






Unveiling Malicious Behavior in Unknown Binaries

Tim Blazytko

 @mr_phrazer
 synthesis.to
 tim@blazytko.to

About Tim

- Chief Scientist, co-founder of emproof
- designs software protections for embedded devices
- trainer for (de)obfuscation and reverse engineering techniques



- ❓ Unknown Binary
- ☠ Malicious?
- ➔ Strategies & Heuristics

 Unknown Binary



Malicious?

Where to start?



Common Approaches

Signature Checks


```
$ yara rules.yar unknown_binary
```

```
escalate_priv
```

```
screenshot
```

```
win_registry
```

```
win_token
```

```
win_files_operation
```

```
CRC32_poly_Constant
```

```
RIPemd160_Constants
```

```
SHA1_Constants
```

```
Codoso_3
```

```
IsPE32
```

```
IsWindowsGUI
```

```
IsPacked
```

```
HasOverlay
```

```
HasDebugData
```

```
HasRichSignature
```

```
$ yara rules.yar unknown_binary
```

```
escalate_priv
```

```
screenshot
```

```
win_registry
```

```
win_token
```

```
win_files_operation
```

```
CRC32 poly Constant
```

pre-defined signatures

```
Codoso_3
```

```
IsPE32
```

```
IsWindowsGUI
```

```
IsPacked
```

```
HasOverlay
```

```
HasDebugData
```

```
HasRichSignature
```

```
$ yara rules.yar unknown_binary
```

```
escalate_priv
```

```
screenshot
```

```
win_registry
```

```
win_token
```

```
win_files_operation
```

```
CRC32_poly_Constant
```

```
RIPEMD160_Constants
```

```
SHA1_Constants
```

```
Codoso_3
```

```
IsPE32
```

```
IsWindowsGUI
```

```
IsPacked
```

```
HasOverlay
```

```
HasDebugData
```

```
HasRichSignature
```

```
$ yara rules.yar unknown_binary
```

```
escalate_priv
```

```
screenshot
```

```
win_registry
```

```
win_token
```

```
win_files_operation
```

```
CRC32 poly Constant
```

potential screen capture

```
Codoso_3
```

```
IsPE32
```

```
IsWindowsGUI
```

```
IsPacked
```

```
HasOverlay
```

```
HasDebugData
```

```
HasRichSignature
```

```
$ yara rules.yar unknown_binary
```

```
escalate_priv
```

```
screenshot
```

```
win_registry
```

```
win_token
```

```
win_files_operation
```

```
CRC32_poly_Constant
```

```
RIPemd160_Constants
```

```
SHA1_Constants
```

```
Codoso_3
```

```
IsPE32
```

```
IsWindowsGUI
```

```
IsPacked
```

```
HasOverlay
```

```
HasDebugData
```

```
HasRichSignature
```

```
$ yara rules.yar unknown_binary
```

```
escalate_priv
```

```
screenshot
```

```
win_registry
```

```
win_token
```

```
win_files_operation
```

```
CRC32 poly Constant
```

cryptographic algorithms

```
Codoso_3
```

```
IsPE32
```

```
IsWindowsGUI
```

```
IsPacked
```

```
HasOverlay
```

```
HasDebugData
```

```
HasRichSignature
```

```
$ yara rules.yar unknown_binary
```

```
escalate_priv
```

```
screenshot
```

```
win_registry
```

```
win_token
```

```
win_files_operation
```

```
CRC32_poly_Constant
```

```
RIPEMD160_Constants
```

```
SHA1_Constants
```

```
Codoso_3
```

```
IsPE32
```

```
IsWindowsGUI
```

```
IsPacked
```

```
HasOverlay
```

```
HasDebugData
```

```
HasRichSignature
```

```
$ yara rules.yar unknown_binary
```

```
escalate_priv
```

```
screenshot
```

```
win_registry
```

```
win_token
```

```
win_files_operation
```

```
CRC32 poly Constant
```

packed/encrypted sections

```
Codoso_3
```

```
IsPE32
```

```
IsWindowsGUI
```

```
IsPacked
```

```
HasOverlay
```

```
HasDebugData
```

```
HasRichSignature
```


