

# Malware Analysis at Scale

## ~ Defeating EMOTET by Ghidra ~

株式会社サイバーディフェンス研究所 中島将太  
トレンドマイクロ株式会社 原弘明

ALLSAFE

## > whoami



Shota Nakajima

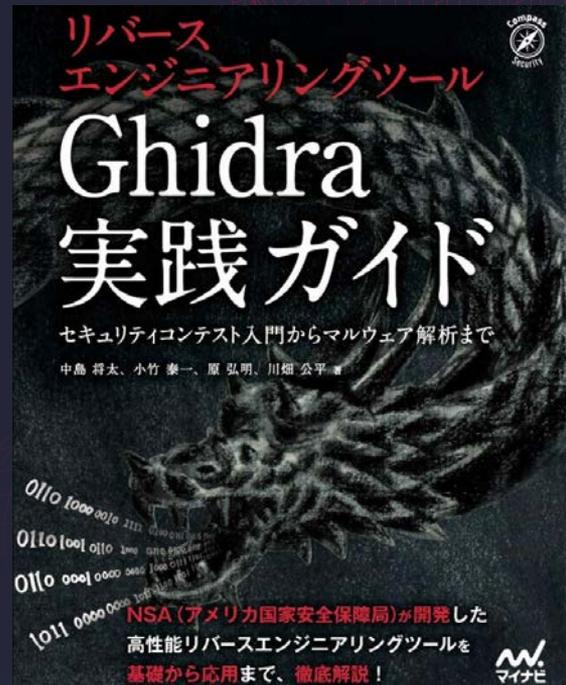
- ▶ 株式会社サイバーディフェンス研究所でマルウェア解析、インシデントレスポンス業務、脅威情報の収集・分析業務に従事。
- ▶ JSAC、HITCON CMT、AVAR、CPRCon、Black Hat EUROPE Arsenal、CodeBlue BlueBoxなどで発表経験あり。
- ▶ 技術系同人サークルAllsafeのプロデューサー。



Hiroaki Hara

- ▶ トレンドマイクロ株式会社にて、マルウェア解析やインシデントレスポンス、スレトリサーチなどに従事。
- ▶ 技術系同人サークルAllsafeの澤村・スペンサー・英梨々（アートディレクター）。

# ALLSAFE



# Malware Analysis at Scale

---

## テーマ: いかにして楽に解析を終わらせるか

- 解析者の現状として、次のような課題がある
  - ✓ 静的解析は時間がかかり、根本的につらい
  - ✓ 膨大な数のマルウェアを解析する必要がある
  - ✓ 手動での解析は現実的に不可能



解析の自動化 (=スケールする解析) をめざす

# Target

---

- マルウェア解析経験者
  - C言語、アセンブリ言語を理解している
  - Windowsのプログラミングについて知識がある
  - IDA, Ghidraを使ってマルウェアの静的解析をしたことがある
- Pythonでのプログラミング経験を有する者
  - Pythonの基本的な文法がわかる

# TABLE OF CONTENTS

01

## Basic of Ghidra

- Ghidra Usage
- Ghidra Script

02

## Automated Analysis of EMOTET

- Surface Analysis
- API Hashing and Dynamic Call
- Decrypt strings
- Config Structure Analysis



# 01 Basic of Ghidra

---

# What is Ghidra?

---

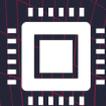
## ソフトウェアリバースエンジニアリングツールセット

- 2019年にNSAによってOpen Source Softwareとして公開
  - <https://github.com/NationalSecurityAgency/ghidra>
- フリーでありながら、豊富な機能を提供



# Ghidra's Basic Features

---



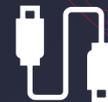
## Disassembler/Decompiler

多様なアーキテクチャに対応した  
ディスアセンブラに加え、  
擬似C言語へ変換するデコンパイラ機能



## Ghidra Script

Ghidraの各種機能をスクリプト  
(Java/Python)から利用可能



## Ghidra Extension

自作プラグインによる  
機能拡張をサポート



## Headless Analyzer

CLI経由での操作



## Ghidra Server

複数ユーザによる共同作業を  
サポート



## Debugger

GDB/WinDBGをバックエンド  
としたデバッグ機能を提供

# Ghidra Usage

---

# Code Browser

## Listing View

- ディスアセンブル・データを表示

## Decompile View

- Listing Viewのデコンパイル結果を表示

## Program Tree

- セクション名などを表示

## Source Tree

- インポート/エクスポート関数、関数名、クラス名などを列挙

## Data Type Manager

- 独自構造体などのデータ型を管理
- ヘッダファイルなどの読み込み

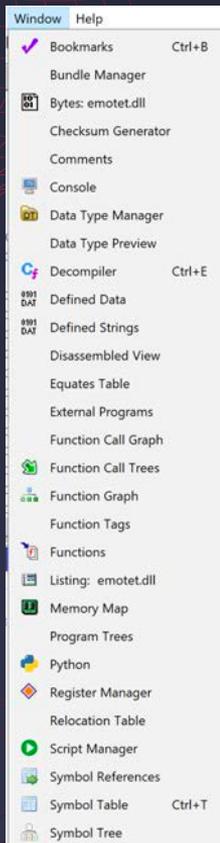
The screenshot shows the CodeBrowser interface with several panels highlighted by colored arrows:

- Program Tree:** Located on the left, showing a tree view of the loaded file 'emotet.dll' with sections like 'text', 'pdata', 'data', and 'reloc'.
- Source Tree:** Located below the Program Tree, showing a tree view of symbols including 'Imports', 'Exports', 'Control\_RunDLL', 'entry', 'RunDLL', 'Functions', 'Labels', 'Classes', and 'Namespaces'.
- Data Type Manager:** Located at the bottom left, showing a list of data types including 'BuiltInTypes', 'emotet.dll', 'winapi\_32', and 'windows\_vs12\_32'.
- Listing View:** The central panel, showing assembly code for the function 'Control\_RunDLL'. It includes addresses, disassembled instructions, and comments.
- Decompile View:** The right panel, showing the decompiled C++ code for 'Control\_RunDLL'. It includes function signatures and the function body.
- Console:** Located at the bottom, showing a 'Console - Scripting' window.

