FLOSS Every Day

Automatically Extracting Obfuscated Strings from Malware
Malicious or not?

d41d8cd98f00b204e9800998ecf8427e
Malicious or not?

a5ca7e7281d8b8a570a529895106b1fa
Uh-oh

File not found

The file you are looking for is not in our database.

Take me back to the main page  Try another search
Next… IDA Pro?
Static analysis

- PE file format
  - Imports
  - Exports
  - Section metadata

- strings.exe
Static analysis

- PE file format
  - Imports
  - Exports
  - Section metadata
- strings.exe
Well this sucks…

```
strings a5ca7e7281d8b8a570a529895106b1fa | tr "\n" " "
```

This program cannot be run in DOS mode. Rich .text .rdata .data .CRT .rsrc .reloc Uint- t8Ht+Ht dt+Ht |vj.w Y_^[YY^[ YY^[ v]W wpw@wj wWwj j@wj PSSV t+w3 AG;} VSh( vsh0 vsh8 vsh@ vshH VShP HvshX VSh` vsh h u hp u$hX u(h` 8]8u u Ohh Hvsh 8]8u hb)! $v3 ssh0 $sj sjsPh8 PVj+ PWj, ssssshX YYhd Sh<' $sj sjsPh PVj+ PWj, sssssh YYhd svw3 towr t_sw 3 YG;~ t@sv ^[[]] YG;~ svw3 mWvh Y_^[ [wwj wsvp ;3Y^t_^[] swwj qqvwj ;fpu f9~8t t.w3 _^[[] ;fpu j tESw u{vhh Y_^[ svwj yy^[ t2v3 _^[[] ssssss sssssss yy8^ svv h Y_^[ qqsv <$xx (svw txhtqH t(Ht!Ht PPPP QPPP svwj+h PWWj j,h( qqsvwj
Time to bring in the big guns

- Reverse Engineer
Time to bring in the big guns

- Reverse Engineer
Time to bring in the big guns

- Reverse Engineer
- Everyone else
Time to bring in the big guns

- Reverse Engineer
- Everyone else
Introducing FLOSS

$ floss a5ca7e7281d8b8a570a529895106b1fa
/index.html
http://
POST
GET
User-Agent: FJUR (compatible; MSIE 6.0; Win32)
HOST:
Software\Microsoft\Windows\CurrentVersion\Run
%s\%s.txt
CONNECT %s:%d HTTP/1.1
SetFileAttributesA
#456234
About the speakers

Willi Ballenthin
@williballenthin

Moritz Raabe
@m_r_tz
Packed malware

Original executable

Packed executable

PE Headers

.text Section

.data Section

Executable code and data are compressed and obfuscated

PE Headers

Unpacking Stub

Packed Original Code

Original executable

Packed executable
Packed malware

Packed executable

Executable in memory

PE Headers
Unpacking Stub
Packed Original Code

Unpacking stub unpacks the data

PE Headers
.text Section
.data Section
Unpacking Stub
Packed malware

Packed executable

Unpacking stubunpacks the data

Executable in memory
String obfuscation

```bash
$ strings a5ca7e7281d8b8a570a529895106b1fa | tr "\n" " "
```

This program cannot be run in DOS mode. Rich .text `.rdata .CRT .rsrc .reloc Utilities t8Ht+Ht dt+Ht }Vj.w Y_^[YY[^]y]v]vW wpWj@wj wWwj j@wj PSSV t+w3 AG;} vSh(vSh0 vSh8 vSh@ vShH vShP HVSHx vSh` vShh u hp u$hX u(h`8]8u u Ohh HvSh 8]8u hb)! $sv3 ssh0 $ssj ssjPh8 PVj+ PWj, ssSssshy Yyhd Sh<' $ssj ssjPh PVj+ PWj, ssSssshy Yyhd SVW3 t0WR t_sw 3 YG;~ t@sv ^[[] YG;~ SVW3 MWVh Y_^[][WWj WSVP ;3Y^t _^[[]] SVWj QQVWj ;FPU f9~8t t.w3 _^[[]] ;FPUj tESw u{Vhh Y_^[svWj yy^[yy^[ t2v3 _^[[]] ssSssss ssSssss yy8^ ssVh Y_^[QQVSV $#XX (svW tXHtQH t(Ht!Ht PPPP QPPP SVWj+h PWWj j,h( QQSVWj
```
String obfuscation