Suspicious Strings

```
$ strings unknown_file
```

```
XPlugKeyLogger.cpp
```

```
XPlugProcess.cpp
```

```
XPlugRegedit.cpp
```

```
XPlugScreen.cpp
```

```
XPlugShell.cpp
```

```
XPlugSQL.cpp
```

```
XPlugTelnet.cpp
```

```
/update?id=%8.8x
```

```
POST
```

```
HttpOpenRequestA
```

```
HttpSendRequestExA
```

```
$ strings unknown_file
```

```
XPlugKeyLogger.cpp
```

```
XPlugProcess.cpp
```

```
XPlugRegedit.cpp
```

```
XPlugScreen.cpp
```

```
XPlugShell.cpp
```

```
XPlugSQL.cpp
```

```
XPlugTelnet.cpp
```

```
/update?id=%8.8x
```

```
POST
```

```
HttpOpenRequestA
```

```
HttpSendRequestExA
```

```
$ strings unknown_file
```

```
XPlugKeyLogger.cpp
```

```
XPlugProcess.cpp
```

```
XPlugRegedit.cpp
```

```
XPlugScreen.cpp
```

malicious functionalities

```
XPlugTelnet.cpp
```

```
/update?id=%8.8x
```

```
POST
```

```
HttpOpenRequestA
```

```
HttpSendRequestExA
```

```
$ strings unknown_file
```

```
XPlugKeyLogger.cpp
```

```
XPlugProcess.cpp
```

```
XPlugRegedit.cpp
```

```
XPlugScreen.cpp
```

```
XPlugShell.cpp
```

```
XPlugSQL.cpp
```

```
XPlugTelnet.cpp
```

```
/update?id=%8.8x
```

```
POST
```

```
HttpOpenRequestA
```

```
HttpSendRequestExA
```

```
$ strings unknown_file
```

```
XPlugKeyLogger.cpp
```

```
XPlugProcess.cpp
```

```
XPlugRegedit.cpp
```

```
XPlugScreen.cpp
```

network communication

```
XPlugTelnet.cpp
```

```
/update?id=%8.8x
```

```
POST
```

```
HttpOpenRequestA
```

```
HttpSendRequestExA
```

Suspicious API Functions

```
$ objdump -p unknown_binary
```

```
DLL Name: USER32.dll
```

```
Hint/Ord Name
```

```
317 GetKeyState
```

```
263 GetAsyncKeyState
```

```
DLL Name: KERNEL32.dll
```

```
Hint/Ord Name
```

```
313 FindFirstFileW
```

```
325 FindNextFileW
```

```
1317 WriteFile
```

```
1326 WriteProcessMemory
```



```
$ objdump -p unknown_binary
```

```
DLL Name: USER32.dll
```

```
Hint/Ord Name
```

```
317 GetKeyState
```

```
263 GetAsyncKeyState
```

imported system APIs

```
Hint/Ord Name
```

```
313 FindFirstFileW
```

```
325 FindNextFileW
```

```
1317 WriteFile
```

```
1326 WriteProcessMemory
```

```
$ objdump -p unknown_binary
```

```
DLL Name: USER32.dll
```

```
Hint/Ord Name
```

```
317 GetKeyState
```

```
263 GetAsyncKeyState
```

```
DLL Name: KERNEL32.dll
```

```
Hint/Ord Name
```

```
313 FindFirstFileW
```

```
325 FindNextFileW
```

```
1317 WriteFile
```

```
1326 WriteProcessMemory
```

```
$ objdump -p unknown_binary
```

```
DLL Name: USER32.dll
```

```
Hint/Ord Name
```

```
317 GetKeyState
```

```
263 GetAsyncKeyState
```

keylogging functionality

```
Hint/Ord Name
```

```
313 FindFirstFileW
```

```
325 FindNextFileW
```

```
1317 WriteFile
```

```
1326 WriteProcessMemory
```

```
$ objdump -p unknown_binary
```

```
DLL Name: USER32.dll
```

```
Hint/Ord Name
```

```
317 GetKeyState
```

```
263 GetAsyncKeyState
```

```
DLL Name: KERNEL32.dll
```

```
Hint/Ord Name
```

```
313 FindFirstFileW
```

```
325 FindNextFileW
```

```
1317 WriteFile
```

```
1326 WriteProcessMemory
```

```
$ objdump -p unknown_binary
```

```
DLL Name: USER32.dll
```

```
Hint/Ord Name
```

```
317 GetKeyState
```

```
263 GetAsyncKeyState
```