## Console

- スクリプトの標準出力結果を表示

# Other Windows



## ■ Bookmarks

- 特定のアドレスに対し、名前をつけてブックマークとして保存

## ■ Bytes

- HEXエディタ

## ■ Defined Strings

- プログラム内で定義された可読文字列

## ■ Function Graph

- ディスアセンブル結果をグラフ形式で表示

## ■ Python

- Pythonインタプリタの起動

## ■ Script Manager

- Ghidra Scriptを管理するマネージャの起動

# Name

- Ghidraではバイナリのロード時に自動解析して関数や変数に名前を付ける
  - 関数(FUN\_\*)
  - 文字列(s\_\*)
  - ローカル変数(local\_\*)
  - 変数(\*Var\*)

```

27 LAB_100037ff:
28   do {
29       if (iVar3 == 0xf4e9d7) {
30           local_94 = FUN_100010cf();
31           iVar3 = 0xacc66de;
32       }

```

	LAB_10003a00		XREF[1]:	10003805(j)
10003a00	8b 44 24 30	MOV	EAX,dword ptr [ESP + local_118]	
10003a04	8b 84 24 ...	MOV	EAX,dword ptr [ESP + local_bc]	
10003a0b	e8 bf d6 ...	CALL	FUN_100010cf	undefined FUN_100010cf(void)
10003a10	89 84 24 ...	MOV	dword ptr [ESP + local_94],EAX	
10003a17	be de 66 ...	MOV	ESI,0xacc66de	

	s_Control_RunDLL_1001e052		XREF[1]:	1001e040(*)
1001e052	43 6f 6e ...	ds	"Control_RunDLL"	
	s_RunDLL_1001e061		XREF[1]:	1001e044(*)
1001e061	52 75 6e ...	ds	"RunDLL"	

# Change Name

- リネームして解析結果を反映することで、Ghidraを読みやすくしていく
- 右クリックメニューから選択

LAB\_10003a00

```

10003a00 8b 44 24 30  MOV     EAX,dword ptr [ESP +
10003a04 8b 84 24 ...  MOV     EAX,dword ptr [ESP +
10003a0b e8 bf d6 ...  CALL    FUN_100010cf
10003a10 89 84 24 ...  MOV     dword ptr [ESP + loc
10003a17 be de 66 ...  MOV     ESI,0xacc66de
  
```

LAB\_10003a1c

Context Menu:

- Create Function
- Create Thunk Function
- Function
- Function Variables
- Edit Label... L

```

27 LAB_100037ff:
28 do {
29     if (iVar3)
30         local
31         iVar3
32     }
  
```

Context Menu:

- Edit Function Signature
- Rename Variable L

C:\Decompile: FUN\_1000323c - (emotet.dll)

```

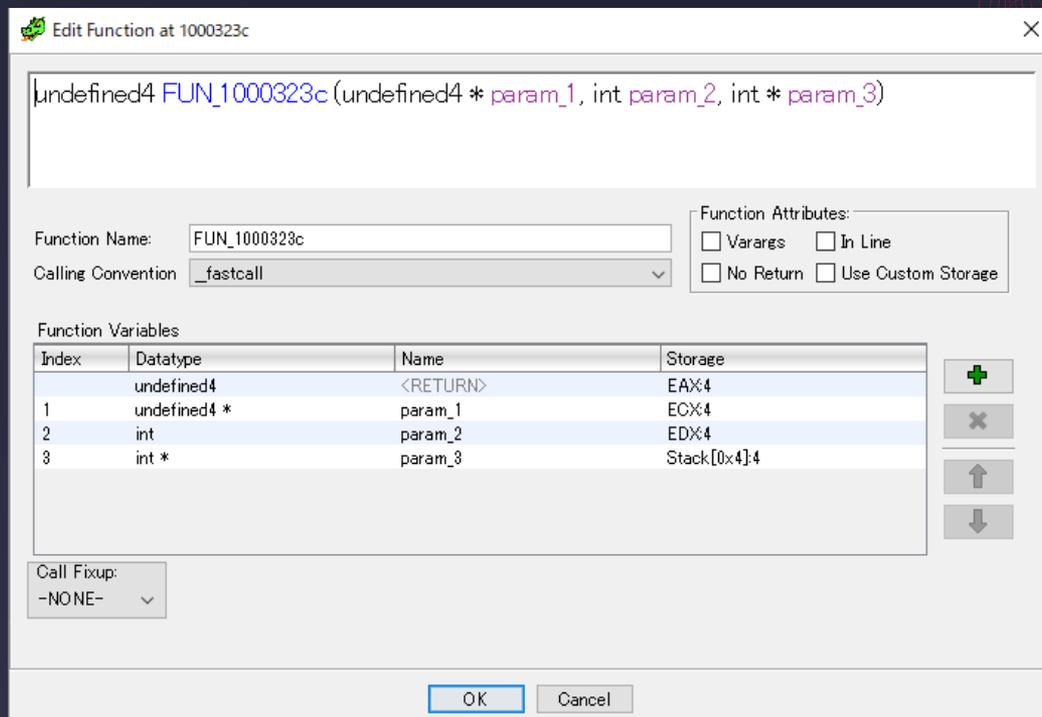
1
2 undefined4 __fastcall FUN_1000323c(undefined4 param_1,int param_2,int *param_3)
3
4 {
5     uint uVar1;
  
```

Context Menu:

- Edit Function Signature
- Rename Function L

# Change Function Signature

- 関数の定義を変更する



# XREF(Cross References)

## ■ データの参照元の表示

```

undefined __stdcall FUN_1000d26b(undefined4 param_1, und...
undefined      AL:1      <RETURN>
undefined4     Stack[0x4]:4 param_1      XREF[1]: 1000d27d (R)
undefined4     Stack[0x8]:4 param_2      XREF[1]: 1000d27a (R)
undefined4     Stack[0xc]:4 param_3      XREF[1]: 1000d277 (R)
undefined4     Stack[0x10]:4 param_4     XREF[1]: 1000d274 (R)
undefined4     Stack[0x14]:4 param_5     XREF[2]: 1000d271 (R),
                                                1000d28a (R)
undefined4     Stack[0x18]:4 param_6     XREF[2]: 1000d26e (R),
                                                1000d28d (R)
FUN_1000d26b                                         XREF[1]: FUN_1000323c:10003a74 (c)

```

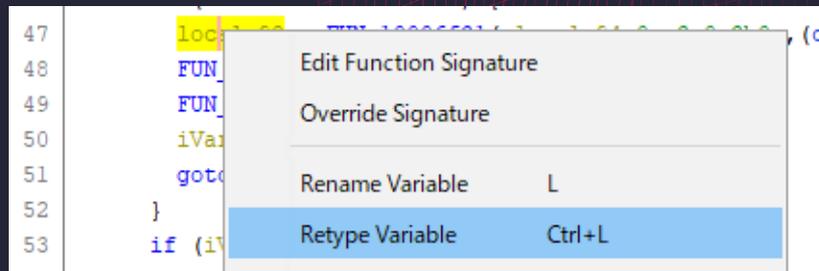
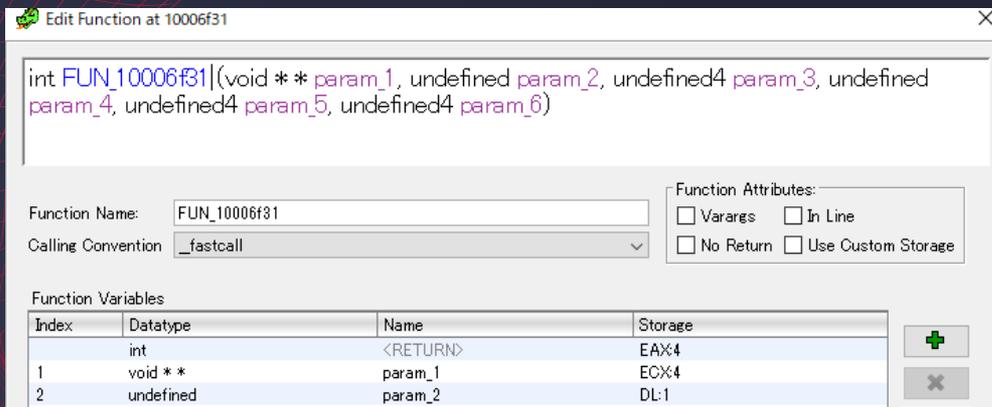
References to FUN\_1000d126 - 4 locations [CodeBrowser: ghidra\_project:/emotet.dll]

