MIRcon 2014

PIN Down the Malware:

Using Machine Learning Techniques to Augment IOCs

Andrei Saygo
Microsoft

Jason Coleman
Microsoft

About us

Who we are.

Ninjas. Seriously ☺

Agenda

- IOCs at a glance
- Instrumenting with PIN
- Our PIN plugin
- Classification
  - Association rule learning
  - Naive Bayes classifiers
  - Feature selection
  - Call Hashing
- Visualisation
- Signatures
- Demo
- Next steps

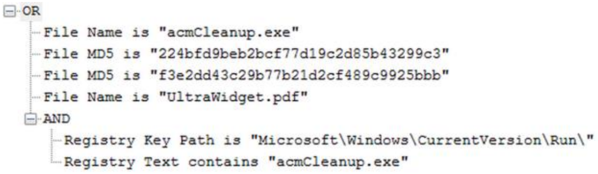
IOCs at a glance

IOC = **I**ndicator **O**f **C**ompromise

IOCs are forensic artefacts of an intrusion that can be identified on a host or network.

An IOC (also sometimes just called an Indicator) is a logically grouped set of descriptive terms (each called an “Indicator Term”) about a specific threat.

IOCs at a glance



```
OR
  File Name is "acmCleanup.exe"
  File MD5 is "224bfd9beb2bcf77d19c2d85b43299c3"
  File MD5 is "f3e2dd43c29b77b21d2cf489c9925bbb"
  File Name is "UltraWidget.pdf"
  AND
    Registry Key Path is "Microsoft\Windows\CurrentVersion\Run\"
    Registry Text contains "acmCleanup.exe"
```

They are written in XML and are based on the Open IOC schema (www.openioc.org)

More details:

http://openioc.org/resources/An_Introduction_to_OpenIOC.pdf

https://www.mandiant.com/blog/openioc-series-investigating-indicators-compromise-iocs-part/

Instrumenting with PIN

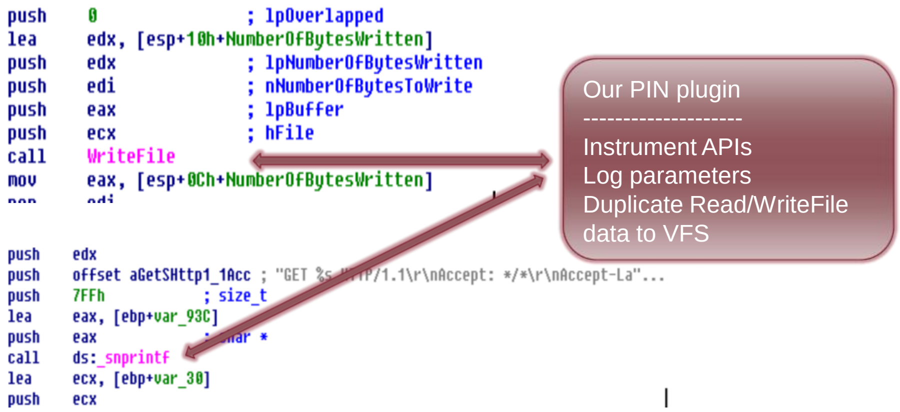
Instrumenting with PIN: About

Dynamic binary instrumentation is a method of analyzing the behaviour of an application by injecting instrumentation code that executes as part of the normal stream of instructions.

Our PIN plugin performs:

- API Hooking (for API's of Interest)
- Memory Modification Detection (for regions of interest –heap, .data etc…) – leading to DataViz
- Trace executed instructions

Instrumenting with PIN



```
push    0                 ; lpOverlapped
lea     edx, [esp+10h+NumberOfBytesWritten]
push    edx               ; lpNumberOfBytesWritten
push    edi               ; nNumberOfBytesToWrite
push    eax               ; lpBuffer
push    ecx               ; hFile
call    WriteFile
mov     eax, [esp+0Ch+NumberOfBytesWritten]
```

```
push    edx
push    offset aGetSHttp1_1Acc ; "GET %s HTTP/1.1\r\nAccept: */*\r\nAccept-La"...
push    7FFh            ; size_t
lea     eax, [ebp+var_93C]
push    eax             ; char *
call    ds:_snprintf
lea     ecx, [ebp+var_30]
push    ecx
```

