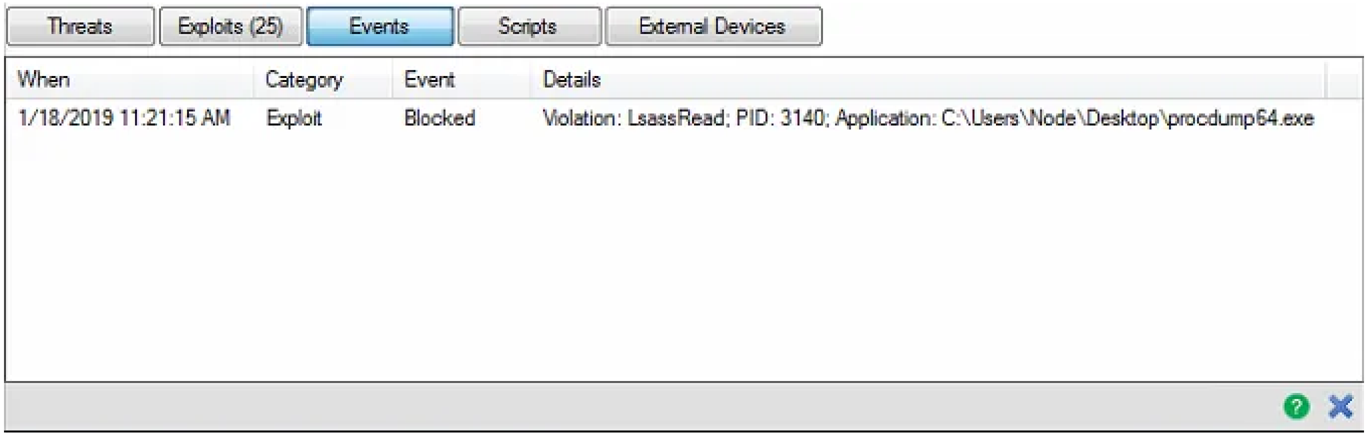
Bypass EDR's memory protection, introduction to hooking