Help

Location	Label	Code Unit	Context	Function Name
10003916		CALL FUN_1000d126	UNCONDITIONAL_CALL	FUN_1000323c
10003a49		CALL FUN_1000d126	UNCONDITIONAL_CALL	FUN_1000323c
1000c63f		CALL FUN_1000d126	UNCONDITIONAL_CALL	FUN_1000c351
1000e7f8		CALL FUN_1000d126	UNCONDITIONAL_CALL	FUN_1000e6f1

# Change Data Type

- 関数や変数の型の変更
  - 右クリックメニューから選択



# Ghidra Script

---

# Ghidra Script

---

## Ghidraの各種操作を自動化・効率化する機能

- こんなことができる
  - ✓ バイト列の検索やコメントの付与など、手動作業の自動化
  - ✓ 大量のファイルのインポート・解析処理など、反復作業の自動化
  - ✓ 暗号化されたデータや文字列の復号など、解析の自動化
  - ✓ エミュレータによる実行

# Ghidra API

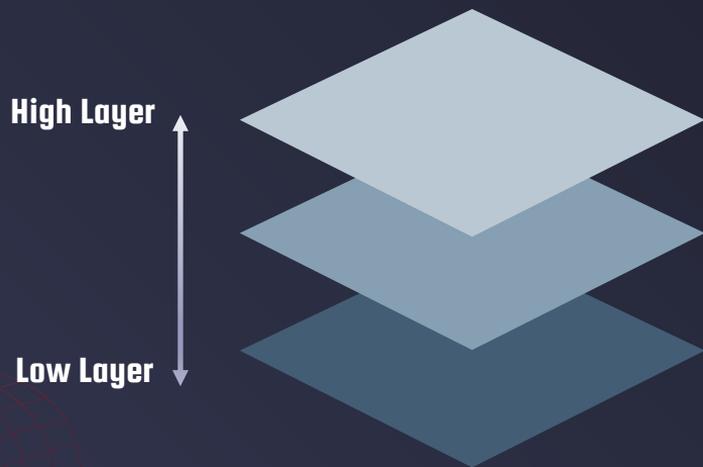
- スクリプトからGhidraの機能にアクセスするためのAPI
- JavaとPython (Jython2.7) で実装が可能

## Python実装の例

```
Ghidra > ghidra_9.1.2_PUBLIC > Ghidra > Features > Python > ghidra_scripts > ghidra_basics.py > ...
1  # Examples of basic Ghidra scripting in Python
2  # @category: Examples.Python
3
4  # Get info about the current program
5  print
6  print "Program Info:"
7  program_name = currentProgram.getName()
8  creation_date = currentProgram.getCreationDate()
9  language_id = currentProgram.getLanguageID()
10 compiler_spec_id = currentProgram.getCompilerSpec().getCompilerSpecID()
11 print "%s: %s_%s (%s)\n" % (program_name, language_id, compiler_spec_id, creation_date)
12
```

# Type of Ghidra API

- Ghidra APIは大きく3つに分類でき、次のような階層構造になっている
  - ほとんどの場合で FlatProgram API/Script API さえ理解していれば事足りる
  - なにか必要な機能があったらまずはFlatProgramAPIのドキュメントを検索する



- **Script API**

[https://ghidra.re/ghidra\\_docs/api/ghidra/app/script/GhidraScript.html](https://ghidra.re/ghidra_docs/api/ghidra/app/script/GhidraScript.html)

- **FlatProgram API**

[https://ghidra.re/ghidra\\_docs/api/ghidra/program/flatapi/FlatProgramAPI.html](https://ghidra.re/ghidra_docs/api/ghidra/program/flatapi/FlatProgramAPI.html)

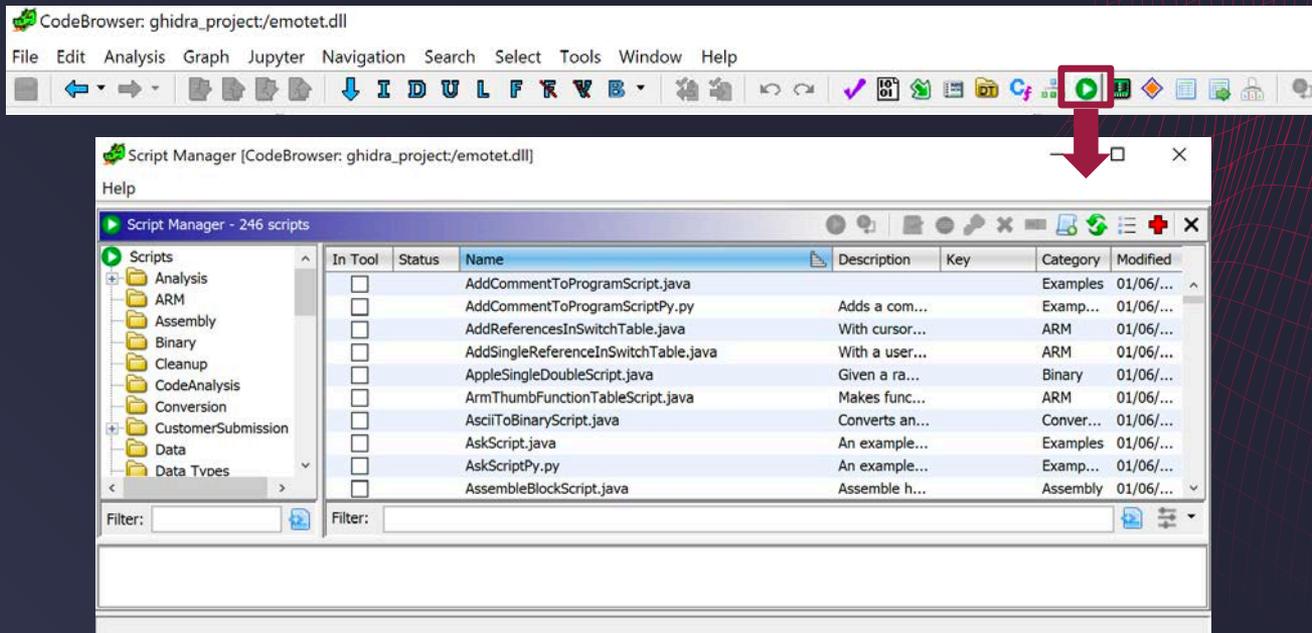
- **Program API**

[https://ghidra.re/ghidra\\_docs/api/ghidra/program/model/listing/Program.html](https://ghidra.re/ghidra_docs/api/ghidra/program/model/listing/Program.html)