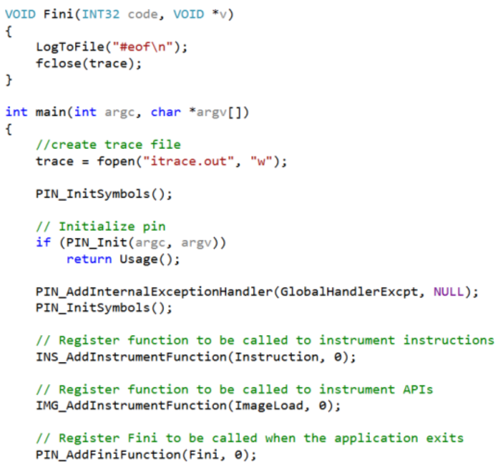
Our PIN plugin

Instrument APIs
Log parameters
Duplicate Read/WriteFile data to VFS

Instrumenting with PIN: PIN plugin

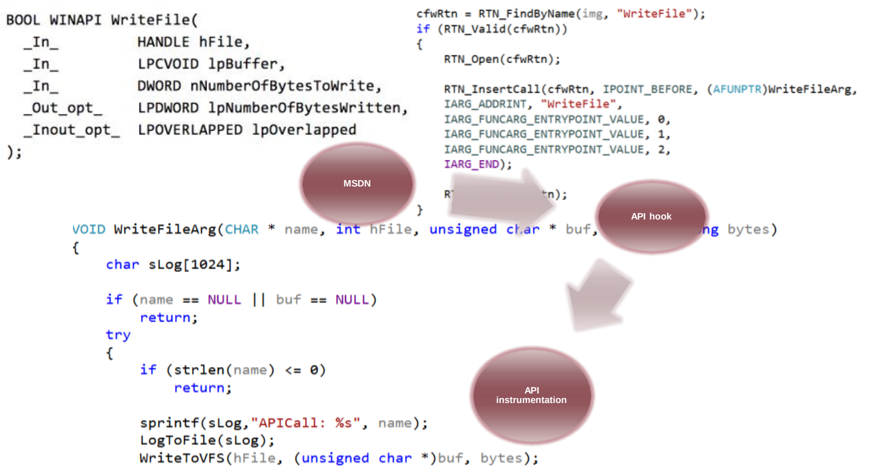
Image instrumentation can inspect and instrument an entire image when it’s first loaded.

Instrumentation can be inserted so that it’s executed before or after a code sequence is executed.



```
VOID Fini(INT32 code, VOID *v)
{
    LogToFile("#eof\n");
    fclose(trace);
}

int main(int argc, char *argv[])
{
    //create trace file
    trace = fopen("itrace.out", "w");

    PIN_InitSymbols();

    // Initialize pin
    if (PIN_Init(argc, argv))
        return Usage();

    PIN_AddInternalExceptionHandler(GlobalHandlerExcpt, NULL);
    PIN_InitSymbols();

    // Register function to be called to instrument instructions
    INS_AddInstrumentFunction(Instruction, 0);

    // Register function to be called to instrument APIs
    IMG_AddInstrumentFunction(ImageLoad, 0);

    // Register Fini to be called when the application exits
    PIN_AddFiniFunction(Fini, 0);
```

Instrumenting with PIN: PIN plugin



```
BOOL WINAPI WriteFile(
  _In_          HANDLE hFile,
  _In_          LPCVOID lpBuffer,
  _In_          DWORD nNumberOfBytesToWrite,
  _Out_opt_     LPDWORD lpNumberOfBytesWritten,
  _Inout_opt_   LPOVERLAPPED lpOverlapped
);
```

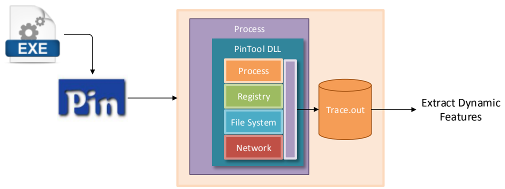
```
cfwRtn = RTN_FindByName(img, "WriteFile");
if (RTN_Valid(cfwRtn))
{
    RTN_Open(cfwRtn);

    RTN_InsertCall(cfwRtn, IPOINT_BEFORE, (AFUNPTR)WriteFileArg,
    IARG_ADDRINT, "WriteFile",
    IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
    IARG_FUNCARG_ENTRYPOINT_VALUE, 1,
    IARG_FUNCARG_ENTRYPOINT_VALUE, 2,
    IARG_END);

    R            tn);
}
```

```
VOID WriteFileArg(CHAR * name, int hFile, unsigned char * buf,          ng bytes)
{
    char sLog[1024];

    if (name == NULL || buf == NULL)
        return;
    try
    {
        if (strlen(name) <= 0)
            return;

        sprintf(sLog,"APICall: %s", name);
        LogToFile(sLog);
        WriteToVFS(hFile, (unsigned char *)buf, bytes);
```

Classification

Dynamic Analysis

- We use Intel PIN to instrument the sample under test
- The PinTool has been designed to **extract features** related to the behaviour of the sample



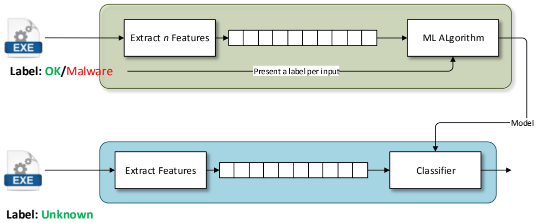
- We focus on the samples’ interaction with:
  - Network, file system, registry and process sub-system

Malware Classification

Problem: Given an unknown sample, determine if it is malicious

Types of ML: Supervised/Unsupervised

- Steps:
  1. Train the model using a set of known files
  2. Use the generated model to classify an unknown file



Malware Classification (cont.)

