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

Setting the Scene

- 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
Codoso_3
IsPE32
IsWindowsGUI
IsPacked
HasOverlay
HasDebugData
HasRichSignature
```

pre-defined signatures

```
$ 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
Codoso_3
IsPE32
IsWindowsGUI
IsPacked
HasOverlay
HasDebugData
HasRichSignature
```

potential screen capture

```
$ 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
Codoso_3
IsPE32
IsWindowsGUI
IsPacked
HasOverlay
HasDebugData
HasRichSignature
```

cryptographic algorithms

```
$ 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
Codoso_3
IsPE32
IsWindowsGUI
IsPacked
HasOverlay
HasDebugData
HasRichSignature
```

packed/encrypted sections

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
XPlugTelnet.cpp
/update?id=%8.8x
POST
HttpOpenRequestA
HttpSendRequestExA
```

malicious functionalities

```
$ 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
XPlugTelnet.cpp
/update?id=%8.8x
POST
HttpOpenRequestA
HttpSendRequestExA
```

network communication

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

    Hint/Ord Name
        313 FindFirstFileW
        325 FindNextFileW
       1317 WriteFile
       1326 WriteProcessMemory
```

imported system APIs

```
$ 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

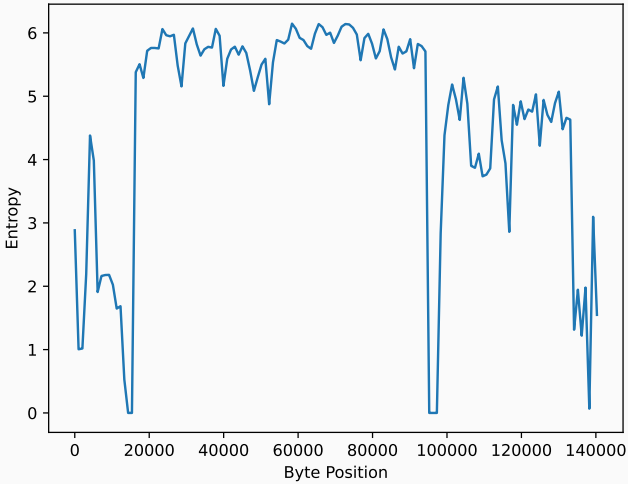
```
    Hint/Ord Name
        313 FindFirstFileW
        325 FindNextFileW
       1317 WriteFile
       1326 WriteProcessMemory
```

```
$ 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
```

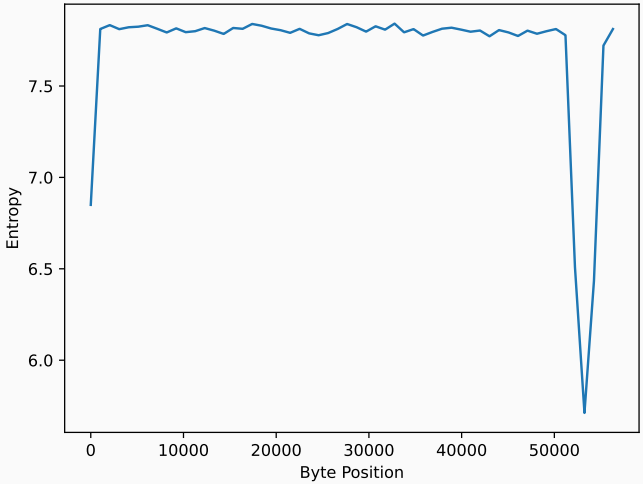
```
$ objdump -p unknown_binary

    DLL Name: USER32.dll
    Hint/Ord Name
        317 GetKeyState
        263 GetAsyncKeyState

    Hint/Ord Name
        313 FindFirstFileW
        325 FindNextFileW
       1317 WriteFile
       1326 WriteProcessMemory
```