# Script Manager

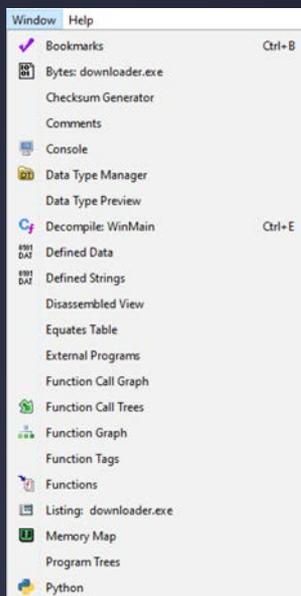
- Ghidra Scriptの実行や管理を行う機能

Script Managerの起動手順



# Built-in Python Interpreter

- ビルトインのPythonインタプリタが提供されている
  - メニューの「Window」 > 「Python」で起動



A screenshot of the Python interpreter window in Ghidra. The window title is 'Python [CodeBrowser(2): ghidra\_project/downloader.exe]'. The interpreter is running a script that finds and prints URLs from a program's memory. The output shows two URLs: 'http://allsafe.local/payload.bin' and 'http://allsafe.local/malware.bin'.

```
Python [CodeBrowser(2): ghidra_project/downloader.exe]
Help
Python - Interpreter
Python Interpreter for Ghidra
Based on Jython version 2.7.1 (default:0df7adb1b397, Jun 30 2017, 19:02:43)
[OpenJDK 64-Bit Server VM (Oracle Corporation)]
Press 'F1' for usage instructions
>>> currentProgram
downloader.exe - .ProgramDB
>>> founds = findBytes(None, 'https?://.*', -1)
>>> founds
array(ghidra.program.model.address.Address, [00418030, 00418060])
>>> for found in founds:
...     value = getDataAt(found).getValue()
...     print('{} -> {}'.format(found, value))
...
00418030 -> http://allsafe.local/payload.bin
00418060 -> http://allsafe.local/malware.bin
>>>
```

# Why Interpreter?

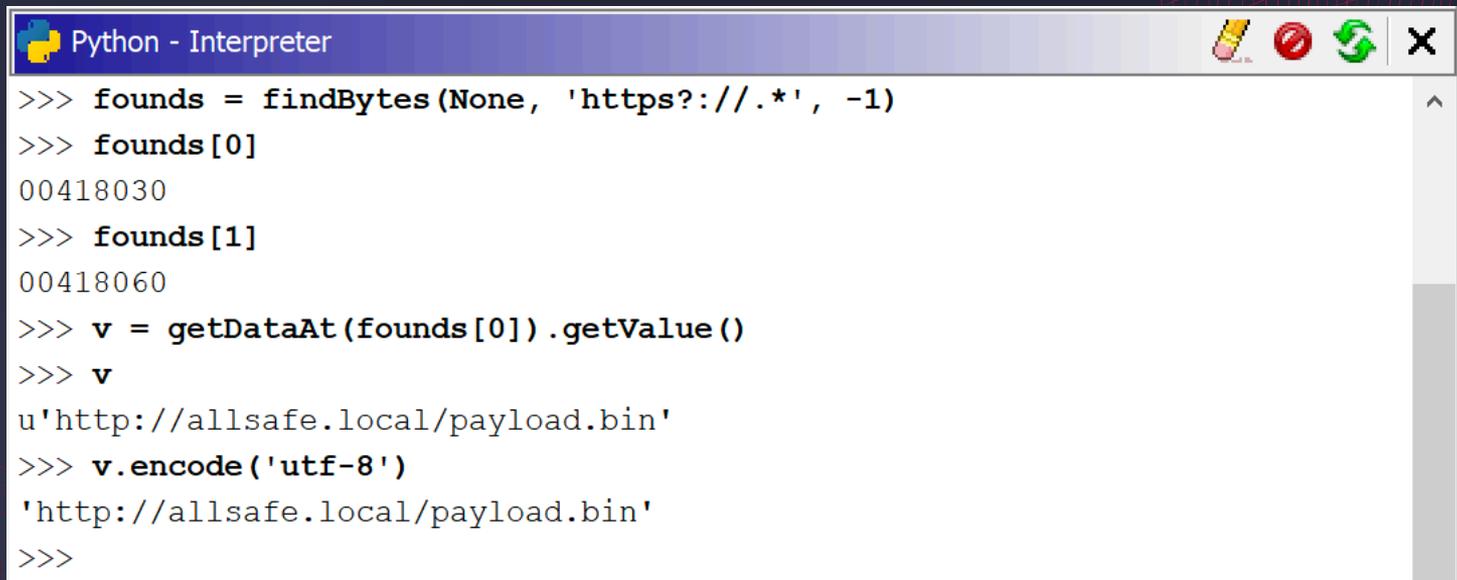
---

検証のためのトライアンドエラーに便利



# Why Interpreter? Pt.I

- 変数内部のデータを都度確認しながら処理を記述できる



```
Python - Interpreter
>>> founds = findBytes(None, 'https?://.*', -1)
>>> founds[0]
00418030
>>> founds[1]
00418060
>>> v = getDataAt(founds[0]).getValue()
>>> v
u'http://allsafe.local/payload.bin'
>>> v.encode('utf-8')
'http://allsafe.local/payload.bin'
>>>
```

# Why Interpreter? Pt.2

- コード補完してくれる
  - 入力中にTabを押すと利用可能なメソッドを表示してくれる

```
>>> get|
```

getAddressFactory (Method)	^
getAnalysisOptionDefaultValue (Method)	
getAnalysisOptionDefaultValues (Method)	
getAnalysisOptionDescription (Method)	
getAnalysisOptionDescriptions (Method)	
getBookmarks (Method)	
getByte (Method)	
getBytes (Method)	v

```
>>> currentProgram.
```

ADDED_VARIABLE_STORAGE_MANAGER_VERSION (Integer)	^
ANALYSIS_OPTIONS_MOVED_VERSION (Integer)	
ANALYSIS_PROPERTIES (Unicode)	
ANALYSIS_START_DATE (Unicode)	
ANALYSIS_START_DATE_FORMAT (Unicode)	
ANALYZED (Unicode)	
AUTO_PARAMETERS_ADDED_VERSION (Integer)	
COMPOUND_VARIABLE_STORAGE_ADDED_VERSION (Integer)	v