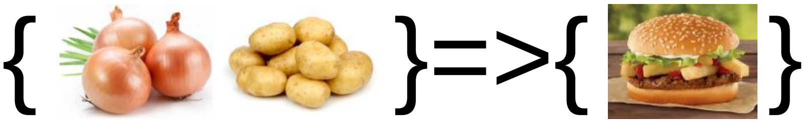
The classifier module uses two supervised approaches:

- Association Rules Mining/Learning (primary)
- Naïve Bayes (optional-backup)

Note: We may reduce the number of classifiers to one once we have identified the best performing sample

Association rule learning

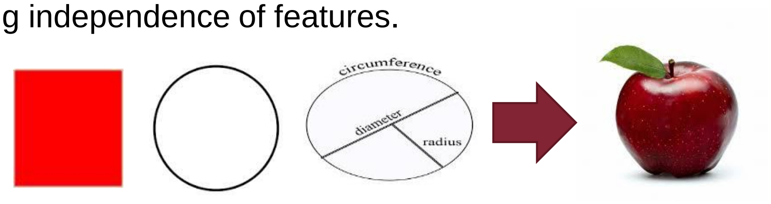
Is a method for discovering relations between variables in large databases.



Rakesh Agrawal et al. introduced association rules for discovering regularities between products sold in supermarkets.

Naïve Bayes Classification

A simple probabilistic classifier based on an assumption of strong independence of features.



Uses a supervised learning approach (you need to tell it what’s good and what’s bad) – pick your samples well.

Feature Selection

- We use a combination of features extracted during the static and dynamic analysis phase
- The table shows a selection of some of the features used
- The last sample is actually Internet Explorer.exe

Static Features				Dynamic Features					Label
Size	Entropy	Packed	Type of Packer	Network Download/Data Exfil	File System Modification	Registry Modification	Persistence	Process Creation/ Tampering	
65K	7.8	Yes	Mod UPX	Yes	Yes	Yes	Yes	Yes	Malicious
794K	5.2	No	MSVC++	Yes	Yes	Yes	No	No	Benign

MIRcon 2014

Feature Selection: Call Hashing

Based on the technique explained here https://www.mandiant.com/blog/tracking-malware-import-hashing/

It may be difficult to reliably use import hashes with packed/obfuscated malware or when various APIs are used just to defeat emulators.

MIRcon 2014

Feature Selection: (Dynamic) Call Hashing

We are hashing only functions that are actually called/used by the program with or without their parameters.

Different hashes for different groups of APIs

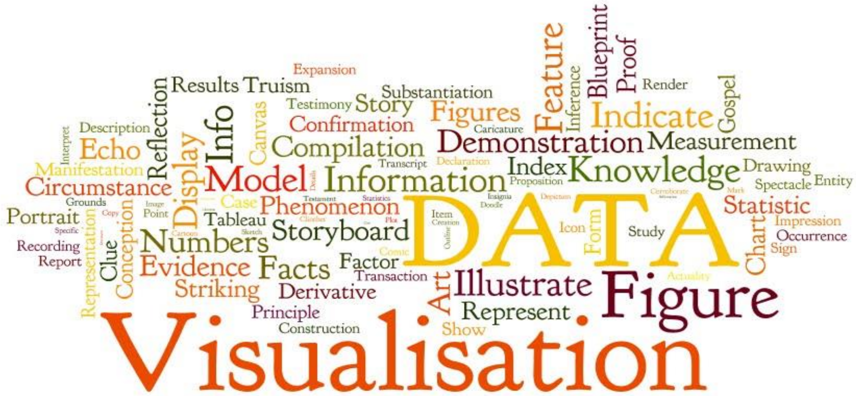
e.g.:

- Group1: CreateFile, ReadFile, WriteFile, CloseHandle
- Group 2: InternetOpen, InternetConnect, InternetSetOption, HttpSendRequest …
- Group N: RtlInitString, strcpy, strlen, strcmp

Feature Selection: (Dynamic) Call Hashing

```
…
APICall: lstrlenA(m***i.com)
APICall: lstrlenA(http://%s)                     1
APICall: InternetOpenA
APICall: InternetConnectA(m***i.com)             2
APICall: RegCreateKeyExA(Software\Microsoft\windows\CurrentVersion\Internet
Settings\Connections)                            3
APICall: lstrlenA(POST)
APICall: lstrlenA(HTTP/1.1)                      1
APICall: lstrlenA(h***i.net)
APICall: lstrlenA(http://%s)
APICall: InternetConnectA(h***i.net)             2
APICall: lstrlenA(POST)                          1
APICall: lstrlenA(HTTP/1.1)
…
```

Visualisation



Visualisation

Problem: reading raw assembler trace data is necessary but time-consuming

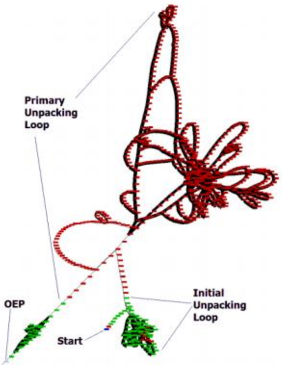
We currently have more information than we know what to do with.

