

# To Catch an APT: YARA



# The Threat Intelligence Problem



# The Threat Intelligence Problem

## WebSense Security Labs Blog

WebSense Security Labs discovers, investigates and reports on advanced Internet threats that traditional research methods miss.

OPPORTUNITY KNOWS NO BOUNDARY: A CASE STUDY OF ACQUISITION

[Opportunity Knows No Boundary: A Case Study of Acquisition](#)

Posted: 24 Apr 2015 10:35 AM | [uwang](#) | [no comments](#)



BLOG HI



## PLUGX USES LEGITIMATE SAMSUNG APPLICATION FOR DLL SIDE-LOADING

20

POSTED BY: [Robert Falcone](#) on May 1, 2015 1:29 PM



# Malicious Code Analysis





## The YARA Project

<https://github.com/plusvic/yara/releases>

The pattern matching swiss-knife for malware researchers (and everyone else)



# What does YARA look like?

**Usage:** `yara [OPTION] [RULES_FILE] [SOURCE_FILE | SOURCE_DIR | PID]`

**Output:** `[RULE_NAME] [STREAM_LOCATION]`

```
C:\_tools\Yara3.3>yara32.exe -r rules.yr C:\malwr\PlugX\24FC5871407F180ECAD9DA6F67DD1878
```

```
UNKNOWN_PlugXTrojanLoader_PayloadNames [SOURCE_DIR]\Extracted\msi.dll
```

```
APTGroupX_PlugXTrojanLoader_StringDecode [SOURCE_DIR]\Extracted\msi.dll
```

```
GENERIC_SFXRAR_Installer [SOURCE_DIR]\x2015.exe
```

```
C:\_tools\Yara3.3>yara32.exe -r rules.yr C:\malwr\PlugX\D9AB2B14E9B2F1D78C117FDB1BF0601E
```

```
UNKNOWN_PlugXPayload_XVHeader [SOURCE_DIR]\Extracted\FromMem  
\Region00AB0000-00AD7000.dmp
```



# What does YARA scan?

- Data Stream
- File
- Network
- Memory



# First YARA Rule

```
rule ExampleRuleName
{
    meta:
        source =
"http://yara.readthedocs.org/en/v3.4.0/writingrules.html"
        description = "This is a very basic example rule."
    strings:
        $my_text_string = "text here"
        $my_hex_string = { E2 34 A1 C8 23 FB }
        $my_regex = /[0-9a-zA-Z]{32}/
    condition:
        $my_text_string or $my_hex_string or $my_regex
}
```





# What can you signature?

## TEXT STRINGS

STRING CONSTANTS

API Names

Error messages

String formatting style

Grammar mistakes

C&C commands

Timestamp formatting

Unique Sequences

Regular Expressions

## IMPLEMENTATION TRAITS

Memory allocation habits

Use of global variables

Multi-threading model

Software architecture and design

Constructor design

Dynamic API loading technique

Exception handling

Usage of public source code

Programming language and compiler

Compilation time stamps and time zones

## CUSTOM FEATURES

Obfuscation techniques

Stealth and evasion techniques

Encryption and compression algorithms

Cryptographic Keys & Constants

Re-used source code

Malware specific features

System infiltration

Propagation mechanisms

Artifact naming schemes / algorithms

Data exfiltration techniques

System / OS version determination technique

C&C command parsing implementation



# Using Yara Effectively

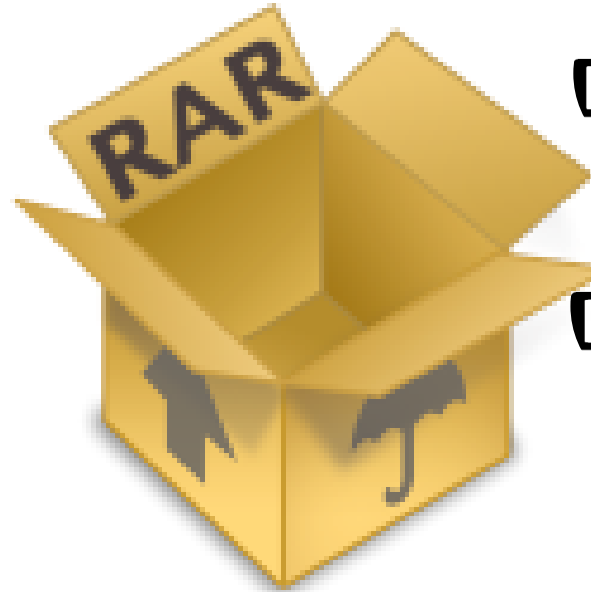
1. **Rules, Rules, Rules**
2. Descriptive Names
3. Have at least **THREE** sources of data
4. Constant enrichment and tuning