# Why Interpreter? Pt.3

- APIのヘルプが参照できる
  - `help` に参照したいオブジェクトを渡す

インターフェイス名がわかれば検索できる

OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP

ALL CLASSES SEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHOD    DETAIL: FIELD | CONSTR | METHOD

**Package** ghidra.program.model.listing

**Interface Listing**

**All Known Implementing Classes:**  
ListingStub

---

public interface Listing

This interface provides all the methods needed to create, delete, retrieve, modify code level constructs (CodeUnits, Macros, Fragments, and Modules).

```
Python - Interpreter
>>> listing = currentProgram.getListings()
>>> help(listing)
Searching API for ghidra.program.database.ListingDB...
#####
class ListingDB
    extends java.lang.Object
    implements ghidra.program.model.listing.Listing
Database implementation of Listing.
#####
ghidra.program.database.ProgramDB program
-----
ghidra.program.database.code.CodeManager codeMgr
-----
ghidra.program.database.module.TreeManager treeMgr
-----
ghidra.program.model.listing.FunctionManager functionMgr
-----
void <init>()
-----
void setProgram(ProgramDB program)
    Set the program.
    @param program (ghidra.program.database.ProgramDB):
-----
CodeUnit getCodeUnitAt(Address addr)
    @param addr (ghidra.program.model.address.Address):
    @return ghidra.program.model.listing.CodeUnit:
```

← Listingインターフェイスの実装クラスだとわかる

← 公開メソッドのシグネチャ

# Interpreter Tips ~ import ~

- 複数行にわたるペーストができないので、関数などがコピー&ペーストできない
- なので、使いたい関数やクラスをスクリプト内で定義しておき、**import** する
  - `import` で読み込めるファイルはScript Directories配下に存在しているファイルのみなので、今回はC:¥Ghidra¥ghidra\_scripts配下に配置する
  - Script Directoriesに登録された別フォルダに同名のファイルが存在するとエラーの原因になるので、ユニークにする

```
Python - Interpreter
>>> get_all_functions()
Traceback (most recent call last):
  File "python", line 1, in <module>
NameError: name 'get_all_functions' is not defined
>>> from snippets import *
>>> funcs = get_all_functions()
```

← get\_all\_functions関数が現在の名前空間に存在しないのでエラー

← snippets.pyn内のget\_all\_functions関数が呼び出せるようになるので便利

```
Ghidra > ghidra_scripts > snippets.py > get_all_functions
18
19 def get_all_functions():
20     funcs = []
21     func = getFirstFunction()
22     while func is not None:
23         funcs.append(func)
24         func = getFunctionAfter(func)
25     return funcs
```

# Interpreter Tips ~ import ~

---

- ただし、そのままとインポートされたスクリプト側からGhidra APIを参照できないので、次のimport文を必ずスクリプトの最初に記述しておく！（重要）

```
Ghidra > ghidra_scripts >  snippets.py  
1  from __main__ import *  
2
```

# Interpreter Tips ~ reload ~

- すでにimport済みのスクリプトを編集した後再度importするには、**reload** を使う
  - 例えば、一度importしたがエラーが出たのでスクリプト内の関数を書き換えた場合など
  - Pythonは一度importしたコードをコンパイルしてキャッシュしているので、importだけでは更新が反映されないのが原因

```
Python - Interpreter
>>> import snippets
>>> snippets.test_func()    ← この後のタイミングで編集
'first run'
>>> reload(snippets)      ← reloadで編集を反映
<module 'snippets' from 'C:\Ghidra\ghidra_scripts\snippets.py'>
>>> snippets.test_func()
'reloaded!'
```

編集された関数

```
def test_func():
    #return 'first run'
    return 'reloaded!'
```

# Interpreter Tips ~ reload ~

- ただし、`from name import *` でimportしている場合、↓のように`sys.module['name']`をreloadしてから再インポートする必要がある



```
Python - Interpreter
Press 'F1' for usage instructions
>>> from snippets import *
>>> test_func()
'first run'
>>> import sys
>>> reload(sys.modules['snippets']) ← reload
<module 'snippets' from 'C:\Ghidra\ghidra_scripts\snippets.py'>
>>> from snippets import *
>>> test_func()
'reloaded!'
```

