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 in the Wild

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

Kaspersky Crimeware Reports: Common TTPs of modern ransomware groups
Credentials Dumping Flow

1. Logged on user credentials stored in process memory
2. Dump process memory to disk
3. Read memory dump and parse credentials (e.g. Mimikatz)
4. Use new credentials for lateral movement
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

```c
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

- **ComSvc.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`
Silent Process Exit

New LSASS Dumping Method via SilentProcessExit
depinstinct.com/2021/02/16/Isa...

6:43 PM · Feb 23, 2021

197 Retweets 2 Quote Tweets 395 Likes

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

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

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

**Client**
1. Create Message
2. Signal Service
3. Wait for Service to Start
4. Initialize Server
5. Send Message

**Server**
6. Receive Message
7. Validate Request
8. Perform Dump
Prerequisites

This method requires the following:

• Inheritable process handle to target process with the following access:
  • \texttt{PROCESS\_VM\_READ}
  • \texttt{PROCESS\_QUERY\_LIMITED\_INFORMATION}

• Inheritable thread handle a thread in the target process with the following access:
  • \texttt{THREAD\_QUERY\_LIMITED\_INFORMATION}

• Registry value “DumpType” set to 2 (Full dump) for the \texttt{HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\LocalDumps} key

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()`
Memory Dumping Techniques
Overview of known techniques and tools

LSASS Shtinkering
Reverse Engineering the WER Client Side

LSASS Shtinkering
Reverse Engineering the WER Server Side

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:

```c
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, &TargetProcessHandle, 0x1FFFFFu, 1, 0))
{
    MappedViewStruct[v8++] = (int)TargetProcessHandle;
    v52 = (void *)(v8);
}
v41 = DuplicateHandle(CurrentProcess, CurrentThreadHandle, CurrentProcess, &TargetThreadHandle, 0x1FFFFFu, 1, 0);
if (v41)
{
    MappedViewStruct[v8] = (int)TargetThreadHandle;
    v52 = (void *)(v8 + 1);
}
CurrentProcessId = GetCurrentProcessId();
v17 = RtlWerpReportException(CurrentProcessId, v29, MappedViewStruct, v32, 0, &v51);
```
Advanced Local Procedure Call

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

```
ZwAlpcConnectPort(&PortHandle,
    "\MyAlpcPortName",
    ...
)
```

```
NtAlpcSendWaitReceivePort(PortHandle,
    ..., SendingMessage,
    ..., ReceivingMessage, ...
)
```
Sending the Message to WER

- `SendMessageToWERService()` performs the following actions:

```c
ntstatus = NtSignalStartWerSvc(); // 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, 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
Manually Report an Exception to WER

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:
  \texttt{WerFault.exe -pss -s <file mapping handle> -p <target process> -ip <source process>}
- Spawn WerFault.exe as a child of the sending process via \texttt{CreateProcessAsUserW()}:
  \texttt{WerFault.exe -u -p <target process> -s <file mapping handle>}
  - Calls \texttt{MiniDumpWriteDump()}
  - Report exception to event log
Manually Report an Exception to WER

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

- 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

- CWerService::_StartLpcServer()
Find Error Code Origin in WerSvc.dll

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text00007FED0C8753</td>
<td>?CheckIfSystemConnectingToPort@AEAAJPEAU_WERSVC_MSG@B@Z</td>
<td>; CWes:CheckIfSystemConnectingToPort(WERSVC_MSG*)+2451j</td>
</tr>
<tr>
<td>.text00007FED0C875F</td>
<td>?CheckIfYanPortMessage@CWes:Service@B@AEAAJPEAU_WERSVC_MSG@B@Z</td>
<td>mov ecx, 80070005h</td>
</tr>
<tr>
<td>.text00007FED0C8A953</td>
<td>?SvcReportHang@CWes:Service@B@AEAAJPEAU_WERSVC_MSG@B@Z</td>
<td>mov ecx, 80070005h</td>
</tr>
<tr>
<td>.text00007FED0C8A953</td>
<td>?SvcReportCrash@CWes:Service@B@AEAAJPEAU_WERSVC_MSG@B@Z</td>
<td>mov edx, 80070005h</td>
</tr>
<tr>
<td>.text00007FED0C8F3CD</td>
<td>_NonElevatedProcessStart@@YAJPEAKPEAPEAN@Z</td>
<td>mov edx, 80070005h</td>
</tr>
<tr>
<td>.text00007FED0C91AD</td>
<td>?_CheckIfOKToReport@ChangregServer@AEAAJPEAU_WERSVC_MSG@B@Z</td>
<td>mov eax, 80070005h</td>
</tr>
<tr>
<td>.text00007FED0C91DB</td>
<td>?Cancel@ChangregServer@AEAAJPEAU_WERSVC_MSG@B@Z</td>
<td>mov ebx, 80070009h</td>
</tr>
<tr>
<td>.text00007FED0CAD8</td>
<td>_UtilVerifyMacrath@@YAJPEAKPEAPEAN@Z</td>
<td>mov ebx, 80070009h</td>
</tr>
<tr>
<td>.text00007FED0CAD8</td>
<td>?GetProcessAppId@CallerIdentity@YAJPEAKPEAPEAN@Z</td>
<td>cmp ebx, 80070009h</td>
</tr>
</tbody>
</table>
Find Error Code Origin in WerSvc.dll