# Typical PlugX Malware Packaging

Self-Extracting Archive  
(SFX)

- Digitally Signed EXE
- DLL Loader
- Compressed PlugX Payload



# Installer/Dropper



```
rule GENERIC_SFXRAR_Installer {  
  strings:
```

```
    $str1 = "RarSFX" ascii wide  
    $str2 = "RENAMEDLG" ascii wide  
    $str3 = "GETPASSWORD1" ascii wide  
    $str4 = "ASKNEXTVOL"ascii wide  
    $str5 = "STATIC" ascii wide  
    $str6 = "REPLACEFILEDLG" ascii wide  
    $str7 = "winrarsfxmappingfile.tmp" ascii
```

```
  wide
```

```
  condition:
```

```
    (uint16(0) == 0x5A4D) and //Check for MZ  
                                offset 0
```

```
  at
```

```
  all of them
```

```
}
```



# Tuning String Rules

wide and fullword

```
$str1 = "setup.msi" wide fullword
```

nocase

```
// will hit on "KeRnEl32.DIL"
```

```
$str2 = "kernel32.dll" wide ascii nocase
```

whitespace characters

```
$str4 = "\nuname\n\n" wide ascii
```





### ASCII/8859-1 Text

A	0100 0001
S	0101 0011
C	0100 0011
I	0100 1001
I	0100 1001
/	0010 1111
8	0011 1000
8	0011 1000
5	0011 0101
9	0011 1001
-	0010 1101
l	0011 0001
	0010 0000
t	0111 0100
e	0110 0101
x	0111 1000
t	0111 0100

### Unicode Text

A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
س	0000 0110 0011 0011
ج	0000 0110 0100 0100
ی	0000 0110 0011 0111
ف	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
κ	0010 0010 0111 0000
γ	0000 0011 1011 0011



# Strings Have a Purpose

## **METASPLOIT\_UACBypass\_OpenProcessFail**

*\$a1 = "Couldn't open process " wide*

*\$a2 = "ERROR\_ACCESS\_DENIED\n(We probably tried to inject into an elevated process\nwhich isn't allowed unless we're also elevated.\nPick an unelevated process.)" wide*

## **GENERIC\_CMDShell\_ComSpecVariable**

*\$shell = "COMSPEC" wide ascii nocase*

## **DarkSeoul\_TDrop2\_Base64Alphabet**

*\$b64alpha = "3bcd1fghijklmABCDEFGH-  
J+LMnopq4stuvwxyzNOPQ7STUVWXYZ0e2ar56R89K/" wide ascii*



# You're Crazy!!










# Trojan Loader

- Side Loaded
- Small File Size
- Very Little Executable Code



# Trojan Loader (Basic Analysis)

File: msi.dll  
Size: 3584  
MD5:  
9530B64683D7397D081D538C46C4314E  
Compiled: Fri, Mar 13 2015, 15:35:47 - 32 Bit  
DLL

Function name	
	sub_10001000
	sub_10001110
	sub_10001180
	sub_100016B0
	<b>DllEntryPoint</b>

## Ascii Strings:

0000004D !This program cannot be run in  
DOS mode.  
000001DF ` .rdata  
00000207 @.data  
00000230 .reloc  
00000F16 GetProcAddress  
00000F28 LoadLibraryA  
00000F36 KERNEL32.dll  
00000F46 LineTo  
00000F50 MoveToEx  
00000F5A GDI32.dll

## Unicode Strings:

00000E54 msi.dll.en



# Trojan Loader Rule 1

```
rule PlugX_TrojanLoader_PayloadNames
{
    strings:
        $str1 = "msi.dll.eng" wide fullword
    condition:
        (uint16(0) == 0x5A4D) and //Check for MZ at offset 0
        any of them
}
```



# Reduce False Positives w/Conditions

```
rule PlugX_TrojanLoader_PayloadNames
{
    strings:
        $str1 = "msi.dll.eng" wide fullword
    condition:
        (uint16(0) == 0x5A4D) and //Check for MZ at offset 0
        (filesize < 11KB) and
        any of them
}
```