file traversal & modification

```
Hint/Ord Name
```

```
313 FindFirstFileW
```

```
325 FindNextFileW
```

```
1317 WriteFile
```

```
1326 WriteProcessMemory
```

```
$ objdump -p unknown_binary
```

```
DLL Name: USER32.dll
```

```
Hint/Ord Name
```

```
317 GetKeyState
```

```
263 GetAsyncKeyState
```

```
DLL Name: KERNEL32.dll
```

```
Hint/Ord Name
```

```
313 FindFirstFileW
```

```
325 FindNextFileW
```

```
1317 WriteFile
```

```
1326 WriteProcessMemory
```

```
$ objdump -p unknown_binary
```

```
DLL Name: USER32.dll
```

```
Hint/Ord Name
```

```
317 GetKeyState
```

```
263 GetAsyncKeyState
```

process injection

```
Hint/Ord Name
```

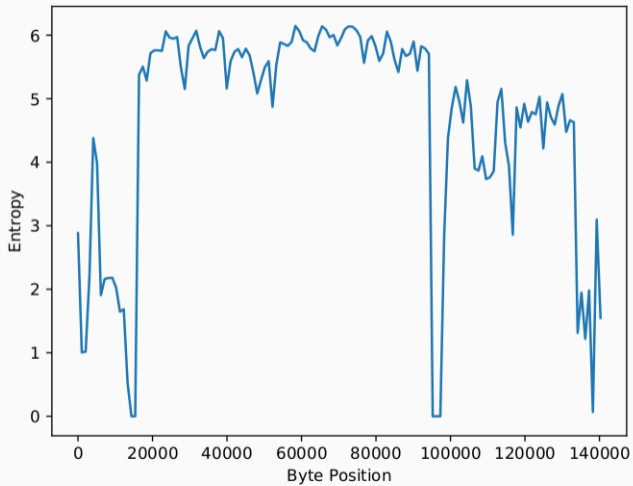
```
313 FindFirstFileW
```

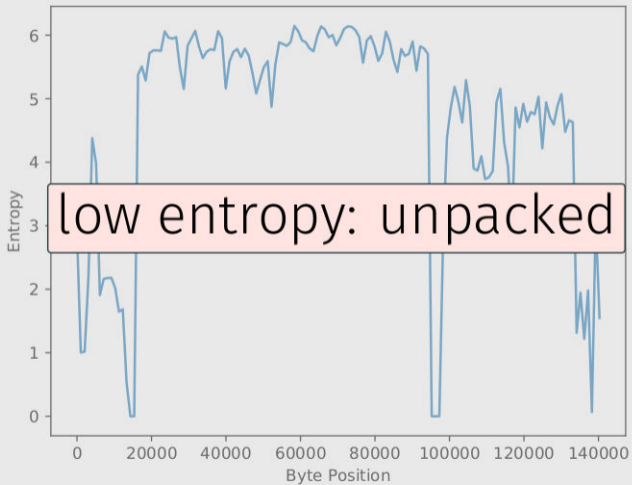
```
325 FindNextFileW
```

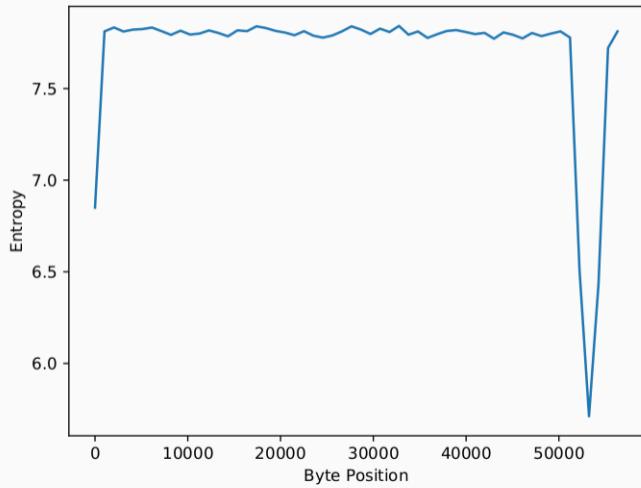
```
1317 WriteFile
```

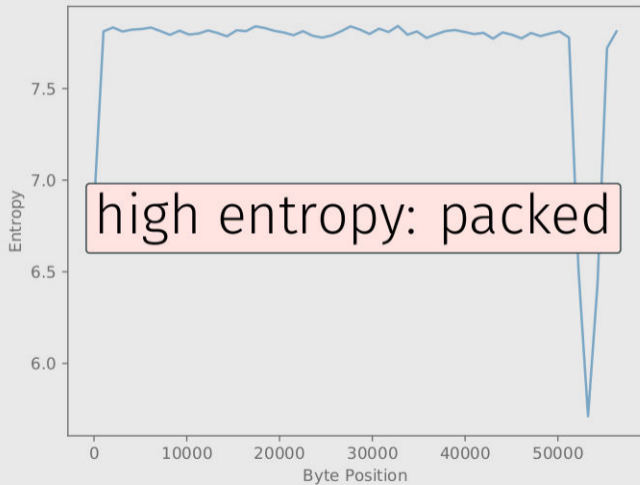
```
1326 WriteProcessMemory
```

Entropy Analysis











Recap

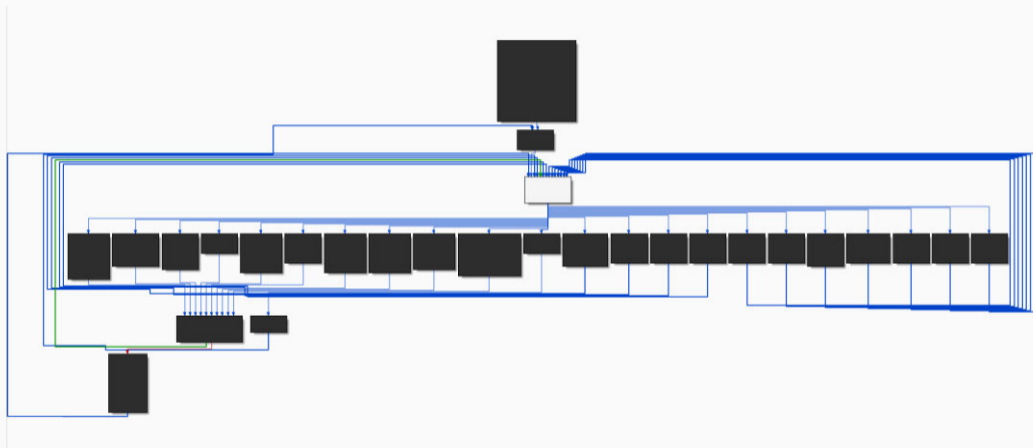
Signatures—Strings—API—Entropy



Need to Dive Deeper

→ Manual Reverse Engineering
(Guided by Heuristics)

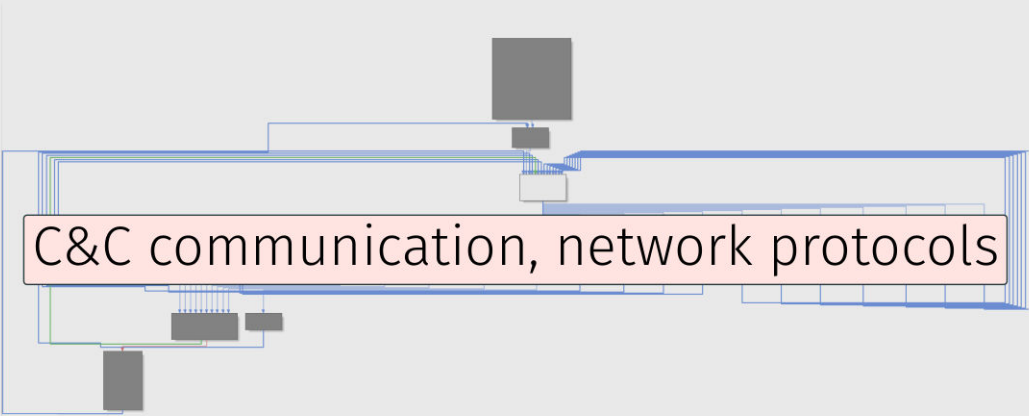
Identification of State Machines





The diagram shows a complex system architecture. A central horizontal bar is highlighted in light blue. Above this bar, a large grey rectangular block is connected to a smaller grey block, which in turn connects to a white rectangular block. Below the bar, several grey blocks are arranged, with multiple blue lines connecting them to the central bar. On the far left, a vertical stack of blue lines connects to the bar. On the far right, another vertical stack of blue lines connects to the bar. A central text box with a pink background and black border is overlaid on the bar.

loop-based dispatching routines

A schematic diagram of a network architecture. It features a central horizontal bus with multiple parallel lines. Above the bus, a large dark grey rectangular block is connected to a smaller dark grey block, which in turn connects to a light grey rectangular block. Below the bus, there are several dark grey rectangular blocks of varying sizes, connected to the bus lines. The entire diagram is enclosed in a blue border. A large, light pink rectangular box with a black border is superimposed over the center of the diagram, containing the text "C&C communication, network protocols".