← この後のタイミングで編集

← 再インポート

編集された関数

```
def test_func():
    #return 'first run'
    return 'reloaded!'
```

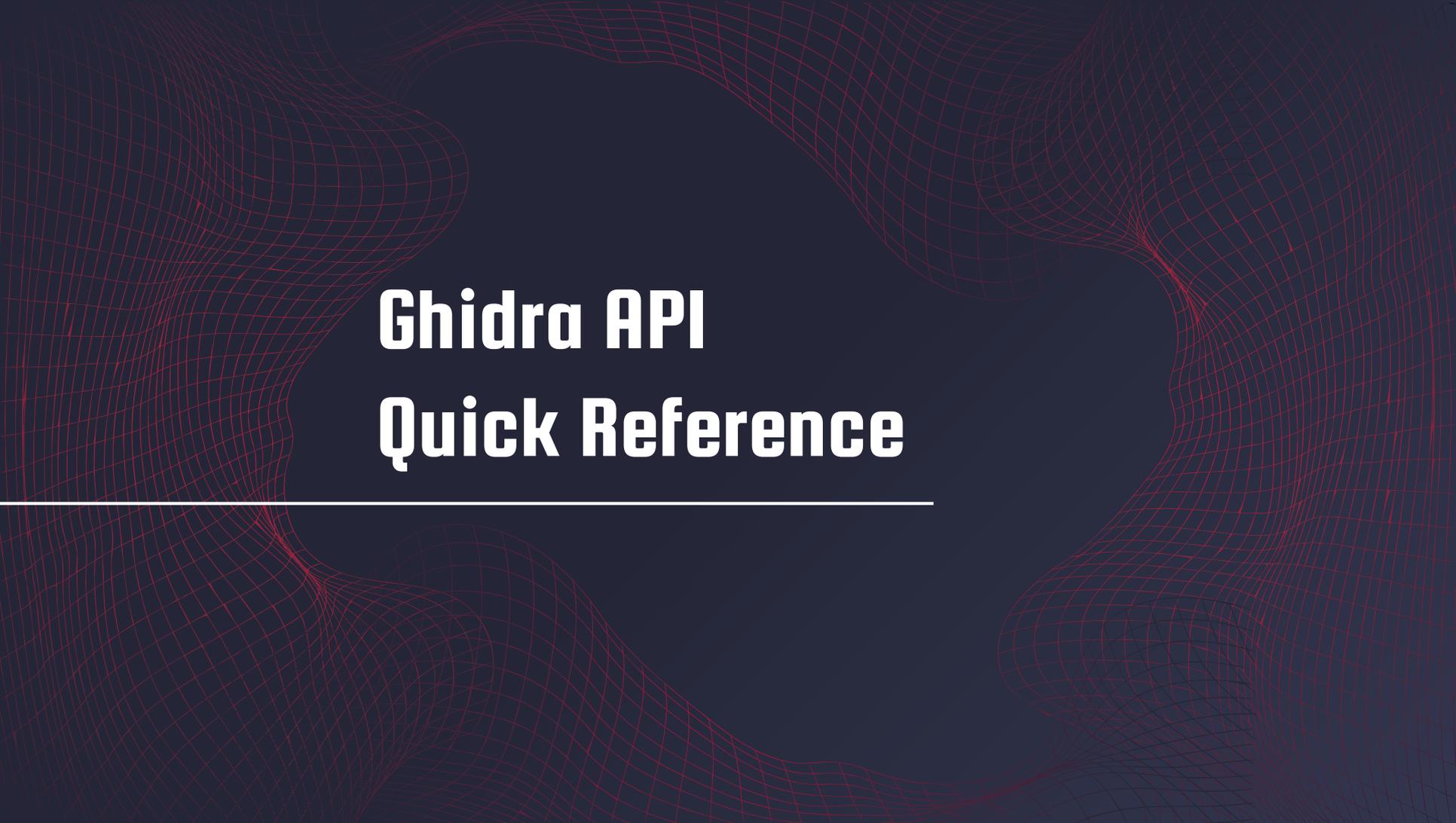
# Headless Analyzer

CLIからGhidra Scriptを操作するための機能

- ユーザインタラクションが不要なので、バッチ処理が可能
- バッチ処理により、ファイルのインポートや解析処理の自動化が可能

```
cmd
C:\Ghidra>%GHIDRA_INSTALL_DIR%\support\analyzeHeadless C:\Ghidra ghidra_project -process downloader.exe -scriptPath C:\Ghidra\ghidra_scripts -postScript analyze_downloader.py
INFO Using log config file: jar:file:/C:/Ghidra/ghidra_9.1.2_PUBLIC/Ghidra/Framework/Generic/lib/Generic.jar!/generic.log4j.xml (LoggingInitialization)
INFO Using log file: C:\Users\john\.ghidra\.ghidra_9.1.2_PUBLIC\application.log (LoggingInitialization)
INFO Loading user preferences: C:\Users\john\.ghidra\.ghidra_9.1.2_PUBLIC\preferences (Preferences)

(AutoAnalysisManager)
INFO REPORT: Analysis succeeded for file: /downloader.exe (HeadlessAnalyzer)
INFO SCRIPT: C:\Ghidra\ghidra_scripts\analyze_downloader.py (HeadlessAnalyzer)
[*] found http://allsafe.local/payload.bin at 00418030
[*] found http://allsafe.local/malware.bin at 00418060
INFO ANALYZING changes made by post scripts: /downloader.exe (HeadlessAnalyzer)
INFO REPORT: Post-analysis succeeded for file: /downloader.exe (HeadlessAnalyzer)
INFO REPORT: Save succeeded for processed file: /downloader.exe (HeadlessAnalyzer)
```



# **Ghidra API**

## **Quick Reference**

---

# FlatProgramAPI Class

## Program APIのラッパークラス

- [https://ghidra.re/ghidra\\_docs/api/ghidra/program/flatapi/FlatProgramAPI.html](https://ghidra.re/ghidra_docs/api/ghidra/program/flatapi/FlatProgramAPI.html)

## Fields

- `protected Program currentProgram`
  - ◇ 現在読み込んでいるプログラムに関する様々な情報を保持するフィールド、Program APIを提供する

## Methods

- `Address[] findBytes(Address start, java.lang.String byteString, int matchLimit)`
  - ◇ 指定したバイト列（Python2では文字列と同じ）を検索してヒットしたアドレス一覧を返す
- `byte[] getBytes(Address address, int length)`
  - ◇ 指定したaddressから指定したlength分をバイト列として取得
- `Function getFirstFunction()`
  - ◇ 現在読み込んでいるプログラムの最初の関数を取得
- `Instruction getInstructionAt(Address address)`
  - ◇ 指定したaddressの命令を取得
- `Instruction getInstructionBefore(Address address)`
  - ◇ 指定したaddressの一つ手前（低位アドレス）の命令を取得、一つ先（高位アドレス）の命令を取得するgetInstructionAfterもある
- `Function getFunctionContaining(Address address)`
  - ◇ 指定したaddressを含む関数を取得
- `Reference[] getReferencesTo(Address address)`
  - ◇ 指定したaddressの参照元（=addressを参照しているオブジェクト）一覧を取得
- `Address toAddr(int offset)`
  - ◇ 指定した値をAddressオブジェクトに変換

# FlatProgramAPI Class Use-Case:

## 01. Address Manipulation

---

- `Address toAddr(int offset)`
- `Address toAddr(long offset)`
- `Address toAddr(String address)`
  - ◇ 指定した値をAddressオブジェクトに変換

数値や文字列をAddressオブジェクトに変換し、比較

```
Python - Interpreter
>>> addr_from_long = toAddr(0x401000)
>>> addr_from_str = toAddr("0x401000")
>>> addr_from_long.equals(addr_from_str)
True
```

# Address Interface

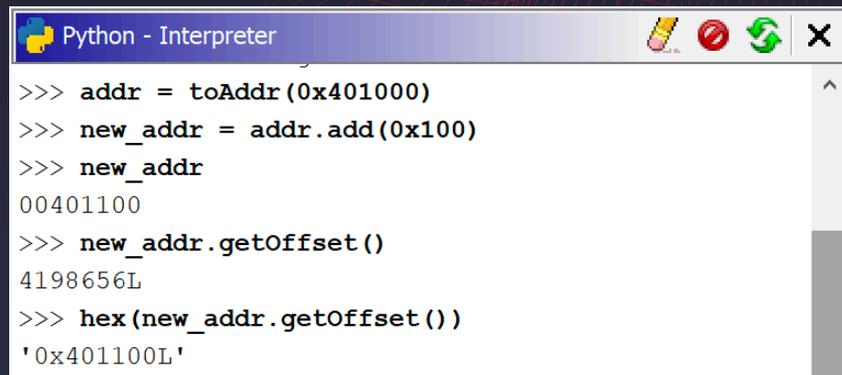
アドレスを表現するインターフェイス

- [https://ghidra.re/ghidra\\_docs/api/ghidra/program/model/address/Address.html](https://ghidra.re/ghidra_docs/api/ghidra/program/model/address/Address.html)

## Methods

- `Address add(long displacement)`
  - ◇ アドレスを引数`displacement`のサイズ分加算し、新しい`Address`を返す
- `boolean equals(java.lang.Object o)`
  - ◇ 引数`o`とアドレスを比較
- `long getOffset()`
  - ◇ アドレスのオフセットを`long`で返す
- `Address next()`
  - ◇ 次のアドレスを返す

オフセット値の取得



```
Python - Interpreter
>>> addr = toAddr(0x401000)
>>> new_addr = addr.add(0x100)
>>> new_addr
00401100
>>> new_addr.getOffset()
4198656L
>>> hex(new_addr.getOffset())
'0x401100L'
```

# FlatProgramAPI Class Use-Case:

## 02. Binary Manipulation

- `byte[] getBytes(Address address, int length)`
  - ◇ 指定したaddressから指定したlength分をバイト列として取得
- `int getInt(Address address)`
  - ◇ 指定したaddressから4バイトをintとして取得