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

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

Validate Request

```c
if (!ConvertStringSecurityDescriptorToSecurityDescriptorW(
    L"D:(A:OICI;GR;;SY)", // Allow "NT AUTHORITY\SYSTEM" GENERIC_READ
    &securityDescriptor.lpSecurityDescriptor, 0;64) )
{
    LABEL_23:
    LastError = GetLastError();
    v6 = (unsigned __int16)LastError | 0x80070000;
    if ( LastError <= 0 )
        v6 = LastError;
    if ( v6 >= 0 )
        v6 = -2147483648;
    goto LABEL_51;
}

SecurityDescriptor.mLength = 24;
SecurityDescriptor.mInheritHandle = 0;
v12 = CreateEventW(&SecurityDescriptor, 0, 0, L"WerSvc\WerSvcSystemPermissionsEvent");
```
Tracing Back Event Creation

Validate Request
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"

### Diagram

**Malware**

- CreateMutex("ServiceMutex")
- If (error == ERROR_SUCCESS)
- If (error == ERROR_SUCCESS)
- No → Exit
  -Yes → Inject Code

**EDR Agent**

- CreateMutex("ServiceMutex")
- If (error == ERROR_SUCCESS)
- No → Exit
  -Yes → Protect
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.

Malware

CreateMutex("SvcNS\ServiceMutex")

If (error == ERROR_SUCCESS)

No → Exit

Yes → Inject Code

EDR Agent

CreateMutex("SvcNS\ServiceMutex")

If (error == ERROR_SUCCESS)

No → Exit

Yes → Protect
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 boundary descriptor via `AddSIDToBoundaryDescriptor()`
- Namespace is created via `CreatePrivateNamespace()`

The boundary descriptor is sent as a parameter

```
“MyNamespace”
```

```
“MyBoundaryDescriptor”
```

```
SYSTEM SID
```

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 namespace 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

```c
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

Checks performed by `CWerService::CheckIfCrashIsValid()`

- Sender PID == Target PID
  - yes
  - no

- `CheckIfSystemConnectingToPort()` (Check client SID)
  - yes
  - no

- Is package name the same?
  - yes

Return STATUS_SUCCESS
Opening the Event

- `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

**Client**
1. Execute tool as SYSTEM
2. Create MappedViewStruct
3. Create ALPC message
4. Signal Service
5. Create namespace with security boundary
6. Create event under the namespace
7. Create ALPC port
8. Wait for service to start
9. Send message

**Server**
5. Create namespace with security boundary
6. Create event under the namespace
7. Create ALPC port
10. Receive message
11. Compare sender PID to target PID
12. Open event after impersonation
13. Validate request
14. Perform Dump
Demonstration
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`

![Table showing dump files]

This PC > Local Disk (C) > Windows > system32 > config > systemprofile > AppData > Local > CrashDumps

<table>
<thead>
<tr>
<th>Name</th>
<th>Date Modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsass.exe.680.dmp</td>
<td>7/5/2022 8:47 AM</td>
<td>DMP File</td>
<td>50,655 KB</td>
</tr>
</tbody>
</table>
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
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

WerFault.exe is doing the dump

- Legitimate system process
- Crash report doesn't implicate sender process
- Registry key has legitimate usages
- Found in all Windows systems
- Binary architecture is irrelevant
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://flylib.com/books/en/2.294.1.98/1/
- 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/