Visualisation can help **BUT** you need to ensure that the method suits the data set

Visualisation: Previous Work

VeraTrace – Danny Quist – provides a graph based view of code execution. Visualisation of complex decryption.

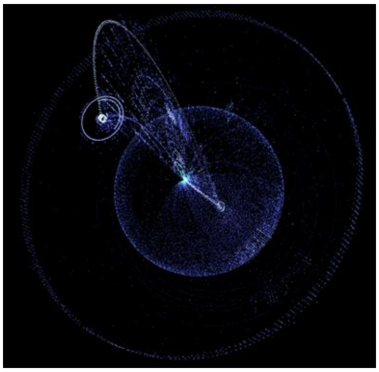
http://www.offensivecomputing.net/?q=node/1687



Visualisation: Previous Work

CantorDust – Recon 2013 – allows searching for image patterns instead of code patterns.

https://sites.google.com/site/xxcantorxdustxx/about

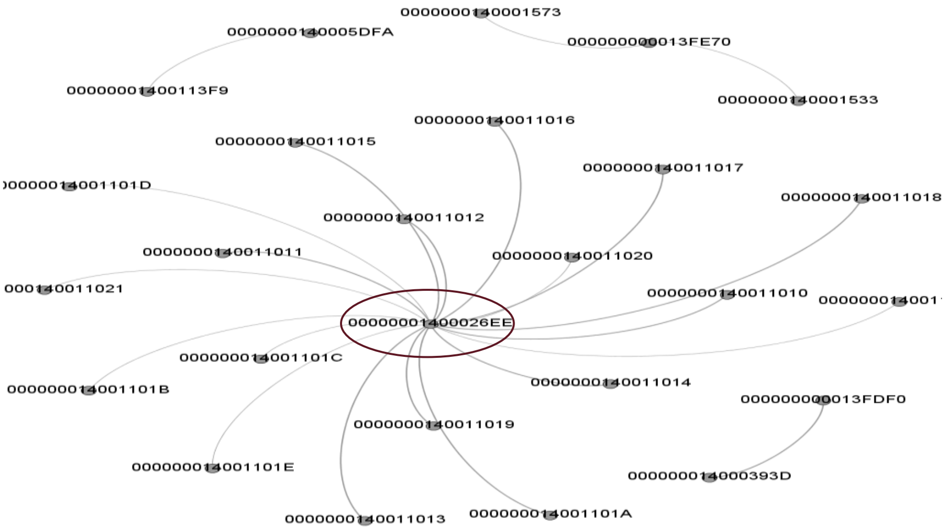


Visualisation: PinTool

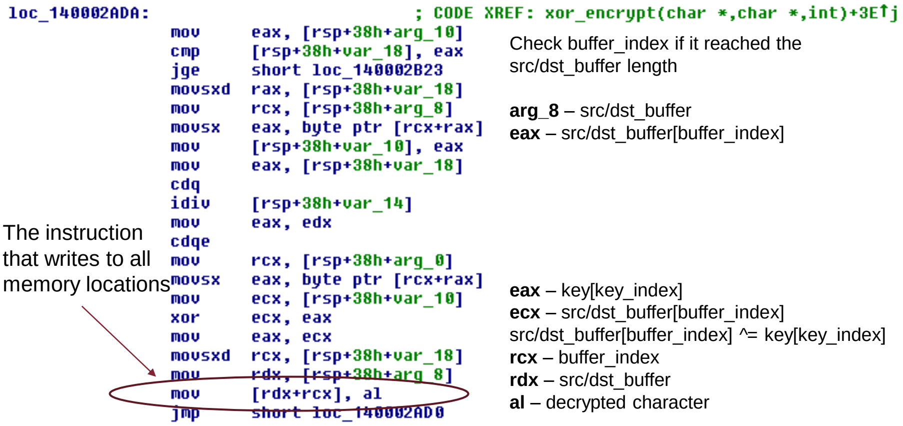
Determine memory regions written – useful for decryption/deobfucation loops.

```
if (INS_IsMemoryWrite(ins)) {
    INS_InsertPredicatedCall(
        ins, IPOINT_BEFORE, (AFUNPTR)RecordMemWrite,
        IARG_MEMORYWRITE_EA,
        IARG_MEMORYWRITE_SIZE,
        IARG_END); }
…
VOID RecordMemWrite(VOID * addr, UINT32 size) {
    if ( size == 1 ) {   //---> detect byte decryption loops
        MemWriteToVFS(0, (unsigned char *)addr, size, s_addr);
```