特定のアドレスから、指定したサイズ分バイト列の読み出し

```
Python - Interpreter
>>> url_bytes = getBytes(toAddr(0x418030), 33)
>>> url_bytes
array('b', [104, 116, 116, 112, 58, 47, 47, 97, 108, 108, 115, 97,
102, 101, 46, 108, 111, 99, 97, 108, 47, 112, 97, 121, 108, 111, 97,
100, 46, 98, 105, 110, 0])
>>> url_bytes.toString()
'http://allsafe.local/payload.bin\x00'
```

# FlatProgramAPI Class Use-Case:

## 03. Search

- `Address find(java.lang.String text)`
  - ◇ プログラム内のコメントやラベル、コードユニットのやオペランドなどから、引数のtextで与えられた文字列を検索
  - ◇ 最初にマッチした結果のアドレスが返される
- `Address find(Address start, byte[] values)`
  - ◇ 第1引数にAddressオブジェクトが渡されると、引数startで指定したアドレス以降のプログラムのメモリからvalueで指定したバイト列を検索し、最初にマッチした結果のアドレスを1つだけ返す
  - ◇ 正規表現はサポートされていない
- `Address[] findBytes(Address start, java.lang.String byteString, int matchLimit)`
  - ◇ 引数startで指定したアドレス以降のメモリから、byteStringで指定された値を検索
  - ◇ このメソッドは、byteStringで正規表現を使うことも可能、例えば任意の4バイトをマッチさせたい場合「.{4}」といった表現を使用可能

レジスタへの  
MOV命令を検索

The image shows a screenshot of the Ghidra Python interpreter. On the left, a disassembly view shows several MOV instructions. The first MOV instruction at address 00401029 is highlighted with a red box. The Python interpreter on the right shows a call to `findBytes` with a regular expression pattern `'\\x8b.{1}\\xbc\\xfe\\xff\\xff'` and a match limit of `-1`. The result is an array containing the address `00401029`.

```

LAB_00401029  8b 85 bc      MOV     EAX,dword ptr [EBP + local_148]
              fe ff ff
0040102f  83 c0 0c      ADD     EAX,0xc
00401032  89 85 bc      MOV     dword ptr [EBP + local_148],EAX=>
              fe ff ff

LAB_00401038
00401038  3b 8d bc      MOV     ECX,dword ptr [EBP + local_148]
              fe ff ff
0040103e  83 39 00      CMP     dword ptr [ECX]=>PTR s_payload.ex
00401041  74 1e        JZ     LAB_00401061
00401043  8d 95 c0      LEA   EDX=>local_144,[EBP + 0xfffffec0]
              fe ff ff
00401049  52          PUSH  EDX
0040104a  8b 85 bc      MOV     EAX,dword ptr [EBP + local_148]
              fe ff ff
  
```

```

Python [CodeBrowser(2): ghidra_project/downloader.exe]
Help
Python - Interpreter
Python Interpreter for Ghidra
Based on Jython version 2.7.1 (default:0df7adblb397, Jun 30 2017, 19:02:43)
[OpenJDK 64-Bit Server VM (Oracle Corporation)]
Press 'F1' for usage instructions
>>> findBytes(None, '\\x8b.{1}\\xbc\\xfe\\xff\\xff', -1)
array(ghidra.program.model.address.Address, [00401029, 00401038, 0040104a])
  
```

# FlatProgramAPI Class Use-Case:

## 04. Function Manipulation

- `Function getFirstFunction()`
  - ◇ 現在のプログラムの最初の関数オブジェクトを取得
- `Function getFunctionAt(Address entryPoint)`
  - ◇ 引数entryPointで指定したアドレスから始まる関数オブジェクトを取得
- `Function getFunctionAfter(Address address)`
  - ◇ 引数addressで指定したアドレスで始まる関数の次の関数オブジェクトを取得
- `Function getFunctionBefore(Address address)`
  - ◇ 引数addressで指定したアドレスで始まる関数の前の関数オブジェクトを取得
- `Function getFunctionContaining(Address address)`
  - ◇ 引数addressで指定したアドレスが関数のアドレスレンジに含まれる場合、その関数オブジェクトを取得

プログラム内の  
全関数を列挙

```
Python - Interpreter
>>> def get_all_functions():
...     func = getFirstFunction()
...     while func is not None:
...         yield func
...         func = getFunctionAfter(func)
...
>>> funcs = get_all_functions()
>>> for func in funcs:
...     print('func: {} at {}'.format(func.getName(), func.getEntryPoint()))
...
func: FUN_00401000 at 00401000
func: FUN_004011e0 at 004011e0
func: FUN_00401220 at 00401220
func: FUN_00401260 at 00401260
```

# Function Interface

---

関数を表現するインターフェイス

- [https://ghidra.re/ghidra\\_docs/api/ghidra/program/model/listing/Function.html](https://ghidra.re/ghidra_docs/api/ghidra/program/model/listing/Function.html)

## Methods

- `Address` `getEntryPoint()`
  - ◇ 関数のエントリーポイント（先頭アドレス）を取得
- `java.lang.String` `getName()`
  - ◇ 関数の名前を取得
- `Parameter[]` `getParameters()`
  - ◇ 関数の引数を `Parameter` オブジェクトで取得

# FlatProgramAPI Class Use-Case:

## 05. Instruction Manipulation

- `Instruction` `getFirstInstruction(Function function)`
  - ◇ 引数functionで指定した関数オブジェクトに含まれる、最初の命令を取得
- `Instruction` `getInstructionAt(Address address)`
  - ◇ 引数addressで指定したアドレスの命令を取得
- `Instruction` `getInstructionAfter(Instruction instruction)`
  - ◇ 引数instructionで指定した命令の次（高位アドレス）の命令を取得
- `Instruction` `getInstructionBefore(Instruction instruction)`
  - ◇ 引数instructionで指定した命令の1つ前（低位アドレス）の命令を取得
  - ◇ 関数呼び出しを行うCALL 命令に対する引数をたどる際などに利用可能

特定の関数内の全ての命令を取得

```
Python - Interpreter
>>> def get_instructions_in_func(func):
...     inst = getFirstInstruction(func)
...     while inst is not None:
...         if getFunctionContaining(inst.getAddress()) == func:
...             yield inst
...             inst = getInstructionAfter(inst)
...
>>> entry_func = getFirstFunction()
>>> insts = get_instructions_in_func(entry_func)
>>> for inst in insts:
...     print('{} {}'.format(inst.getAddressString(True, True), inst))
...
.text:00401000 PUSH EBP
.text:00401001 MOV EBP,ESP
.text:00401003 SUB ESP,0x19c
.text:00401009 MOV EAX,[0x00418090]
.text:0040100e XOR EAX,EBP
```

# Instruction Interface

---

## 命令を表現するインターフェイス

- [https://ghidra.re/ghidra\\_docs/api/ghidra/program/model/listing/Function.html](https://ghidra.re/ghidra_docs/api/ghidra/program/model/listing/Function.html)