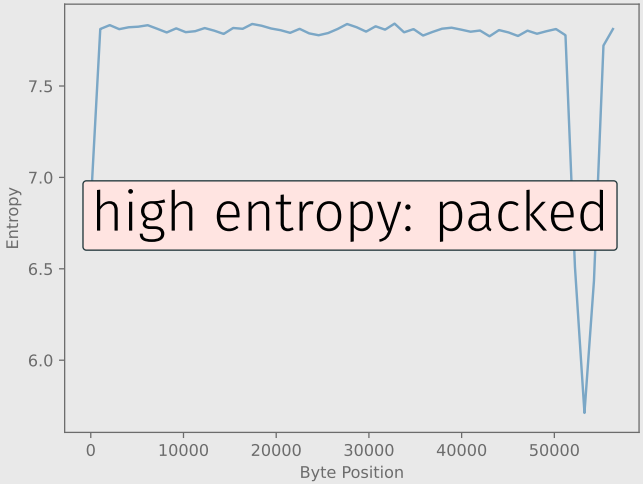
process injection

Entropy Analysis


Entropy
0
1
2
3
4
5
6
0
20000
40000
60000
80000
100000
120000
140000
Byte Position


low entropy: unpacked
Entropy
Byte Position
0
1
2
3
4
5
6
0
20000
40000
60000
80000
100000
120000
140000


Entropy
7.5
7.0
6.5
6.0
0
10000
20000
30000
40000
50000
Byte Position


7.5
7.0
6.5
6.0
Entropy
high entropy: packed
0
10000
20000
30000
40000
50000
Byte Position