Visualisation: Gephi



Visualisation: IDA



```
loc_140002ADA:                          ; CODE XREF: xor_encrypt(char *,char *,int)+3E↑j
                mov     eax, [rsp+38h+arg_10]
                cmp     [rsp+38h+var_18], eax
                jge     short loc_140002B23
                movsxd  rax, [rsp+38h+var_18]
                mov     rcx, [rsp+38h+arg_8]
                movsx   eax, byte ptr [rcx+rax]
                mov     [rsp+38h+var_10], eax
                mov     eax, [rsp+38h+var_18]
                cdq
                idiv    [rsp+38h+var_14]
                mov     eax, edx
                cdqe
                mov     rcx, [rsp+38h+arg_0]
                movsx   eax, byte ptr [rcx+rax]
                mov     ecx, [rsp+38h+var_10]
                xor     ecx, eax
                mov     eax, ecx
                movsxd  rcx, [rsp+38h+var_18]
                mov     rdx, [rsp+38h+arg_8]
                mov     [rdx+rcx], al
                jmp     short loc_140002AD0
```

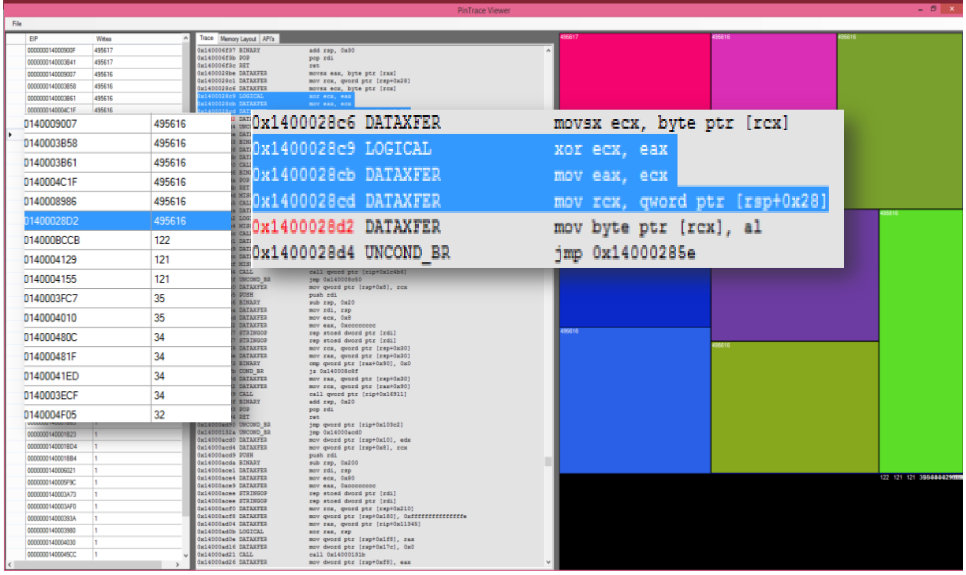
Check buffer_index if it reached the src/dst_buffer length

arg_8 – src/dst_buffer
eax – src/dst_buffer[buffer_index]

The instruction that writes to all memory locations

eax – key[key_index]
ecx – src/dst_buffer[buffer_index]
src/dst_buffer[buffer_index] ^= key[key_index]
rcx – buffer_index
rdx – src/dst_buffer
al – decrypted character

Visualisation: PIN Trace Viewer



0140009007	495616
0140003B58	495616
0140003B61	495616
0140004C1F	495616
0140008986	495616
01400028D2	495616
014000BCCB	122
0140004129	121
0140004155	121
0140003FC7	35
0140004010	35
014000480C	34
014000481F	34
01400041ED	34
0140003ECF	34
0140004F05	32

```
0x1400028c6 DATAXFER    movsx ecx, byte ptr [rcx]
0x1400028c9 LOGICAL     xor ecx, eax
0x1400028cb DATAXFER    mov eax, ecx
0x1400028cd DATAXFER    mov rcx, qword ptr [rsp+0x28]
0x1400028d2 DATAXFER    mov byte ptr [rcx], al
0x1400028d4 UNCOND_BR   jmp 0x14000285e
```