C&C communication, network protocols

Frequently Called Functions

call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef
call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef call 0xdeadbeef

Functions which are often called by other functions

XOR DDoS (Statically-linked)

free	293
memcpy	191
strlen	184
memset	174
__libc_malloc	151
__lll_unlock_wake_private	148
__lll_lock_wait_private	122
ptmalloc_init	114
__strtol_internal	99
strcmp	93

free	293
memcpy	191
strlen	184
memset	174
libc malloc	151
__gcc_lock_wait_private	122
ptmalloc_init	114
__strtol_internal	99
strcmp	93

frequently called API functions

<code>crc32</code>	1253
<code>LoadLibraryA</code>	1253
<code>__seterrormode</code>	320

```
crc32          1253  
LoadLibraryA  1253
```

hash-based import hiding

XOR Decryption Loops

```
void xor_decrypt(char *input, int length) {  
    for (int i = 0; i < length; i++) {  
        input[i] = input[i] ^ 0xde;  
    }  
}
```

```
void xor_decrypt(char *input, int length) {  
    for (int i = 0; i < length; i++) {
```

XOR operation with a constant in a loop

```
}
```

```
void xor_decrypt(char *input, int length) {  
    for (int i = 0; i < length; i++) {
```

string decryption and code unpacking

```
}
```

RC4 Crypto Algorithm

Most-widely used cryptographic
algorithm in malware

How to identify?

```
// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length);

// RC4 Encryption/Decryption Function
void rc4(uint8_t *key, int key_length, uint8_t *input, uint8_t *output,
         int data_length);

// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length) {
    for(int i = 0; i < 256; ++i) {
        ...
    }

    int j = 0;
    for(int i = 0; i < 256; ++i) {
        ...
    }
}
```

```
// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length);

// RC4 Encryption/Decryption Function
void rc4(uint8_t *key, int key_length, uint8_t *input, uint8_t *output,
         int data_length);

// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length) {
    for(int
        ...
    }

    int j = 0;
    for(int i = 0; i < 256; ++i) {
        ...
    }
}
```

two algorithms: KSA and PRGA

```
// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length);

// RC4 Encryption/Decryption Function
void rc4(uint8_t *key, int key_length, uint8_t *input, uint8_t *output,
         int data_length);

// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length) {
    for(int i = 0; i < 256; ++i) {
        ...
    }

    int j = 0;
    for(int i = 0; i < 256; ++i) {
        ...
    }
}
```

```
// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length);

// RC4 Encryption/Decryption Function
void rc4(uint8_t *key, int key_length, uint8_t *input, uint8_t *output,
         int data_length);

// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length) {
```

KSA: Function with two loops and constant 256

```
    }

    int j = 0;
    for(int i = 0; i < 256; ++i) {
        ...
    }
}
```

```
// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length);

// RC4 Encryption/Decryption Function
void rc4(uint8_t *key, int key_length, uint8_t *input, uint8_t *output,
         int data_length);

// Key Scheduling Algorithm (KSA)
void ksa(uint8_t *key, uint8_t *S, int key_length) {
    string decryption, domain generation
}

    int j = 0;
    for(int i = 0; i < 256; ++i) {
        ...
    }
}
```

→ Recap

State Machines—API—XOR—RC4

Conclusion

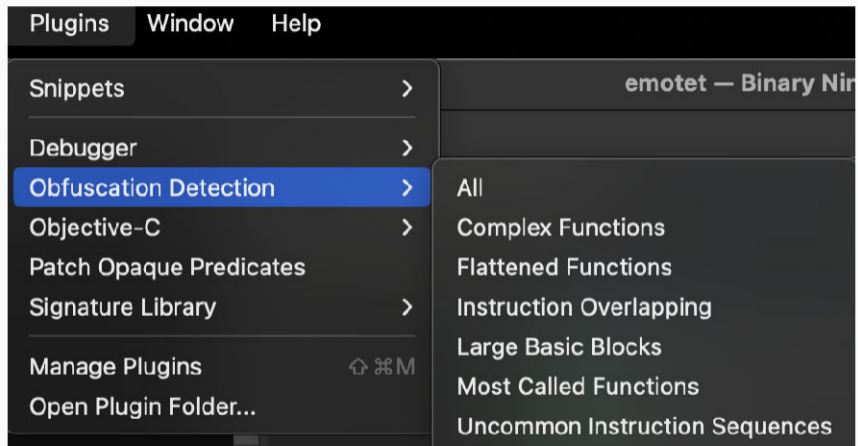
Takeaways

1. binaries from unknown sources cannot be trusted
2. common analysis techniques provide first insights
3. guided manual analysis can confirm initial indicators