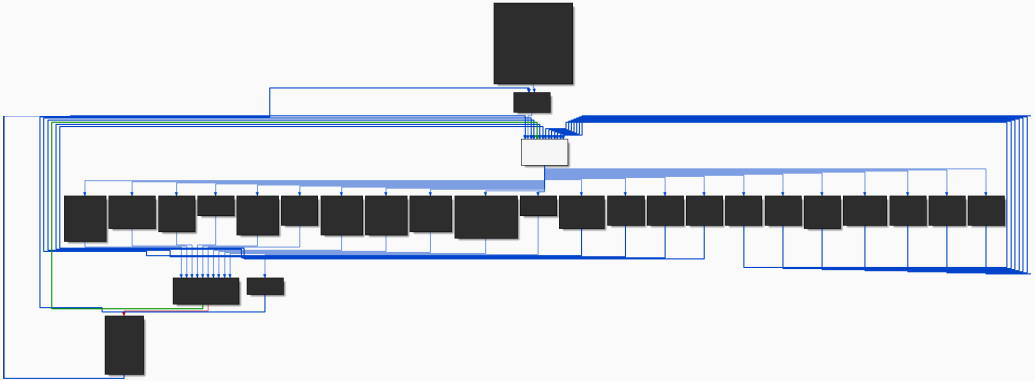
Recap

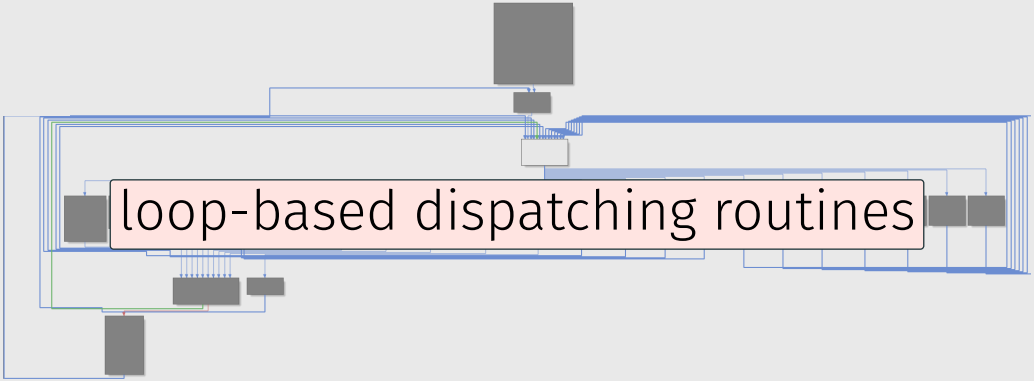
Signatures—Strings—API—Entropy

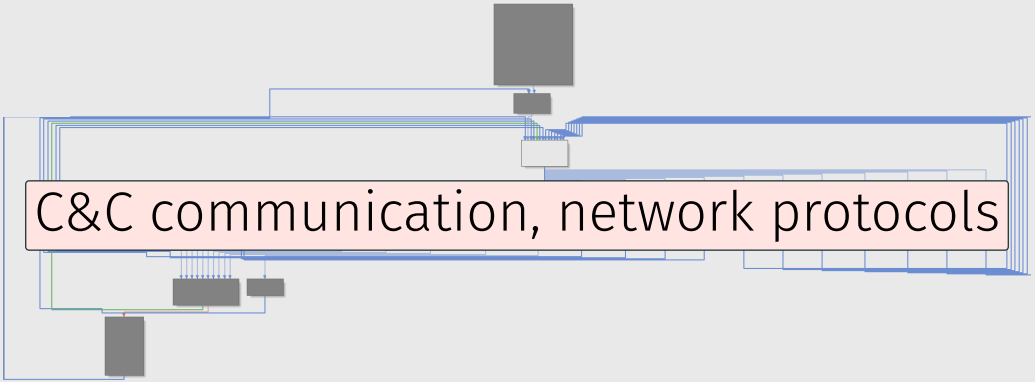
Need to Dive Deeper

→ Manual Reverse Engineering
(Guided by Heuristics)

Identification of State Machines

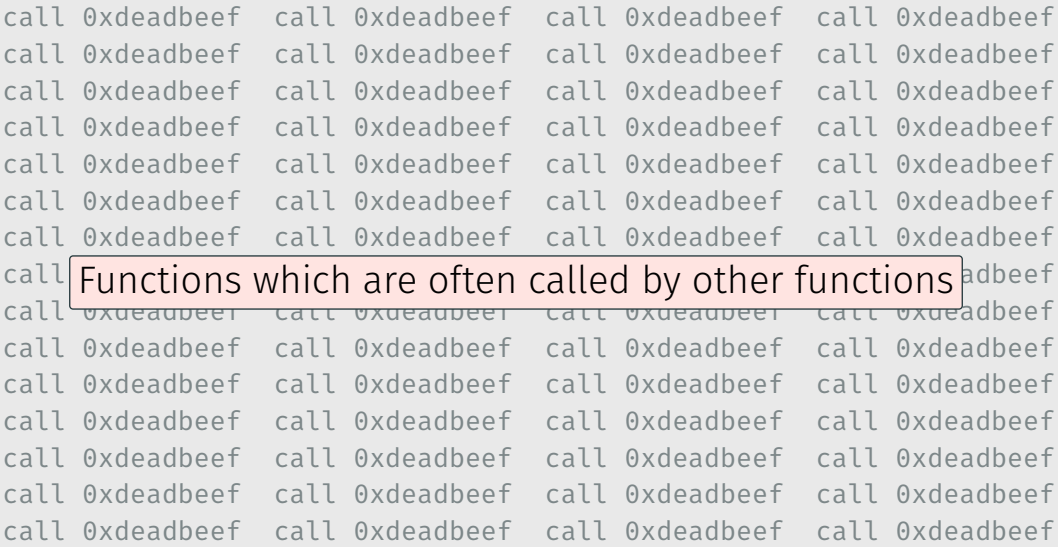



loop-based dispatching routines


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
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

XOR DDoS (Statically-linked)

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

frequently called API functions

```
crc32            1253
LoadLibraryA     1253
__seterrormode    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++) {
        input[i] = input[i] ^ 0x42;
    }
}
```

XOR operation with a constant in a loop

```
void xor_decrypt(char *input, int length) {
    for (int i = 0; i < length; i++) {
        input[i] = input[i] ^ 0x42;
    }
}
```