Visualisation: PIN Trace Viewer

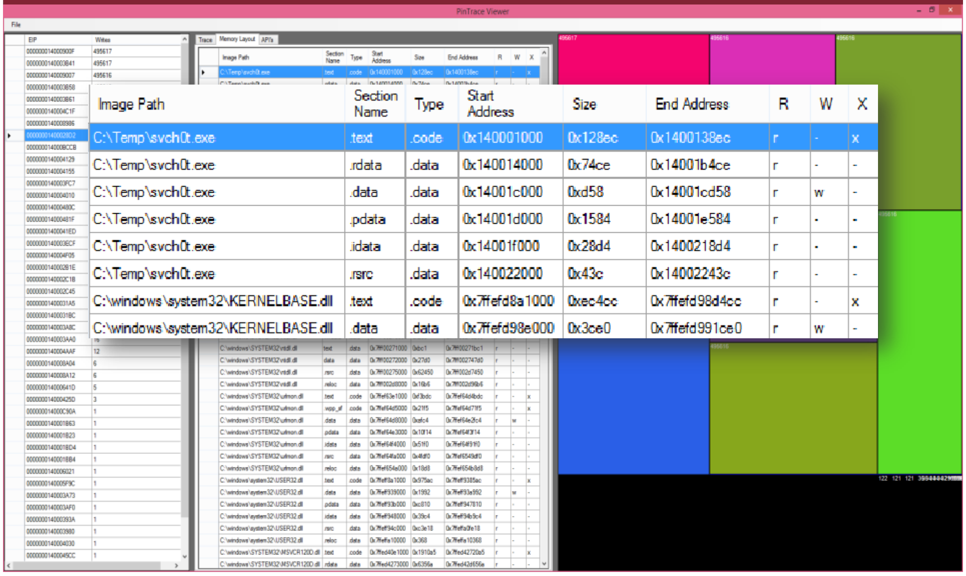
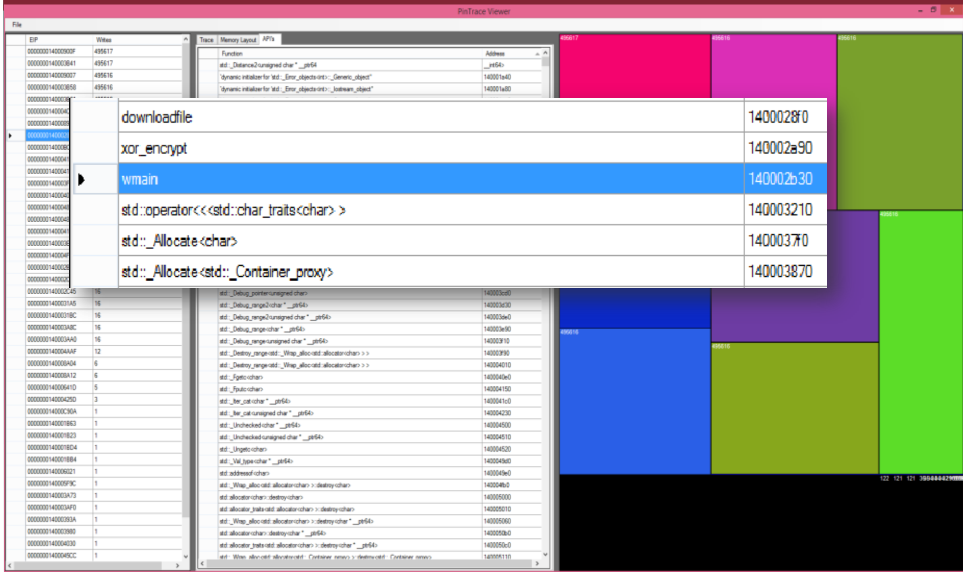


Image Path	Section Name	Type	Start Address	Size	End Address	R	W	X
C:\Temp\svch0t.exe	.text	.code	0x140001000	0x128ec	0x1400138ec	r	-	x
C:\Temp\svch0t.exe	.rdata	.data	0x140014000	0x74ce	0x14001b4ce	r	-	-
C:\Temp\svch0t.exe	.data	.data	0x14001c000	0xd58	0x14001cd58	r	w	-
C:\Temp\svch0t.exe	.pdata	.data	0x14001d000	0x1584	0x14001e584	r	-	-
C:\Temp\svch0t.exe	.idata	.data	0x14001f000	0x28d4	0x1400218d4	r	-	-
C:\Temp\svch0t.exe	.rsrc	.data	0x140022000	0x43c	0x14002243c	r	-	-
C:\windows\system32\KERNELBASE.dll	.text	.code	0x7ffefd8a1000	0xec4cc	0x7ffefd98d4cc	r	-	x
C:\windows\system32\KERNELBASE.dll	.data	.data	0x7ffefd98e000	0x3ce0	0x7ffefd991ce0	r	w	-

Visualisation: PIN Trace Viewer



downloadfile	1400028f0
xor_encrypt	140002a90
wmain	140002b30
std::operator<<<std::char_traits<char> >	140003210
std::_Allocate<char>	1400037f0
std::_Allocate<std::_Container_proxy>	140003870

Guess what's next… ☺

IOC extractor

Other things that we can do with the PIN plugin.

Our PIN plugin extracts API calls along with parameters.
Why not just convert that to the Open IOC format ?!

