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

Posted: 24 Apr 2015 10:35 AM | uwang | no comments

Palo Alto Networks

Plugx Uses Legitimate Samsung Application for DLL Side-Loading

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
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
        all of them
}
Tuning String Rules

wide and fullword

$str1 = “setup.msi” wide fullword

casenon

// will hit on “KeRnEl32.DlL”

$str2 = “kernel32.dll” wide ascii casenon

whitespace characters

$str4 = “\nuname\n\n” wide ascii
<table>
<thead>
<tr>
<th>ASCII/8859-1 Text</th>
<th>Unicode Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0000 0000 0100 0001</td>
</tr>
<tr>
<td>S</td>
<td>0000 0000 0101 0011</td>
</tr>
<tr>
<td>C</td>
<td>0000 0000 0100 0011</td>
</tr>
<tr>
<td>I</td>
<td>0000 0000 0100 1001</td>
</tr>
<tr>
<td>I</td>
<td>0000 0000 0100 1001</td>
</tr>
<tr>
<td>8</td>
<td>0000 0000 0100 1000</td>
</tr>
<tr>
<td>8</td>
<td>0000 0000 0100 1000</td>
</tr>
<tr>
<td>5</td>
<td>0000 0000 0110 0001</td>
</tr>
<tr>
<td>9</td>
<td>0000 0000 0110 0001</td>
</tr>
<tr>
<td>-</td>
<td>0000 0000 0110 1101</td>
</tr>
<tr>
<td>t</td>
<td>0000 0000 0111 0001</td>
</tr>
<tr>
<td>e</td>
<td>0000 0000 0110 0101</td>
</tr>
<tr>
<td>x</td>
<td>0000 0000 0111 1000</td>
</tr>
<tr>
<td>t</td>
<td>0000 0000 0111 0100</td>
</tr>
</tbody>
</table>
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

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.eng
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          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
```
**X86 Disassembly (On the Brief)**

<table>
<thead>
<tr>
<th>Opcodes</th>
<th>Mnemonic</th>
<th>Operand(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 F1 3F</td>
<td>xor</td>
<td>cl, 3Fh</td>
</tr>
<tr>
<td>pf</td>
<td>0F</td>
<td>po</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>00</td>
<td>r</td>
<td>r</td>
</tr>
<tr>
<td>01</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>02</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>03</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>04</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>05</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>06</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>07</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>08</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>09</td>
<td>r</td>
<td>L</td>
</tr>
<tr>
<td>0A</td>
<td>r</td>
<td>OR</td>
</tr>
</tbody>
</table>
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