Takeaways

1. binaries from unknown sources cannot be trusted
2. common analysis techniques provide first insights
3. guided manual analysis can confirm initial indicators

Supported analysis is crucial to detect potential malicious behavior in unknown binaries.



https://github.com/mrphrazer/obfuscation_detection

```
Log Python Console
Search log
[Default] =====
[Default] Control Flow Flattening
[Default] Function 0x4063f0 (sub_4063f0) has a flattening score of 0.9929577464788732.
[Default] Function 0x4012a0 (sub_4012a0) has a flattening score of 0.9855072463768116.
[Default] Function 0x402b60 (sub_402b60) has a flattening score of 0.9855072463768116.
[Default] Function 0x409e20 (sub_409e20) has a flattening score of 0.9846153846153847.
[Default] Function 0x40a4b0 (sub_40a4b0) has a flattening score of 0.9821428571428571.
[Default] Function 0x404f50 (sub_404f50) has a flattening score of 0.9818181818181818.
[Default] Function 0x402210 (sub_402210) has a flattening score of 0.9807692307692307.
[Default] Function 0x4025a0 (sub_4025a0) has a flattening score of 0.9787234042553191.
[Default] Function 0x40a9d0 (sub_40a9d0) has a flattening score of 0.9772727272727273.
[Default] Function 0x409530 (sub_409530) has a flattening score of 0.9761904761904762.
[Default] Function 0x407060 (sub_407060) has a flattening score of 0.975609756097561.
[Default] Function 0x401fa0 (sub_401fa0) has a flattening score of 0.975609756097561.
[Default] Function 0x406080 (sub_406080) has a flattening score of 0.975.
[Default] Function 0x4038b0 (sub_4038b0) has a flattening score of 0.975.
[Default] Function 0x401940 (sub_401940) has a flattening score of 0.9736842105263158.
[Default] Function 0x408660 (sub_408660) has a flattening score of 0.972972972972973.
[Default] Function 0x408f30 (sub_408f30) has a flattening score of 0.972972972972973.
[Default] Function 0x409860 (sub_409860) has a flattening score of 0.9714285714285714.
```

https://github.com/mrphrazer/obfuscation_detection

Obfuscation Detection 1.7

Tim Blazytko | community | GPL-2.0 | ☆ 351 | Last Update: 2023-03-14

Category: helper

Automatically detect obfuscated code and other interesting code constructs

Description License

Obfuscation Detection (v1.7)

Author: **Tim Blazytko**

[Automatically detect obfuscated code and other interesting code constructs](#)

Description:

Obfuscation Detection is a Binary Ninja plugin to detect obfuscated code and interesting code constructs (e.g., state machines) in binaries. Given a binary, the plugin eases analysis by identifying code locations which might be worth a closer look during reverse engineering.

Based on various heuristics, the plugin pinpoints functions that contain complex or uncommon code constructs. Such code constructs may implement

- obfuscated code
- state machines and protocols
- C&C server communication
- string decryption routines
- cryptographic algorithms

The following blog posts provide more information about the underlying heuristics and demonstrate their use cases:

- [Automated Detection of Control-flow Flattening](#)
- [Automated Detection of Obfuscated Code](#)
- [Statistical Analysis to Detect Uncommon Code](#)

Some example use cases can be found in [examples](#).

Core Features

Install

Summary

- common approaches to identify malicious behavior in unknown binaries
- manual reverse engineering to dive deeper
- heuristics to guide manual approaches

https://github.com/mrphrazer/obfuscation_detection/

Tim Blazytko

 @mr_phrazer

 synthesis.to

 tim@blazytko.to

References

- “Automated Detection of Obfuscated Code” by Tim Blazytko
https://synthesis.to/2021/08/10/obfuscation_detection.html
- “Automated Detection of Control-flow Flattening” by Tim Blazytko
https://synthesis.to/2021/03/03/flattening_detection.html
- “Statistical Analysis to Detect Uncommon Code” by Tim Blazytko
https://synthesis.to/2023/01/26/uncommon_instruction_sequences.html
- “Identification of API Functions in Binaries” by Tim Blazytko
https://synthesis.to/2023/08/02/api_functions.html