IOC extractor

Not so simple, we have a lot of information and not all is actually malicious, so we’re doing a filtering with several types of white lists:

- clean/not so useful APIs

  e.g.: GetModuleHandle, LoadLibraryA

- clean files

  e.g.: all the files in a known clean VM image and that aren’t changed

- artefacts to ignore

  e.q.: environment variables like:

  ALLUSERSPROFILE=C:\ProgramData
  APPDATA=C:\Users
  SystemRoot=C:\Windows

IOC extractor

Besides whitelists we are filtering the API parameters to include only relevant content:

- we have regexes that are looking for executable files, archives, registry keys, URLs, IP addresses, etc.

- we match APIs to various items from the Open IOC schema.

IOC extractor

Example:

CopyFile, CreateFileA, CreateFileW, …

```
<iocterm term-source="application/vnd.mandiant.mir" display-type="string"
text="FileItem/FullPath" title="File Full Path" data-type="xs:string" />
```

```
<IndicatorItem id=”" condition="matches" preserve-case="false" negate="false”>
<Context document="FileItem" search="FileItem/FullPath"
type="mir"></Context>
<Content type="string">$api_arg</Content>
</IndicatorItem>
```

yara

Multi-platform pattern matching Swiss-army knife for malware researchers (and everyone else).

It is used to identify and classify malware samples.

More details here: http://plusvic.github.io/yara/

yara

We can use the events recorded by our PIN tool to create yara rules.

Example:

```
private rule IsPE
{
        condition:
                uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550
}
rule Trojan.Autom.1
{
        strings:
                $s0 = /Mozilla\/4\.0 \(compatible; MSIE 6\.0; Windows NT 5\.1\)/
                $s1 = /Mozilla\/4\.0 \(compatible; MSIE 6\.0; Windows NT 5\.1\)/ wide
                $s2 = /http:\/\/users\.***\.com\/fcg-bin\/cgi_get_portrait\.fcg?uins=/
                $s3 = /http:\/\/users\.***\.com\/fcg-bin\/cgi_get_portrait\.fcg?uins=/ wide
        condition:
                IsPE and any of ($s*)
}
```

snort

Network Intrusion Detection System (NIDS) mode.

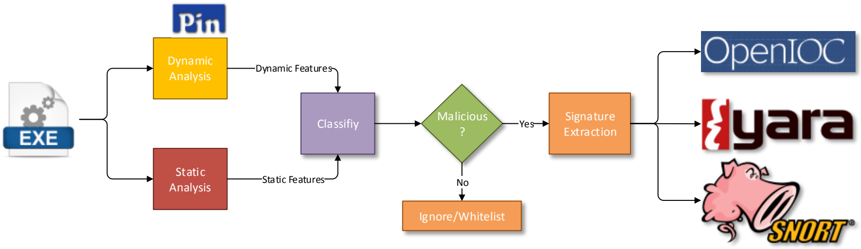
The alerts that are generated will not contain any values that may be system dependent.

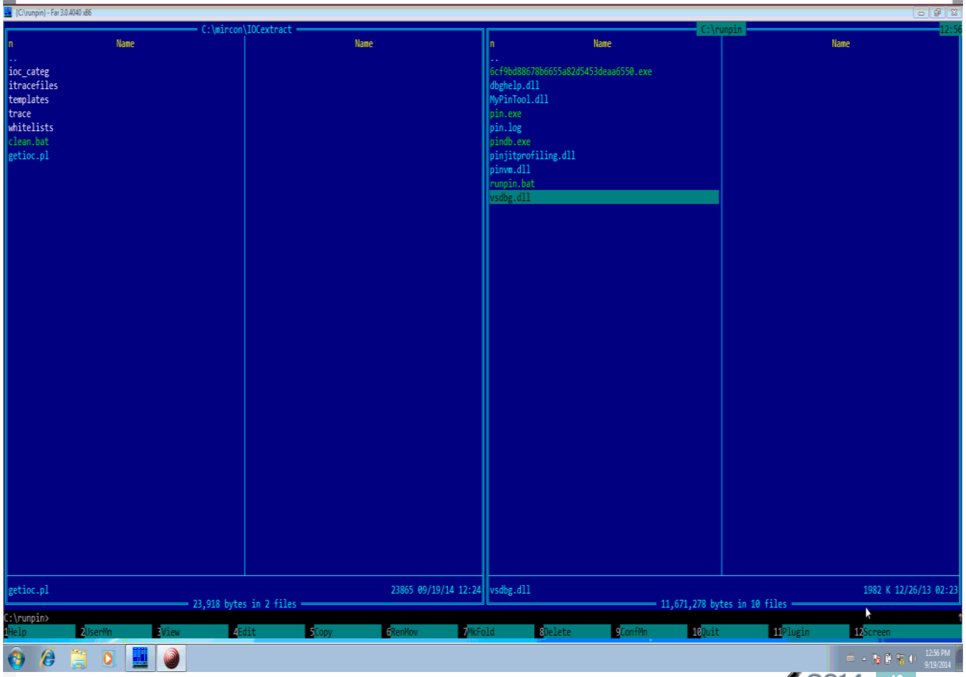
Example:

*alert tcp any any -> any any (msg:"potential malicious traffic http://users.***.com/fcg-bin/cgi_get_portrait.fcg?uins=211284131"; content:"**/fcg-bin/cgi_get_portrait.fcg?uins=**"; content:"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" content:"Host: users.***.com"; sid:10000001;)*

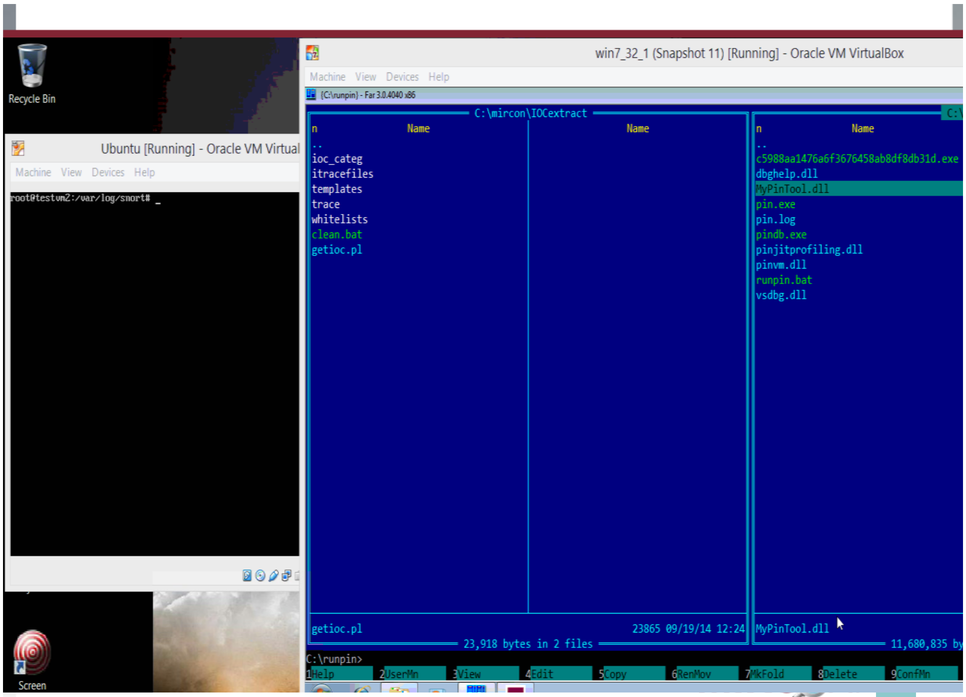
How all ties together

- Perform static and dynamic analysis on a sample
- Classify it
- **If** malicious **then** generate a signature for it





```
{C:\runpin} - Far 3.0.4040 x86

C:\mircon\IOCextract                                   C:\runpin                              12:56
n            Name                  Name                n            Name                  Name
..                                                     ..
ioc_categ                                              6cf9bd88678b6655a82d5453deaa6550.exe
itracefiles                                            dbghelp.dll
templates                                              MyPinTool.dll
trace                                                  pin.exe
whitelists                                             pin.log
clean.bat                                              pindb.exe
getioc.pl                                              pinjitprofiling.dll
                                                       pinvm.dll
                                                       runpin.bat
                                                       vsdbg.dll

getioc.pl                     23865 09/19/14 12:24     vsdbg.dll                    1982 K 12/26/13 02:23
            23,918 bytes in 2 files                              11,671,278 bytes in 10 files
C:\runpin>
1Help  2UserMn  3View  4Edit  5Copy  6RenMov  7MkFold  8Delete  9ConfMn  10Quit  11Plugin  12Screen
```


Recycle Bin
Ubuntu [Running] - Oracle VM Virtual
Machine View Devices Help
win7_32_1 (Snapshot 11) [Running] - Oracle VM VirtualBox
Machine View Devices Help
{C:\runpin} - Far 3.0.4040 x86
Screen

```
root@testvm2:/var/log/snort# _
```

```
C:\mircon\IOCextract
n              Name                        Name              n              Name
..                                                           ..
ioc_categ                                                    c5988aa1476a6f3676458ab8df8db31d.exe
itracefiles                                                  dbghelp.dll
templates                                                    MyPinTool.dll
trace                                                        pin.exe
whitelists                                                   pin.log
clean.bat                                                    pindb.exe
getioc.pl                                                    pinjitprofiling.dll
                                                             pinvm.dll
                                                             runpin.bat
                                                             vsdbg.dll

getioc.pl                      23865 09/19/14 12:24          MyPinTool.dll
              23,918 bytes in 2 files                                   11,680,835 by
C:\runpin>
1Help  2UserMn  3View  4Edit  5Copy  6RenMov  7MkFold  8Delete  9ConfMn
```

Next steps

Q & A

Thank you

andreisa@microsoft.com

jacolema@microsoft.com