```c
$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
}
```

```c
$byte = { 
  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

```c
$byte = {
    FF 05 [4] //inc DWORD_10003000
    2A [1] //sub cl, bl
    80 [2-5] //xor cl, 3Fh
    02 [1] //add cl, bl
    6A ?? //push 0
    88 ?? //mov [edi], cl
};
```

```c
$byte = {
    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
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

Sample #2

8A 0C 18
8A 45 10
FF 05 00 30 00 10
6A 00
2A C8
6A 00
80 F1 3F
6A 00
02 C8
6A 00
88 0B

mov    cl, [eax+ebx]
mov    al, [ebp+arg_8]
inc    dword_10003000
push   0
sub    cl, al
push   0
xor    cl, 3Fh
push   0
add    cl, al
push   0
mov    [ebx], cl
# Consult the Opcode Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Function</th>
<th>Register</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>83 6 03+</td>
<td>L XOR</td>
<td>r/m16/32</td>
<td>imm8</td>
<td></td>
</tr>
<tr>
<td>83 7</td>
<td>CMP</td>
<td>r/m16/32</td>
<td>imm8</td>
<td></td>
</tr>
<tr>
<td>84 r</td>
<td>TEST</td>
<td>r/m8</td>
<td>r8</td>
<td></td>
</tr>
<tr>
<td>85 r</td>
<td>TEST</td>
<td>r/m16/32</td>
<td>r16/32</td>
<td></td>
</tr>
<tr>
<td>86 r</td>
<td>L XCHG</td>
<td>r8</td>
<td>r/m8</td>
<td></td>
</tr>
<tr>
<td>87 r</td>
<td>L XCHG</td>
<td>r16/32</td>
<td>r/m16/32</td>
<td></td>
</tr>
<tr>
<td>88 r</td>
<td>MOV</td>
<td>r/m8</td>
<td>r8</td>
<td></td>
</tr>
<tr>
<td>89 r</td>
<td>MOV</td>
<td>r/m16/32</td>
<td>r16/32</td>
<td></td>
</tr>
<tr>
<td>8A r</td>
<td>MOV</td>
<td>r8</td>
<td>r/m8</td>
<td></td>
</tr>
<tr>
<td>8B r</td>
<td>MOV</td>
<td>r16/32</td>
<td>r/m16/32</td>
<td></td>
</tr>
<tr>
<td>8C r</td>
<td>MOV</td>
<td>m16</td>
<td>Sreg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>r16/32</td>
<td>Sreg</td>
<td></td>
</tr>
</tbody>
</table>
Adjust for Alternate Opcodes

$byte = {
  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]
  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 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 00 30 00 10` 
  `inc` `dword_10003000`
- `6A 00` 
  `push` `0`
- `2A C8` 
  `sub` `cl, al`
- `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

