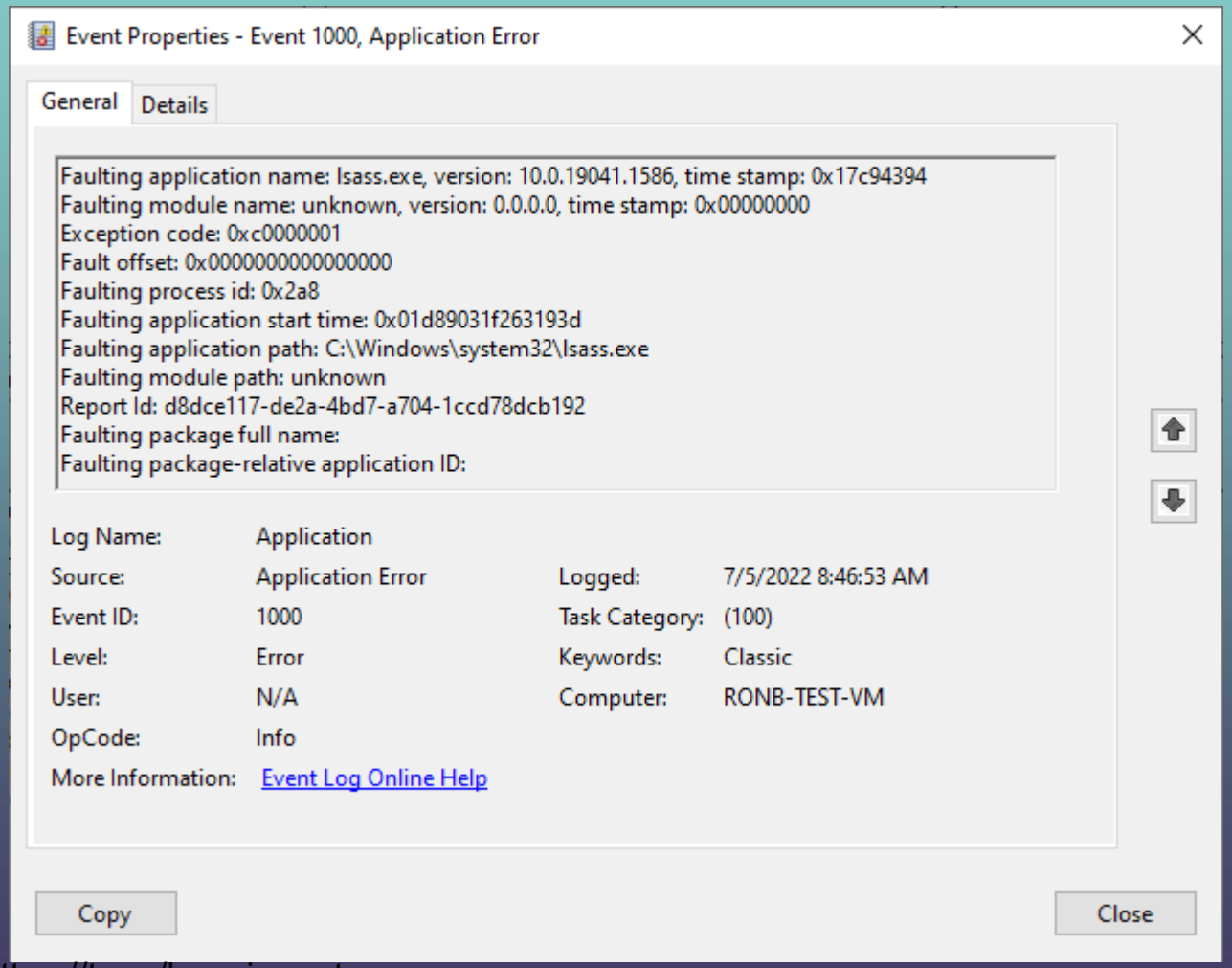
Dump file

WER Report Archive

WerFault command-line

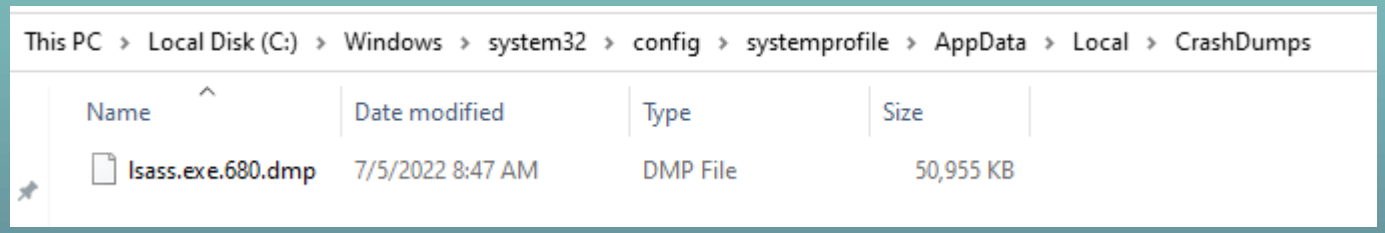
Event Log

- Event ID 1000 is generated under “Windows Logs\Application”
- Event doesn’t specify the sender process