# Trojan Loader Rule 2 (Byte Signature)

```
8B 45 F4
8A 0C 38
FF 05 00 30 00 10
2A CB
80 F1 3F
02 CB
6A 00
88 0F
```

```
mov     eax, [ebp+var_C]
mov     cl, [eax+edi]
inc     dword_10003000
sub     cl, bl
xor     cl, 3Fh
add     cl, bl
push   0
mov     [edi], cl
```



# X86 Disassembly (On the Brief)

Opcodes <->  
Operand(s)

Mnemonic

80 F1 3F

xor

cl, 3Fh



# <http://ref.x86asm.net/coder32.html>

80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F  
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF  
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF  
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF

<u>pf</u>	<u>0F</u>	<u>po</u>	<u>so</u>	<u>o</u>	<u>proc</u>	<u>st</u>	<u>m</u>	<u>rl</u>	<u>x</u>	<u>mnemonic</u>	<u>op1</u>	<u>op2</u>	<u>op3</u>	<u>op4</u>	<u>iext</u>	<u>tested f</u>	<u>mod</u>
		00		r					L	ADD	<b>r/m8</b>	r8					o..sza
		01		r					L	ADD	<b>r/m16/32</b>	r16/32					o..sza
		02		r						ADD	<b>r8</b>	r/m8					o..sza
		03		r						ADD	<b>r16/32</b>	r/m16/32					o..sza
		04								ADD	<b>AL</b>	imm8					o..sza
		05								ADD	<b>eAX</b>	imm16/32					o..sza
		06								PUSH	<b>ES</b>						
		07								POP	<b>ES</b>						
		08		r					L	OR	<b>r/m8</b>	r8					o..sza
		09		r					L	OR	<b>r/m16/32</b>	r16/32					o..sza
		0A		r						OR	<b>r8</b>	r/m8					o..sza



# Byte Sequence Rule Steps

1. Wildcard Operands
2. Consolidate Wildcards with Jumps
3. Pad Jumps for Larger Constants
4. Scan Malware Repository for Other Samples
  1. Adjust for Alternate Opcodes
    - Consult the Opcode table
    - Adjust using Lower Nibble Wildcards or Alternatives
  2. Identify Presence of Junk Code
    - If present, pad jumps for junk code
    - Consolidate Double Jumps
  3. Slightly De-optimize if Needed
    - Re-Insert Higher Nibble Operands for Lower Nibble Wildcards
    - Re-Insert some Operand bytes for Addresses Locations





# Wildcard Operands

```
rule APTGroupX_PlugXTrojanLoader_StringDecode
{
    strings:
    $byte = {
        8B 45 F4 //mov    eax, [ebp+var_C]
        8A 0C 38 //mov    cl, [eax+edi]
        FF 05 00 30 00 10 //inc
    dword_10003000
        2A CB //sub    cl, bl
        80 F1 3F //xor    cl, 3Fh
        02 CB //add    cl, bl
        6A 00 //push  0
        88 0F //mov    [edi], cl
    }
    condition:
        any of them
}
```

```
rule APTGroupX_PlugXTrojanLoader_StringDecode
{
    strings:
    $byte = {
        8B ?? ?? //mov    eax, [ebp+var_C]
        8A ?? ?? //mov    cl, [eax+edi]
        FF 05 ?? ?? ?? ?? //inc
    dword_10003000
        2A ?? //sub    cl, bl
        80 ?? ?? //xor    cl, 3Fh
        02 ?? //add    cl, bl
        6A ?? //push  0
        88 ?? //mov    [edi], cl
    }
    condition:
        any of them
}
```



# Consolidate Wildcards w/Jumps

```
$byte = {  
    8B ?? ?? //mov    eax,  
[ebp+var_C]  
    8A ?? ?? //mov    cl, [eax+edi]  
    FF 05 ?? ?? ?? ?? //inc  
dword_10003000  
    2A ?? //sub    cl, bl  
    80 ?? ?? //xor    cl, 3Fh  
    02 ?? //add    cl, bl  
    6A ?? //push  0  
    88 ?? //mov    [edi], cl  
}
```

```
$byte = {  
    8B [2] //mov    eax,  
[ebp+var_C]  
    8A [2] //mov    cl, [eax+edi]  
    FF 05 [4] //inc  
dword_10003000  
    2A [1] //sub    cl, bl  
    80 [2] //xor    cl, 3Fh  
    02 [1] //add    cl, bl  
    6A ?? //push  0  
    88 ?? //mov    [edi], cl  
}
```