```
```
Consolidate Double Jumps

$\text{byte} = \{
(8A|8B) [2] //\text{mov} \ \text{eax}, [ebp+var}_C\}
8A [2] //\text{mov} \ \text{cl}, [eax+edi]  
\text{FF} \ \text{05} [4] //\text{inc} 
\text{dword}_10003000
[0-5] //\text{push} \ 0 
2A [1] //\text{sub} \ cl, bl 
[0-5] //\text{push} \ 0 
80 [2-5] //\text{xor} \ cl, 3Fh 
[0-5] //\text{push} \ 0 
02 [1] //\text{add} \ cl, bl 
[0-5] //\text{push} \ 0 
88 \ 0? //\text{mov} \ [edi], cl 
\}

$\text{byte} = \{
(8A|8B) [2] //\text{mov} \ \text{eax}, [ebp+var}_C\}
8A [2] //\text{mov} \ \text{cl}, [eax+edi]  
\text{FF} \ \text{05} [4-9] //\text{inc} \ \text{dword}_10003000
[0-5] //\text{push} \ 0 
2A [1-6] //\text{sub} \ cl, bl 
[0-5] //\text{push} \ 0 
80 [2-7] //\text{xor} \ cl, 3Fh 
[0-5] //\text{push} \ 0 
02 [1-6] //\text{add} \ cl, bl 
[0-5] //\text{push} \ 0 
88 \ 0? //\text{mov} \ [edi], cl 
\}
Warning from YARA or VT Hunting

PlugX

Any of them

rule TBHK_Campaign_PlugX_Trojan_Loader_08
{
    strings:
    condition:
    any of them
}

rule TBHK_Campaign_PlugX_Trojan_Loader_09
{
    strings:
    condition:
    any of them
}
Look for Similar Address Locations

Sample #1

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8B 45 F4 [ebp+var_C]</td>
<td>mov eax, [ebp+var_C]</td>
<td></td>
</tr>
<tr>
<td>8A 0C 38</td>
<td>mov cl, [eax+edi]</td>
<td></td>
</tr>
<tr>
<td>FF 05 00 30 00 10 dword_10003000</td>
<td>inc</td>
<td></td>
</tr>
<tr>
<td>2A CB</td>
<td>sub cl, bl</td>
<td></td>
</tr>
<tr>
<td>80 F1 3F</td>
<td>xor cl, 3Fh</td>
<td></td>
</tr>
<tr>
<td>02 CB</td>
<td>add cl, bl</td>
<td></td>
</tr>
<tr>
<td>6A 00</td>
<td>push 0</td>
<td></td>
</tr>
<tr>
<td>88 0F</td>
<td>mov [edi], cl</td>
<td></td>
</tr>
</tbody>
</table>

Sample #2

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8A 0C 18</td>
<td>mov cl, [eax+ebx]</td>
<td></td>
</tr>
<tr>
<td>8A 45 10 [ebp+arg_8]</td>
<td>mov al, [ebp+arg_8]</td>
<td></td>
</tr>
<tr>
<td>FF 05 00 30 00 10 dword_10003000</td>
<td>inc</td>
<td></td>
</tr>
<tr>
<td>6A 00</td>
<td>push 0</td>
<td></td>
</tr>
<tr>
<td>2A C8</td>
<td>sub cl, al</td>
<td></td>
</tr>
<tr>
<td>6A 00</td>
<td>push 0</td>
<td></td>
</tr>
<tr>
<td>80 F1 3F</td>
<td>xor cl, 3Fh</td>
<td></td>
</tr>
<tr>
<td>6A 00</td>
<td>push 0</td>
<td></td>
</tr>
<tr>
<td>02 C8</td>
<td>add cl, al</td>
<td></td>
</tr>
<tr>
<td>6A 00</td>
<td>push 0</td>
<td></td>
</tr>
<tr>
<td>88 0B</td>
<td>mov [ebx], cl</td>
<td></td>
</tr>
</tbody>
</table>
## Look for Higher Nibble Situations

### Sample #1

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>8B 45 F4</td>
<td><code>mov</code> eax, [ebp+var_C]</td>
<td>FF 05 00 30 00 10</td>
</tr>
<tr>
<td>8A 0C 38</td>
<td><code>mov</code> cl, [eax+edi]</td>
<td>2A CB</td>
</tr>
<tr>
<td>FF 05 00 30 00 10</td>
<td><code>inc</code></td>
<td>80 F1 3F</td>
</tr>
<tr>
<td>dword_10003000</td>
<td></td>
<td>02 CB</td>
</tr>
<tr>
<td>6A 00</td>
<td><code>push</code> 0</td>
<td>6A 00</td>
</tr>
<tr>
<td>88 0F</td>
<td><code>mov</code> [edi], cl</td>
<td>80 F1 3F</td>
</tr>
</tbody>
</table>

### Sample #2

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>8A 0C 18</td>
<td><code>mov</code> cl, [eax+ebx]</td>
<td>FF 05 00 30 00 10</td>
</tr>
<tr>
<td>8A 45 10</td>
<td><code>mov</code> al,</td>
<td>2A C8</td>
</tr>
<tr>
<td>[ebp+arg_8]</td>
<td></td>
<td>80 F1 3F</td>
</tr>
<tr>
<td>inc</td>
<td></td>
<td>02 C8</td>
</tr>
<tr>
<td>6A 00</td>
<td><code>push</code> 0</td>
<td>6A 00</td>
</tr>
<tr>
<td>push 0</td>
<td></td>
<td>80 F1 3F</td>
</tr>
<tr>
<td>xor cl, 3Fh</td>
<td></td>
<td>02 C8</td>
</tr>
<tr>
<td>add cl, bl</td>
<td></td>
<td>6A 00</td>
</tr>
<tr>
<td>mov [ebx], cl</td>
<td></td>
<td>02 C8</td>
</tr>
<tr>
<td>push 0</td>
<td></td>
<td>6A 00</td>
</tr>
<tr>
<td>mov [ebx], cl</td>
<td></td>
<td>88 0B</td>
</tr>
</tbody>
</table>
Lower Order Nibble Wildcards

$byte = {
  FF 05 [4] //inc
  dword_10003000
}

$byte = {
  (8A | 8B) [2] //mov  eax, [ebp+var_C]
  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

[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
Trojan Loader Rule 2

rule APTGroupX_PlugXTrojanLoader_StringDecode {
  strings:
  $byte = {
    (8B|8A) [2-4] // mov cl, [eax+ebx]
    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
}
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
seg000:00000005
seg000:00000005
seg000:0000000A
seg000:0000000B
seg000:0000000B
seg000:0000000B
seg000:000000B
seg000:000000B
seg000:000000B
seg000:0000011