```
lea    edx, [esp+1FD0h+var_1C8C]
push   edx    ; void *
push   offset aDhinsqnsQwiTbu ; "Dhisnqns~'QWI'Tbuqndb"
call   sub_401000
lea    eax, [esp+1FD8h+var_1C28]
push   eax    ; void *
push   offset aIhusbkDqdTbuqn ; "Ihusbk'DQD'Tbuqndb"
call   sub_401000
lea    ecx, [esp+1FE0h+var_1BC4]
push   ecx    ; void *
push   offset aIbstdubbiUbjhs ; "IbsTdubbi*Ubjhsb'Tbdruns~'Dknbis"
call   sub_401000
```
Reversing tactics for obfuscated strings

+ ★ OllyDbg
Reversing tactics for obfuscated strings

+ ![OllyDbg](image)

+ ![OllyDbg](image)

#DFIRSummit
Reversing tactics for obfuscated strings

+ 👽 OllyDbg

+ 👽 OllyDbg

+ Python
FireEye Labs Obfuscated String Solver
FLOSS

$ floss a5ca7e7281d8b8a570a529895106b1fa
/index.html
http://
POST
GET
User-Agent: FJUR (compatible; MSIE 6.0; Win32)
HOST:
Software\Microsoft\Windows\CurrentVersion\Run
%s\%s
.txt
CONNECT %s:%d HTTP/1.1
SetFileAttributesA
#456234
FLOSS: user-friendly

https://github.com/fireeye/flare-floss
FLOSS: advanced static analysis techniques

- Disassembly, emulation, CFG analysis, etc.

FLARE in a box
FLOSS: algorithm

1. Analyze control flow of malware
2. Use heuristics to find potential decoding routines
3. Extract arguments passed to decoding routines
4. Emulate decoding routines using extracted arguments
5. Diff memory state from before and after decoder emulation
6. Collect human-readable strings
FLOSS: algorithm

1. **Analyze control flow** of malware
2. Use heuristics to **find potential decoding routines**
3. Extract arguments passed to decoding routines
4. Emulate decoding routines using extracted arguments
5. Diff memory state from before and after decoder emulation
6. Collect human-readable strings
Analyze control flow

- FLOSS uses vivisect
  - Functions, cross references, code, and data
  - https://github.com/vivisect/vivisect
FLOSS: algorithm

1. Analyze control flow of malware
2. Use heuristics to find potential decoding routines
3. Extract arguments passed to decoding routines
4. Emulate decoding routines using extracted arguments
5. Diff memory state from before and after decoder emulation
6. Collect human-readable strings
Find potential decoding routines

- Heuristics express confidence in decoding routines
- Plugin-based system
  - Easy to extend FLOSS
- Most effective heuristics to date:
  - Tight loops
  - Non-zeroing XOR operation
  - Many code cross-references
Find potential decoding routines

```assembly
00401675
00401675
00401675 8B 44 24 08
00401679 8D 0C 02
0040167C 8A 04 02
0040167F 2A C2
00401681 34 07
00401683 8A D8
00401685 C0 EB 05
00401688 C0 E0 03
0040168B 0A D8
0040168D 42
0040168E 3B 54 24 0C
00401692 88 19
00401694 7C DF

loc_401675:

mov  eax, [esp+4+arg_0]
lea  ecx, [edx+eax]
mov  al, [edx+eax]
sub  al, dl
xor  al, 7
mov  bl, al
shr  bl, 5
shl  al, 3
or  bl, al
inc  edx
cmp  edx, [esp+4+arg_4]
mov  [ecx], bl
jl   short loc_401675
```
Find potential decoding routines

```assembly
mov    eax, [esp+4+arg_0]
lea    ecx, [edx+eax]
mov    al, [edx+eax]
sub    al, dl
xor    al, 7
mov    bl, al
shr    bl, 5
shl    al, 3
or     bl, al
inc    edx
cmp    edx, [esp+4+arg_4]
mov    [ecx], bl
```
Find potential decoding routines
FLOSS: algorithm

1. Analyze control flow of malware
2. Use heuristics to find potential decoding routines
3. Extract arguments passed to decoding routines
   a) Brute force emulate all code paths
   b) Snapshot emulator state (registers, memory)
4. Emulate decoding routines using extracted arguments
5. Diff memory state from before and after decoder emulation
6. Collect human-readable strings
FLOSS: algorithm

1. Analyze control flow of malware
2. Use heuristics to find potential decoding routines
3. Extract arguments passed to decoding routines
   a) Brute force emulate all code paths
   b) Snapshot emulator state (registers, memory)
4. Emulate decoding routines using extracted arguments
5. Diff memory state from before and after decoder emulation
6. Collect human-readable strings
FLOSS: algorithm

1. Analyze control flow of malware
2. Use heuristics to find potential decoding routines
3. Extract arguments passed to decoding routines
4. Emulate decoding routines using extracted arguments
5. Diff memory state from before and after decoder emulation
6. Collect human-readable strings
FLOSS: algorithm

1. Analyze control flow of malware
2. Use heuristics to find potential decoding routines
3. Extract arguments passed to decoding routines
4. Emulate decoding routines using extracted arguments
5. Diff memory state from before and after decoder emulation
6. Collect human-readable strings
FLOSS extracts stackstrings and static strings

**Stackstrings**

```
lea   edx, [ebp+var_2A]
mov   [ebp+var_2A], 'g'
mov   [ebp+var_29], 'o'
mov   [ebp+var_28], 'o'
mov   [ebp+var_27], 'd'
mov   [ebp+var_26], 'b'
mov   [ebp+var_25], 'y'
mov   [ebp+var_24], 'e'
mov   [ebp+var_23], '
mov   [ebp+var_22], 'w'
mov   [ebp+var_21], 'o'
mov   [ebp+var_20], 'r'
mov   [ebp+var_1F], 'l'
mov   [ebp+var_1E], 'd'
mov   [ebp+var_1D], '
```

**ASCII and UTF-16**

```
$ floss a5ca7e7281d8b8a570a529895106b1fa
[..]
!This program cannot be run in DOS mode.
Rich
.text
.rdata
@.data
.CRT
@.rsrc
@.reloc
UtI-
t8Ht+Ht
dt+Ht
lVj.W
Y_^[
YY_]
YY^]
v]VW
WPWj@Wj
```
FLOSS demos always work
Summary

- Extract strings from malware binaries
  - Obfuscated strings, stackstrings, and static strings
- Easy to use
- Requires very little investment and training

"FLOSS"
Thank you!

Contact
Moritz Raabe, @m_r_tz
moritz.raabe@fireeye.com

William Ballenthin, @williballenthin
william.ballenthin@fireeye.com