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) {
    }

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

KSA: Function with two loops and constant 256

```
// 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) {
   }

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

string decryption, domain generation

→ Recap

State Machines—API—XOR—RC4

Conclusion

Takeaways

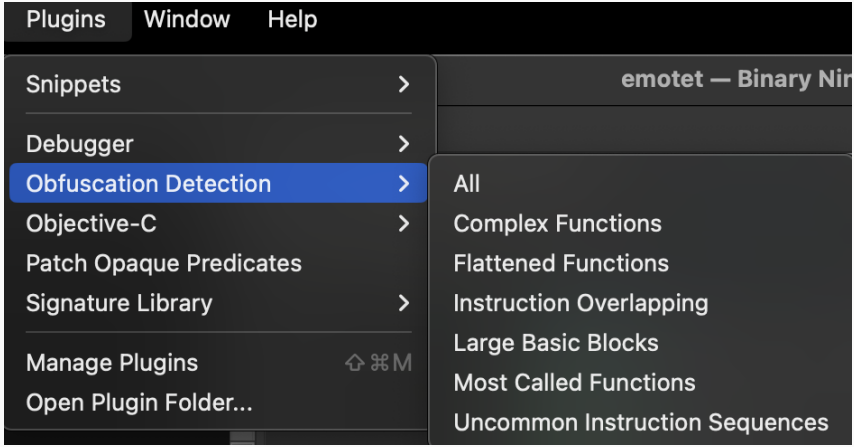
1. binaries from unknown sources cannot be trusted
2. common analysis techniques provide first insights
3. guided manual analysis can confirm initial indicators

Takeaways

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

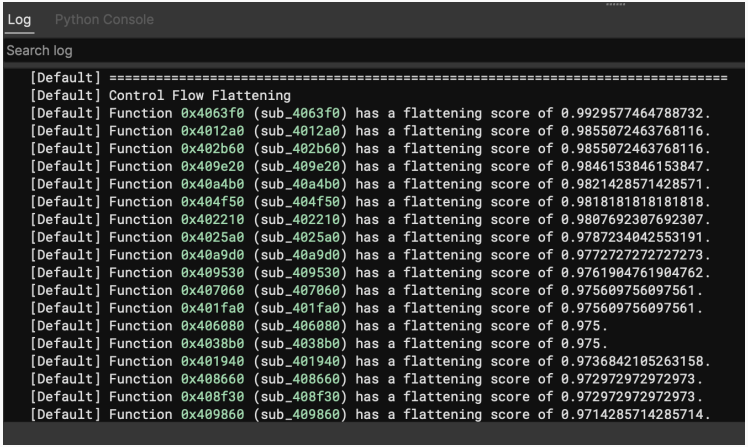
Supported analysis is crucial to detect potential malicious behavior in unknown binaries.

Binary Ninja Plugin



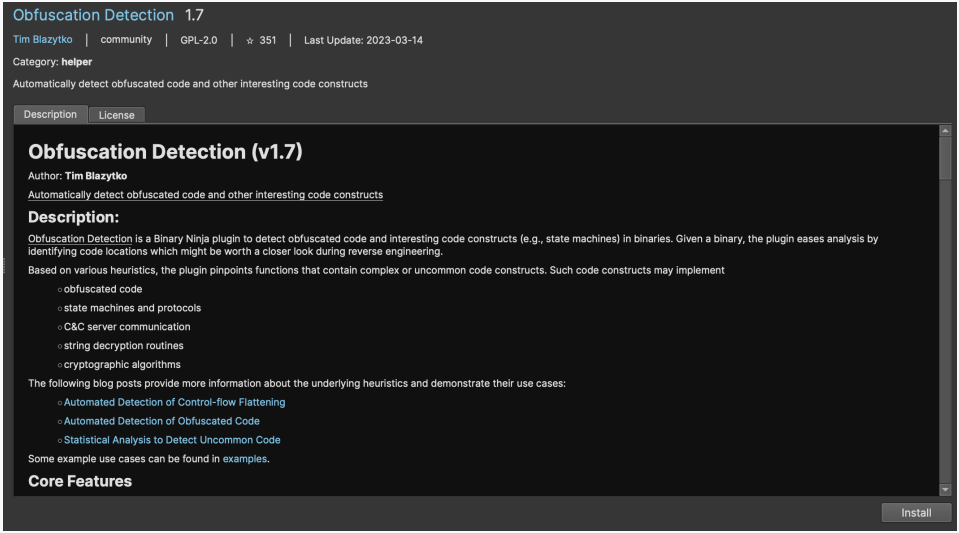
https://github.com/mrphrazer/obfuscation_detection

Binary Ninja Plugin



https://github.com/mrphrazer/obfuscation_detection

Plugin Manager



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`