seg000:0000012
seg000:0000017
seg000:000001C
seg000:000001D
seg000:0000022
seg000:0000025
seg000:000002B

mov   ebx, 36FBDCB2h
jmp    loc_B

; ---------------------------------;
; db 0E9h ; T
; ---------------------------------;

loc_B:   ; CODE XREF:

add    ecx, 0ED43AD3Fh
inc    ecx
sub    eax, 0A38A7DCCh
xor    eax, 441AC7BAh
dec    edx
cmp    eax, 0FA629847h
mov    eax, [esp]
cmp    edx, 0B0A968D3h
and    edx, 513AB2C1h
Inside Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex Value</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>58 56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0020h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0030h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0040h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0050h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0060h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0070h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0080h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0090h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00A0h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00B0h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00C0h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00D0h</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Red boxes highlight specific values in memory.
To Compare

<table>
<thead>
<tr>
<th>Explorer.exe</th>
<th>Edit As: Hex</th>
<th>Run Script</th>
<th>Run Template</th>
<th>0123456789ABCDEF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0001h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0002h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0003h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0004h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0005h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0006h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0007h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0008h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0009h:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>000Ah:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>000Bh:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>000Ch:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>000Dh:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| O0000h:       | 4D 5A        | 00 00 00 00 00 00 00 00 04 00 00 00 FF FF 00 00 | MZ ..........XX |
| O0010h:       | B8 88 00 00 00 00 00 00 00 40 00 00 00 00 00 00 | .............0 |
| O0020h:       | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O0030h:       | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O0040h:       | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O0050h:       | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O0060h:       | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O0070h:       | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O0080h:       | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O0090h:       | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O00Ah:        | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |
| O00Bh:        | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |                       |

D8 00 00 00

0...
## Walking Structures

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h:</td>
<td>58 56</td>
</tr>
<tr>
<td>0010h:</td>
<td>00 00</td>
</tr>
<tr>
<td>0020h:</td>
<td>00 00</td>
</tr>
<tr>
<td>0030h:</td>
<td>00 00</td>
</tr>
<tr>
<td>0040h:</td>
<td>00 00</td>
</tr>
<tr>
<td>0050h:</td>
<td>00 00</td>
</tr>
<tr>
<td>0060h:</td>
<td>00 00</td>
</tr>
<tr>
<td>0070h:</td>
<td>00 00</td>
</tr>
<tr>
<td>0080h:</td>
<td>00 00</td>
</tr>
<tr>
<td>0090h:</td>
<td>00 00</td>
</tr>
<tr>
<td>00A0h:</td>
<td>00 00</td>
</tr>
<tr>
<td>00B0h:</td>
<td>00 00</td>
</tr>
<tr>
<td>00C0h:</td>
<td>00 00</td>
</tr>
<tr>
<td>00D0h:</td>
<td>00 00</td>
</tr>
</tbody>
</table>

**Note:** The highlighted values indicate specific regions of interest.
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
}

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"
- (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
GitHub - fideliscyber

Source MD5s:

- 24FC5871407F180ECAD9DA6F67DD1878
- D9AB2B14E9B2F1D78C117FDB1BF0601E

https://yara.readthedocs.org/
http://ref.x86asm.net/coder32.html
http://researchcenter.paloaltonetworks.com/2015/05/plugx-uses-legitimate-samsung-application-for-dll-side-loading/
END

Questions ?