# Pad Jumps for Larger Constants

```
$byte = {
    8B [2] //mov eax,
[ebp+var_C]
    8A [2] //mov cl, [eax+edi]
    FF 05 [4] //inc dword_10003000
    2A [1] //sub cl, bl
    80 [2] //xor cl, 3Fh
    02 [1] //add cl, bl
    6A ?? //push 0
    88 ?? //mov [edi], cl
}

$byte = {
    8B [2] //mov eax, [ebp+var_C]
    8A [2] //mov cl, [eax+edi]
    FF 05 [4] //inc dword_10003000
    2A [1] //sub cl, bl
    80 [2-5] //xor cl, 3Fh
    02 [1] //add cl, bl
    6A [1-4] //push 0
    88 ?? //mov [edi], cl
}
```



# Look for Alternate Opcodes

## Sample #1

```
8B 45 F4      mov  eax, [ebp+var_C]
8A 0C 38      mov  cl, [eax+edi]
FF 05 00 30 00 10  inc  dword_10003000
2A CB        sub  cl, bl
80 F1 3F      xor  cl, 3Fh
02 CB        add  cl, bl
6A 00        push 0
88 0F        mov  [edi], cl
```

## Sample #2

```
8A 0C 18      mov  cl, [eax+ebx]
8A 45 10      mov  al, [ebp+arg_8]
FF 05 00 30 00 10  inc  dword_10003000
6A 00        push 0
2A C8        sub  cl, al
6A 00        push 0
80 F1 3F      xor  cl, 3Fh
6A 00        push 0
02 C8        add  cl, al
6A 00        push 0
88 0B        mov  [ebx], cl
```



# Consult the Opcode Table

	83		6	03+				L	XOR	<b>r/m16/32</b>	imm8
	83		7						CMP	r/m16/32	imm8
	84		r						TEST	r/m8	r8
	85		r						TEST	r/m16/32	r16/32
	86		r					L	XCHG	<b>r8</b>	<b>r/m8</b>
	87		r					L	XCHG	<b>r16/32</b>	<b>r/m16/32</b>
	88		r						MOV	<b>r/m8</b>	r8
	89		r						MOV	<b>r/m16/32</b>	r16/32
	8A		r						MOV	<b>r8</b>	r/m8
	8B		r						MOV	<b>r16/32</b>	r/m16/32
	8C		r						MOV	<b>m16</b>	Sreg
									MOV	<b>r16/32</b>	Sreg



# Adjust for Alternate Opcodes

```
$byte = {
    8B [2] //mov eax,
[ebp+var_C]
    8A [2] //mov cl, [eax+edi]
    FF 05 [4] //inc
dword_10003000
    2A [1] //sub cl, bl
    80 [2-5] //xor cl, 3Fh
    02 [1] //add cl, bl
    [0-5] //push 0
    88 ?? //mov [edi], cl
}

$byte = {
    (8A|8B) [2] //mov eax, [ebp+var_C]
    8A [2] //mov cl, [eax+edi]
    FF 05 [4] //inc dword_10003000
    2A [1] //sub cl, bl
    80 [2-5] //xor cl, 3Fh
    02 [1] //add cl, bl
    [0-5] //push 0
    88 ?? //mov [edi], cl
}
```



# Junk Code

## Sample #1

```
8B 45 F4      mov    eax,  
[ebp+var_C]  
8A 0C 38      mov    cl, [eax+edi]  
FF 05 00 30 00 10  inc    dword_10003000  
2A CB        sub    cl, bl  
80 F1 3F      xor    cl, 3Fh  
02 CB        add    cl, bl  
6A 00        push  0  
88 0F        mov    [edi], cl
```

## Sample #2

```
8A 0C 18      mov    cl, [eax+ebx]  
8A 45 10      mov    al,  
[ebp+arg_8]  
FF 05 00 30 00 10  inc    dword_10003000  
6A 00        push  0  
2A C8        sub    cl, al  
6A 00        push  0  
80 F1 3F      xor    cl, 3Fh  
6A 00        push  0  
02 C8        add    cl, al  
6A 00        push  0  
88 0B        mov    [ebx], cl
```