Hoang Bui · Follow
8 min read · Jan 18, 2019

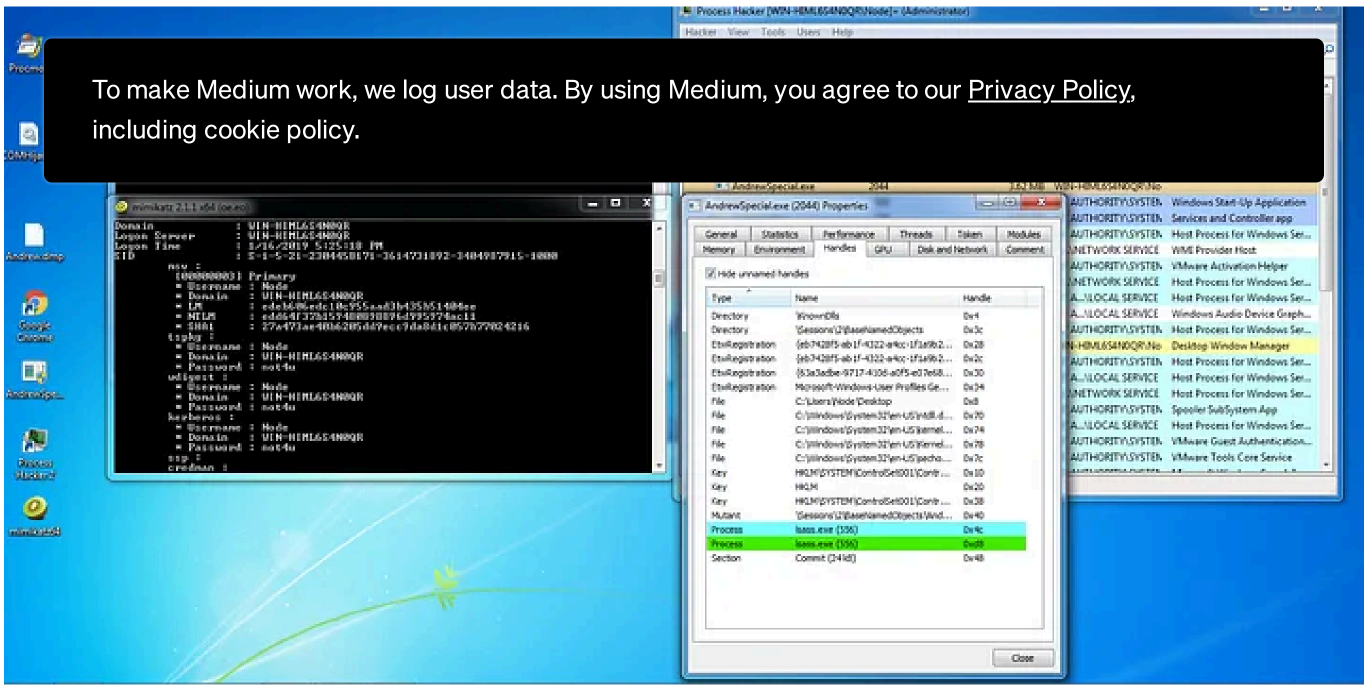
Introduction

On a recent internal penetration engagement, I was faced against an EDR product that I will not name. This product greatly hindered my ability to access lsass' memory and use our own custom flavor of Mimikatz to dump clear-text credentials.



When	Category	Event	Details
1/18/2019 11:21:15 AM	Exploit	Blocked	Violation: LsassRead; PID: 3140; Application: C:\Users\Node\Desktop\procdump64.exe

For those who recommends ProcDump



```
Domain          : WIN-HIML6S4N0QR
Logon Server    : WIN-HIML6S4N0QR
Logon Time      : 1/16/2019 5:25:18 PM
SID             : S-1-5-21-2304450171-3614731892-3404987915-1000
        msv :
         [00000003] Primary
         * Username : Node
         * Domain   : WIN-HIML6S4N0QR
         * LM       : ede b606edc10c955aad3b435b51404ee
         * NTLM     : edd64f37b1574000300076d995774ac11
         * SHA1     : 27a473ae40b6205dd9ecc7da8d1c057b77024216
        tspkg :
         * Username : Node
         * Domain   : WIN-HIML6S4N0QR
         * Password : not4u
        wdigest :
         * Username : Node
         * Domain   : WIN-HIML6S4N0QR
         * Password : not4u
        kerberos :
         * Username : Node
         * Domain   : WIN-HIML6S4N0QR
         * Password : not4u
        ssp :
        credman :
```

AndrewSpecial.exe (2044) Properties

☑ Hide unnamed handles

Type	Name	Handle
Directory	\KnownDlls	0x4
Directory	\Sessions\1\BaseNamedObjects	0x3c
EtwRegistration	{eb7428f5-ab1f-4322-a4cc-1f1a9b2...	0x28
EtwRegistration	{eb7428f5-ab1f-4322-a4cc-1f1a9b2...	0x2c
EtwRegistration	{63a3adbe-9717-410d-a0f5-e07e68...	0x30
EtwRegistration	Microsoft-Windows-User Profiles Ge...	0x34
File	C:\Users\Node\Desktop	0x8
File	C:\Windows\System32\en-US\ntdll.d...	0x70
File	C:\Windows\System32\en-US\kernel...	0x74
File	C:\Windows\System32\en-US\Kernel...	0x78
File	C:\Windows\System32\en-US\pacha...	0x7c
Key	HKLM\SYSTEM\ControlSet001\Contr...	0x10
Key	HKLM	0x20
Key	HKLM\SYSTEM\ControlSet001\Contr...	0x38
Mutant	\Sessions\1\BaseNamedObjects\And...	0x40
Process	lsass.exe (556)	0x4c
Process	lsass.exe (556)	0xd8
Section	Commit (24 kB)	0x48

There is no EDR solution on this machine, this was just an PoC

However, after thinking "I got this!" and was ready to rejoice in victory over defeating a certain EDR, I was met with a disappointing conclusion. The EDR blocked the shellcode injection into csrss as well as the thread creation through *RtlCreateUserThread*. However, for some reason — the code while failing to spawn as a child process and inherit the handle, was still somehow able to get the PROCESS_ALL_ACCESS handle to lsass.exe.