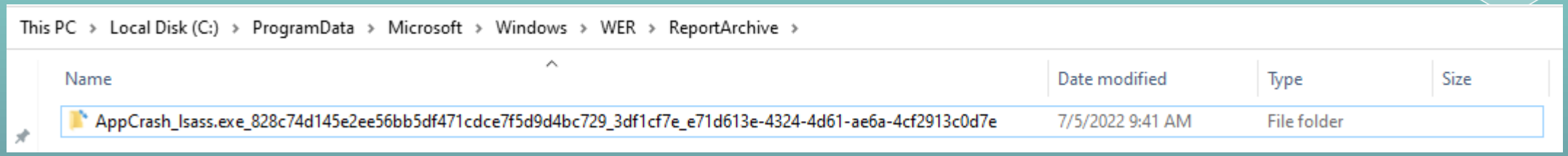
Dump File

- Dump files will be written to %LocalAppData%\CrashDumps
- For processes running as “NT AUTHORITY\SYSTEM”, the path is: C:\Windows\system32\config\systemprofile\AppData\Local\CrashDumps



WER Report Archive

- Archive located at:
  C:\ProgramData\Microsoft\Windows\WER\ReportArchive
- Each directory contains “Report.Wer” – log file that doesn’t specify the sender process

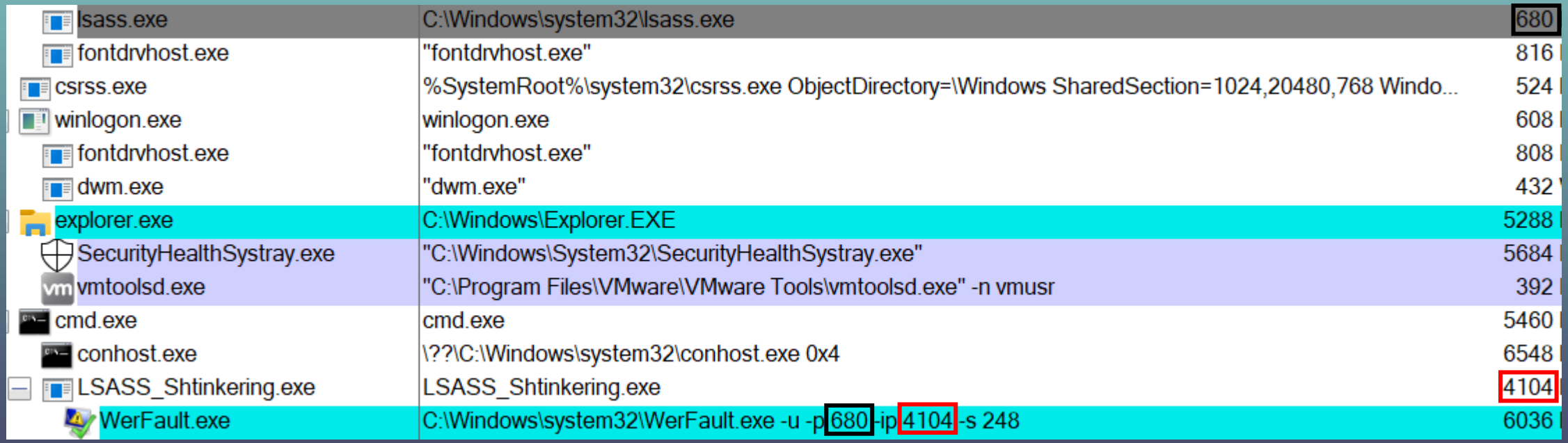


Report.wer

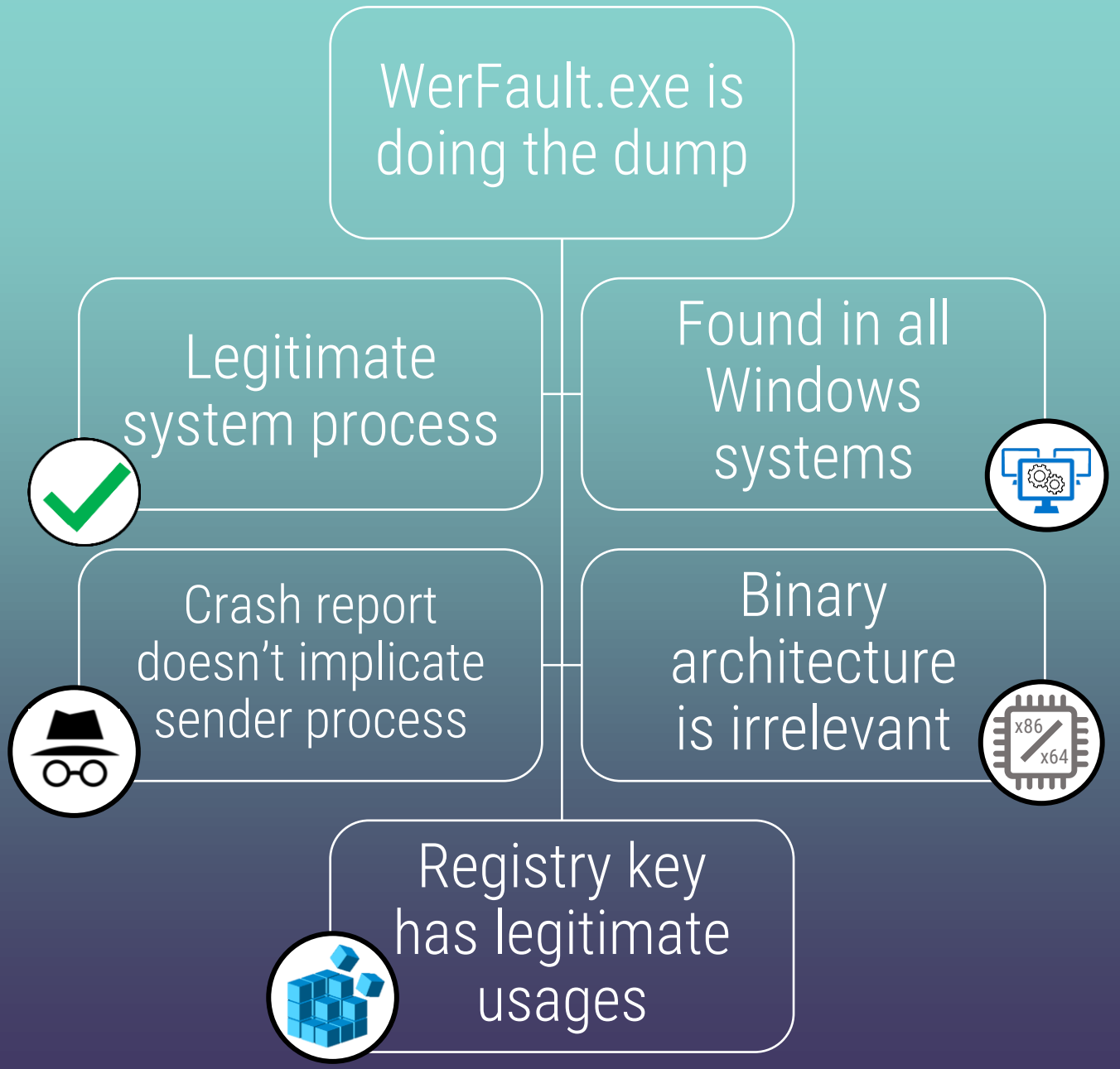
```
1  Version=1
2  EventType=CriticalProcessFault2
3  EventTime=133014760208937570
4  ReportType=2
5  Consent=1
6  UploadTime=133014760292061972
7  ReportStatus=268435456
8  ReportIdentifier=408016a3-ef54-4ff1
9  IntegratorReportIdentifier=c26285f0
10 Wow64Host=34404
11 NsAppName=lsass.exe
12 OriginalFilename=lsass.exe
```

WerFault Command Line

- *WerFault.exe -u -p <target process> -ip <source process> -s <file mapping handle>*
- If the source process is not equal to the target process and the target process is LSASS then this is an indication of this technique



Advantages



Suggested Actions

- Application event ID 1000 (exception reported by WER) which is not followed by a termination of LSASS
- WerFault command line:
  WerFault.exe -u -p <target process> -ip <source process> -s <file mapping handle>
  Source process is not equal to the target process and the target process equals LSASS PID
- Use API monitoring to look for ALPC messages sent to WER with the LSASS PID
- Setting LSASS as PPL prevents from opening a handle with PROCESS_VM_READ

Further Research

- Other Message types
  - What do they cause WerSvc to do? Can it be exploited?
- Undocumented struct might change in future releases

References

- Windows Internals 6th Edition Part 1
- Windows Via C/C++ 5th Edition
- https://docs.microsoft.com/en-us/windows/win32/wer/collecting-user-mode-dumps
- https://flylib.com/books/en/2.294.1.98/1/
- https://www.wikiwand.com/en/Squatting_attack
- https://www.cybereason.com/blog/threat-analysis-report-from-shatak-emails-to-the-conti-ransomware
- https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2022/06/23093553/Common-TTPs-of-the-modern-ransomware_low-res.pdf
- https://slidesgo.com/theme/tech-startup

THANKS

Do you have any questions?

Contact us:

@asaf_gilboa
@RonB_Y

https://www.linkedin.com/in/asaf-gilboa/
https://www.linkedin.com/in/ron-ben-yizhak-039699156/