# Pad Jumps for Junk Code

```
$byte = {
    (8A|8B) [2]    //mov    eax, [ebp+var_C]
    8A [2]        //mov    cl, [eax+edi]
    FF 05 [4]     //inc
dword_10003000
    6A [1-4]      //push  0
    2A [1]        //sub   cl, bl
    6A [1-4]      //push  0
    80 [2-5]     //xor   cl, 3Fh
    6A [1-4]      //push  0
    02 [1]        //add   cl, bl
    6A [1-4]      //push  0
    88 ??        //mov   [edi], cl
}

$byte = {
    (8A|8B) [2]    //mov    eax, [ebp+var_C]
    8A [2]        //mov    cl, [eax+edi]
    FF 05 [4]     //inc
dword_10003000
    [0-5]        //push  0
    2A [1]        //sub   cl, bl
    [0-5]        //push  0
    80 [2-5]     //xor   cl, 3Fh
    [0-5]        //push  0
    02 [1]        //add   cl, bl
    [0-5]        //push  0
    88 ??        //mov   [edi], cl
}
```





# Consolidate Double Jumps

```
$byte = {
    (8A|8B) [2]    //mov    eax, [ebp+var_C]
    8A [2]        //mov    cl, [eax+edi]
    FF 05 [4]     //inc
dword_10003000
    [0-5]        //push  0
    2A [1]       //sub    cl, bl
    [0-5]       //push  0
    80 [2-5]    //xor    cl, 3Fh
    [0-5]      //push  0
    02 [1]     //add    cl, bl
    [0-5]     //push  0
    88 0?     //mov    [edi], cl
}

$byte = {
    (8A|8B) [2]    //mov    eax, [ebp+var_C]
    8A [2]        //mov    cl, [eax+edi]
    FF 05 [4-9]   //inc    dword_10003000
    [0-5]     //push  0
    2A [1-6]     //sub    cl, bl
    [0-5]     //push  0
    80 [2-7]    //xor    cl, 3Fh
    [0-5]     //push  0
    02 [1-6]    //add    cl, bl
    [0-5]     //push  0
    88 0?     //mov    [edi], cl
}
```



# Warning from YARA or VT Hunting

PlugX

Save changes

Enabled

Disabled

\$byte1 is slowing down scanning

```
140         any of them
141     }
142 rule TBHK_Campaign_PlugX_Trojan_Loader_08
143 {
144     strings:
145         $byte = { 8B 45 F4 [0-2] 8A 0? 38 [0-2] FF 05 00 30 00 10 [0-7] 2A [1-6] 80 [2-7] 02 [1-6] 88 }
146     condition:
147         any of them
148 }
149 rule TBHK_Campaign_PlugX_Trojan_Loader_09
150 {
151     strings:
152         $byte1 = { (8B|8A) [2-4] 8A [2-4] FF 05 [4-11] 2A [1-6] 80 [2-7] 02 [1-6] 88 }
153         $byte2 = { (8B|8A) [2-4] 8A [2-4] FF 05 (28|40) 30 00 10 [0-7] 2A [1-6] 80 [2-7] 02 [1-6] 88 }
154     condition:
155         any of them
156 }
```



# Look for Similar Address Locations

## Sample #1

```
8B 45 F4      mov    eax,  
[ebp+var_C]  
8A 0C 38      mov    cl, [eax+edi]  
FF 05 00 30 00 10  inc  
dword_10003000  
2A CB        sub    cl, bl  
80 F1 3F      xor    cl, 3Fh  
02 CB        add    cl, bl  
6A 00        push  0  
88 0F        mov    [edi], cl
```

## Sample #2

```
8A 0C 18      mov    cl, [eax+ebx]  
8A 45 10      mov    al,  
[ebp+arg_8]  
FF 05 00 30 00 10  inc  
dword_10003000  
6A 00        push  0  
2A C8        sub    cl, al  
6A 00        push  0  
80 F1 3F      xor    cl, 3Fh  
6A 00        push  0  
02 C8        add    cl, al  
6A 00        push  0  
88 0B        mov    [ebx], cl
```



# Look for Higher Nibble Situations

## Sample #1

```
8B 45 F4      mov    eax,  
[ebp+var_C]  
8A 0C 38      mov    cl, [eax+edi]  
FF 05 00 30 00 10  inc  
dword_10003000  
2A CB        sub    cl, bl  
80 F1 3F      xor    cl, 3Fh  
02 CB        add    cl, bl  
6A 00        push  0  
88 0F        mov    [edi], cl
```