WHAT?!

Hold up, let me try just opening a handle to lsass.exe without any fancy stuff with just this line:

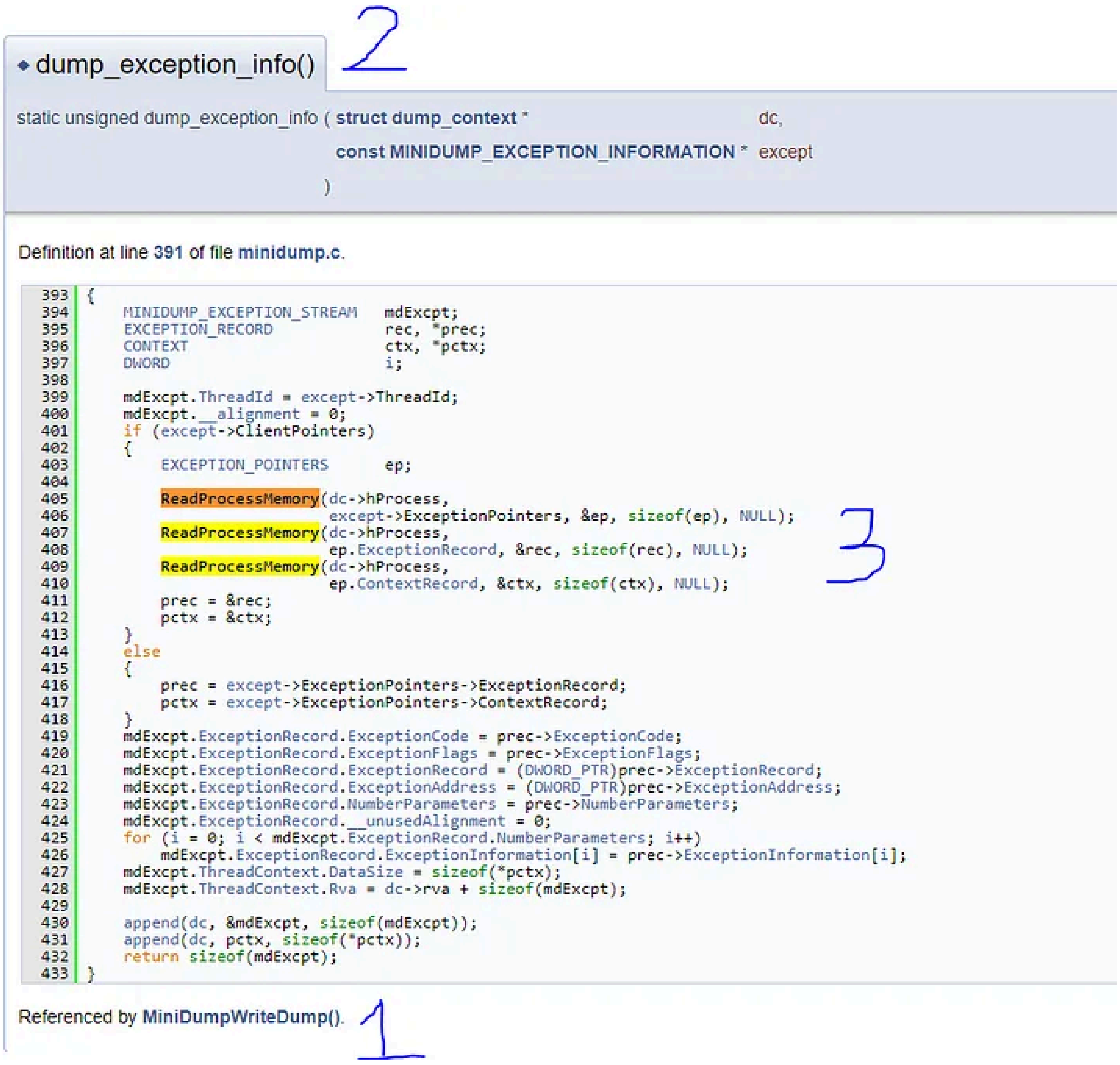
HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, FALSE, lsasspid);

And what do you know, I got a handle with FULL CONTROL over lsass.exe.

Let's dissect this warning further. "Violation: LsassRead". I didn't read an

pr

there must be some sort of WINAPI being called such as ReadProcessMemory (RPM) inside MiniDumpWriteDump(). Let's look at MiniDumpWriteDump's source code at ReactOS.



◆ dump_exception_info()

```
static unsigned dump_exception_info ( struct dump_context *                      dc,
                                      const MINIDUMP_EXCEPTION_INFORMATION *     except
                                    )
```

Definition at line 391 of file minidump.c.

```
{
    MINIDUMP_EXCEPTION_STREAM   mdExcpt;
    EXCEPTION_RECORD            rec, *prec;
    CONTEXT                     ctx, *pctx;
    DWORD                       i;

    mdExcpt.ThreadId = except->ThreadId;
    mdExcpt.__alignment = 0;
    if (except->ClientPointers)
    {
        EXCEPTION_POINTERS      ep;

        ReadProcessMemory(dc->hProcess,
                          except->ExceptionPointers, &ep, sizeof(ep), NULL);
        ReadProcessMemory(dc->hProcess,
                          ep.ExceptionRecord, &rec, sizeof(rec), NULL);
        ReadProcessMemory(dc->hProcess,
                          ep.ContextRecord, &ctx, sizeof(ctx), NULL);
        prec = &rec;
        pctx = &ctx;
    }
    else
    {
        prec = except->ExceptionPointers->ExceptionRecord;
        pctx = except->ExceptionPointers->ContextRecord;
    }
    mdExcpt.ExceptionRecord.ExceptionCode = prec->ExceptionCode;
    mdExcpt.ExceptionRecord.ExceptionFlags = prec->ExceptionFlags;
    mdExcpt.ExceptionRecord.ExceptionRecord = (DWORD_PTR)prec->ExceptionRecord;
    mdExcpt.ExceptionRecord.ExceptionAddress = (DWORD_PTR)prec->ExceptionAddress;
    mdExcpt.ExceptionRecord.NumberParameters = prec->NumberParameters;
    mdExcpt.ExceptionRecord.__unusedAlignment = 0;
    for (i = 0; i < mdExcpt.ExceptionRecord.NumberParameters; i++)
        mdExcpt.ExceptionRecord.ExceptionInformation[i] = prec->ExceptionInformation[i];
    mdExcpt.ThreadContext.DataSize = sizeof(*pctx);
    mdExcpt.ThreadContext.Rva = dc->rva + sizeof(mdExcpt);

    append(dc, &mdExcpt, sizeof(mdExcpt));
    append(dc, pctx, sizeof(*pctx));
    return sizeof(mdExcpt);
}
```

Referenced by MiniDumpWriteDump().

Multiple calls to RPM

RPM — > NtReadVirtualMemory –> SYSCALL->NtReadVirtualMemory

Ke

With that knowledge, we now must identify HOW the EDR product is detecting and stopping the RPM/NtReadVirtualMemory call. This comes as a simple answer which is "hooking". Please refer to my previous post regarding hooking here for more information. In short, it gives you the ability to put your code in the middle of any function and gain access to the arguments as well as the return variable. I am 100% sure that the EDR is using some sort of hook through one or more of the various techniques that I mentioned.