## Sample #2

```
8A 0C 18      mov    cl, [eax+ebx]  
8A 45 10      mov    al,  
[ebp+arg_8]  
FF 05 00 30 00 10  inc  
dword_10003000  
6A 00        push  0  
2A C8        sub    cl, al  
6A 00        push  0  
80 F1 3F      xor    cl, 3Fh  
6A 00        push  0  
02 C8        add    cl, al  
6A 00        push  0  
88 0B        mov    [ebx], cl
```



# Lower Order Nibble Wildcards

\$byte = {		\$byte = {	
8B [2]	//mov eax, [ebp+var_C]	(8A   8B) [2]	//mov eax, [ebp+var_C]
8A [2]	//mov cl, [eax+edi]	8A [2]	//mov cl, [eax+edi]
FF 05 [4]	//inc	FF 05 [4]	//inc
dword_10003000		dword_10003000	
[0-5]	//push 0	[0-5]	//push 0
2A [1]	//sub cl, bl	2A [1]	//sub cl, bl
[0-5]	//push 0	[0-5]	//push 0
80 [2-5]	//xor cl, 3Fh	80 [2-5]	//xor cl, 3Fh
[0-5]	//push 0	[0-5]	//push 0
02 [1]	//add cl, bl	02 [1]	//add cl, bl
[0-5]	//push 0	[0-5]	//push 0
88 ??	//mov [edi], cl	88 0?	//mov [edi], cl
}		}	



# Trojan Loader Rule 2

rule APTGroupX\_PlugXTrojanLoader\_StringDecode

```
{
  strings:
  $byte = {
    (8B|8A) [2-4]    // mov   cl, [eax+ebx]
    8A [2-4]        // mov   al, [ebp+arg_8]
    FF 05 00 30 00 10 // inc   dword_10003000
    [0-5]           // <junk_holder>
    2A [1-6]        // sub   cl, al
    80 [2-7]        // xor   cl, 3Fh
    02 [1-6]        // add   cl, al
    88 0?           // mov   [ebx], cl
  }
  condition:
    any of them
}
```

```
call edi ; LineTo
mov   eax, [ebp+var_C]
mov   cl, [eax+ebx]
mov   al, [ebp+arg_8]
inc   dword_10003000
push  0 ; lppt
sub   cl, al
push  0 ; y
xor   cl, 3Fh
push  0 ; x
add   cl, al
push  0 ; hdc
mov   [ebx], cl
call  esi ; MoveToEx
```



# Byte Sequence Rule Steps

1. Wildcard Operands
2. Consolidate Wildcards with Jumps
3. Pad Jumps for Larger Constants
4. Scan Malware Repository for Other Samples
  - Adjust for Alternate Opcodes
    - Consult the Opcode table
    - Adjust using Lower Nibble Wildcards or Alternatives
  - Identify Presence of Junk Code
    - If present, pad jumps for junk code
    - Consolidate Double Jumps
  - Slightly De-optimize if Needed
    - Re-Insert Higher Nibble Operands for Lower Nibble Wildcards
    - Re-Insert some Operand bytes for Addresses Locations



# More Payload Names

Mc.cp

setup.msi

fslapi.dll.gui

set.conf

McUtil.dll.mc

player.db

msi.dll.eng

MSO.dsm

SXLOC.ZAP

SiteAdv.adv

http.dlp

Pmcutil.dll.bbc

mcf.ep

splash\_screen.dll.sky

httpwin.dat

dot1x.1x

msi.dll.kav

FSMA32.dllfox

McUtil.dll.ping

moic.exe.dat

FSPMAPI.dll.fsp

Samsung.hlp

kav.avp

ssMUIDLL.dll.conf

demo.dat

readme.txt

rapi.dll.rap





# PlugX Payload



```
seg000:00000000          mov     ebx, 36FBDCB2h
seg000:00000005          jmp     loc_B
seg000:00000005          ; -----
seg000:0000000A          db 0E9h ; T
seg000:0000000B          ; -----
seg000:0000000B
seg000:0000000B          loc_B:          ; CODE XREF:
seg000:00000005j
seg000:0000000B          add     ecx, 0ED43AD3Fh
seg000:00000011          inc     ecx
seg000:00000012          sub     eax, 0A38A7DCCh
seg000:00000017          xor     eax, 441AC7BAh
seg000:0000001C          dec     edx
seg000:0000001D          cmp     eax, 0FA629847h
seg000:00000022          mov     eax, [esp]
seg000:00000025          cmp     edx, 0B0A968D3h
seg000:0000002B          and     edx, 513AB2C1h
```



# Inside Memory

Region00AB0000-00AD7000.dmp

Edit As: Hex Run Script Run Template

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000h:	58	56	00	00	00	00	00	00	00	00	00	00	00	00	00	00	XV	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0010h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	D8	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0040h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0050h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0060h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0070h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
00A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
00C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
00D0h:	00	00	00	00	00	00	00	00	00	00	00	00	58	56	00	00	XV	.	.	.	.	.	.	.	.	.	.	.	.	.	.	

# To Compare

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....ÿÿ..
0010h:	80	80	00	00	00	00	00	00	40	00	00	00	00	00	00	00	.....@.....
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	D8	00	00	00	.....@...
0040h:	DE	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	00	00	00	00	..°...í!..
0050h:	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
0060h:	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
0070h:	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....\$.....
0080h:	97	A6	B0	91	D3	C7	DE	C2	D3	C7	DE	C2	D3	C7	DE	C2	-!°'óçPâóçPâóçPâ
0090h:	10	C8	D1	C2	D7	C7	DE	C2	D3	C7	DF	C2	48	C5	DE	C2	.ÈÑÀ×çPâóçPâHâPâ
00A0h:	10	C8	83	C2	C8	C7	DE	C2	10	C8	80	C2	D2	C7	DE	C2	.ÈfâÈçPâ.ÈÈâóçPâ
00B0h:	10	C8	BE	C2	FA	C7	DE	C2	10	C8	81	C2	CE	C7	DE	C2	.È%âúçPâ.È.âîçPâ
00C0h:	10	C8	84	C2	D2	C7	DE	C2	52	69	63	68	D3	C7	DE	C2	.È„âóçPâRíçPâ
00D0h:	00	00	00	00	00	00	00	00	50	45	00	00	4C	01	04	00	.....PE.L...



# Walking Structures

Region00AB0000-00AD7000.dmp

Edit As: Hex Run Script Run Template

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000h:	58	56	00	00	00	00	00	00	00	00	00	00	00	00	00	00	XV	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0010h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	D8	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0040h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0050h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0060h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0070h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
0090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
00A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
00C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
00D0h:	00	00	00	00	00	00	00	00	00	00	00	00	58	56	00	00	XV	.	.	.	.	.	.	.	.	.	.	.	.	.	.	

# Modified MZ & PE Headers

```
rule UNKNOWN_PlugXPayload_XVHeader{
```

```
    meta:
```

```
        source
```

```
        ="D9AB2B14E9B2F1D78C117FDB1BF0601E"
```

```
    condition:
```

```
        uint16(0) == 0x5658 and //Check for XV at offset 0
```

```
        uint16(uint32(0x3C)) == 0x5658 //Check for XV at  
        pointer offset
```

```
}
```



# PE Characteristics Bit Flags

```
rule PlugX_TrojanLoader_PayloadNames
{
    strings:
        $str1 = "msi.dll.eng" wide fullword
    condition:
        //Check PE Characteristics Bit Flags
        (((uint16(uint32(0x3C)+0x16))&0x2002) == 0x2002) and
        (filesize < 11KB) and
        any of them
}
```



# YARA House Keeping

- **Rules, Rules, Rules**
- Signature All Layers
- Naming Scheme
- Tuning
- Data Sources
  - Known Files
  - Malware
  - New Intelligence



# String Rules



We LOVE Unicode

Reduce FPs for String Rules

- wide ascii nocase fullword
- “\nuname\n\n”
- (filesize <11KB)





# Byte Sequences & Other Rules

Consult an Opcode table

Identify junk code

Look for opportunities to use

( 8A | 8B )

88 0?

Make use of Structures

Don't forget checking Bit Flags



# Resources

## GitHub - fideliscyber

Source MD5s:

- 24FC5871407F180ECAD9DA6F67DD1878
- D9AB2B14E9B2F1D78C117FDB1BF0601E

<https://yara.readthedocs.org/>

<http://ref.x86asm.net/coder32.html>

<http://community.websense.com/blogs/securitylabs/archive/2015/04/24/opportunity-knows-no-boundary-a-case-study-of-acquisition.asp>

<http://researchcenter.paloaltonetworks.com/2015/05/pluginx-uses-legitimate-samsung-application-for-dll-side-loading/>

<https://www.blackhat.com/docs/webcast/08202015-big-game-hunting.pdf>



# END

## Questions ?