However, readers should know that most if not all EDR products are using a service, specifically a driver running inside kernel mode. With access to the kernel mode, the driver could perform the hook at ANY of the level in the RPM's callstack. However, this opens up a huge security hole in a Windows environment if it was trivial for any driver to hook ANY level of a function. Therefore, a solution is to put forward to prevent modification of such nature and that solution is known as Kernel Patch Protection (KPP or Patch Guard). KPP scans the kernel on almost every level and will triggers a BSOD if a modification is detected. This includes ntoskrnl portion which houses the WINAPI's kernel level's logic. With this knowledge, we are assured that the EDR would not and did not hook any kernel level function inside that portion of the call stack, leaving us with the user-land's RPM and NtReadVirtualMemory calls.

The Hook

Now, this provides us with the address of both RPM and ntReadVirtualMemory. I will now use my favorite reversing tool to read the memory and analyze its structure, Cheat Engine.

ReadProcessMemory

NtReadVirtualMemory

For the RPM function, it looks fine. It does some stack and register set up and then calls ReadProcessMemory inside Kernelbase (Topic for another time). Which would eventually leads you down into ntdll’s

NtReadVirtualMemory, the first instruction is actually a JMP instruction to an

CyMemDef64.dll

Okay, so we are no longer inside ntdll's module but instead inside CyMemdef64.dll's module. Ahhhhh now I get it.

The EDR placed a jump instruction where the original NtReadVirtualMemory function is supposed to be, redirect the code flow into their own module which then checked for any sort of malicious activity. If the checks fail, the Nt* function would then return with an error code, never entering the kernel land and execute to begin with.

The Bypass

It is now very self-evident what the EDR is doing to detect and stop our WINAPI calls. But how do we get around that? There are two solutions.

Re-Patch the Patch

We know what the NtReadVirtualMemory function *SHOULD* looks like and we can easily overwrite the jmp instruction with the correct instructions. This will stop our calls from being intercepted by CyMemDef64.dll and enter for my manager Andrew who is currently battling a busted appendix in the ho

AndrewSpecial.exe was never caught :P

Conclusion

This currently works for this particular EDR, however — It would be trivial to reverse similar EDR products and create a universal bypass due to their limitation around what they can hook and what they can’t (Thank you KPP).

Did I also mention that this works on both 64 bit (on all versions of windows) and 32 bits (untested)? And the source code is available HERE.

Thank you again for your time and please let me know if I made any mistake.

Programming Hacking Malware Bypass Endpoint Security

312 6

Hoang Bui

Hooking Heaven’s Gate — a WOW64 hooking technique

This is not new, this is not novel, and definitely not my research — but I used it recently so...

May 14, 2019 42 3

Hoang Bui

Faking your return address through Gadget and ROP

Skip the background if you want to keep your sanity, it is meme.

Apr 12, 2019 115

Hoang Bui

Weaponizing vulnerable driver for privilege escalation— Gigabyte...

End Result

Jun 29, 2019 67

Hoang Bui

Vectored Exception Handling, Hooking Via Forced Exception

As a security researcher, it comes to my attention that the ability to modify and...

Jan 13, 2019 16

“From DLL to Shell: A Step-by-Step Guide to
Re

This is the beginning of the series of my brief

Lists

General Coding Knowledge
20 stories · 1693 saves

Coding & Development
11 stories · 881 saves

Stories to Help You Grow as a Software Developer
19 stories · 1452 saves

ChatGPT
21 stories · 855 saves

Satyam Pathania in InfoSec Write-ups

Why I Don’t Recommend People To Get into Cybersecurity?

Cybersecurity isn’t always what it seems — it’s tough, demanding, and stressful.

Oct 24 · 388 · 6

MrHeckerCat

Write Up:Introduction to Malware Analysis- HTB Academy

Hi again! This is my next write up and this time I’m covering the Skill Assessment section of...

Jul 3 · 3