## Methods

- `java.lang.Object[] getOpObjects(int opIndex)`
  - ◇ 命令に渡されているオペランドのうち、`opIndex`で指定されたインデックスのオブジェクト (`Address`, `Scalar` など) 一覧を取得
- `java.lang.String getMnemonicString()`
  - ◇ 正確には `CodeUnit` インターフェイスのメソッド
  - ◇ 命令のニーモニックを文字列として取得
- `PcodeOp[] getPcode()`
  - ◇ 命令を P-Code という中間表現に変換
- `Register getRegister(int opIndex)`
  - ◇ 命令に渡されているオペランドのうち、`opIndex`で指定されたインデックスのレジスタオブジェクトを取得

# FlatProgramAPI Class Use-Case:

## 06. Reference Manipulation

Ghidraはデータやコードなどのオブジェクト間の関係性を「参照」という情報として保持



- `Reference[] getReferencesTo(Address address)`
  - ◇ 引数addressで指定したアドレスへの参照を持つコードユニットのアドレス一覧を取得
  - ◇ つまり、後方参照（クロスリファレンス）を取得
- `Reference[] getReferencesFrom(Address address)`
  - ◇ 引数addressで指定したコードユニットのアドレスが参照している先のアドレス、つまり前方参照の一覧を取得

```
Python - Interpreter
>>> config_addr = toAddr(0x418000)
>>> xrefs = getReferencesTo(config_addr)
>>> xrefs
array(ghidra.program.model.symbol.Reference, [From: 0040024c To: 00418000 Type: DATA
Op: 0 DEFAULT, From: 0040101d To: 00418000 Type: DATA Op: 1 ANALYSIS, From: 0040103e
To: 00418000 Type: READ Op: 0 ANALYSIS, From: 00401050 To: 00418000 Type: DATA Op: 0
ANALYSIS])
>>> for xref in xrefs:
...     print('reference from {}'.format(xref.getFromAddress()))
...
reference from 0040024c
reference from 0040101d
reference from 0040103e
reference from 00401050
```

# Reference Interface

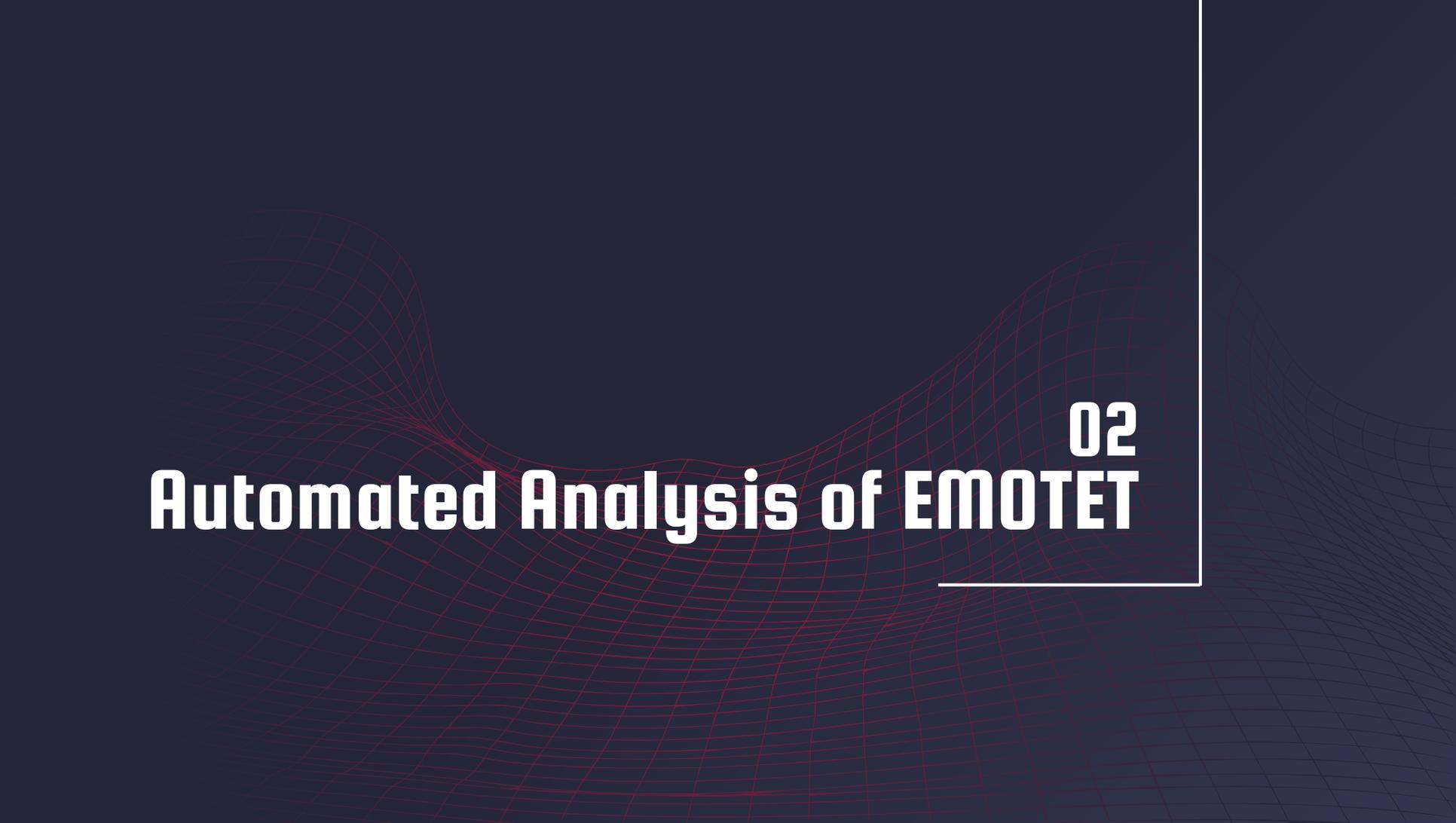
---

参照を表現するインターフェイス

- [https://ghidra.re/ghidra\\_docs/api/ghidra/program/model/symbol/Reference.html](https://ghidra.re/ghidra_docs/api/ghidra/program/model/symbol/Reference.html)

## Methods

- **Address** `getFromAddress()`
  - ◇ 参照しているコードのアドレス、つまり後方参照のアドレスを取得
- **Address** `getToAddress()`
  - ◇ 参照の「宛先」アドレス、つまり前方参照のアドレスを取得
- **RefType** `getReferenceType()`
  - ◇ 参照のタイプを取得
  - ◇ 参照タイプ: [https://ghidra.re/ghidra\\_docs/api/ghidra/program/model/symbol/RefType.html](https://ghidra.re/ghidra_docs/api/ghidra/program/model/symbol/RefType.html)



# Automated Analysis of EMOTET

02

# Goals of Malware Analysis

「解析」といっても目的（期待されるアウトプット）はさまざま

## Detection

不正かどうかの判定、およびマルウェアのファミリー名やカテゴリを特定する。この作業により、マルウェアの目的や関連する攻撃者を認識することができる（可能性がある）。

## Indicator of Compromise

ネットワークやホスト内において、侵害の有無を判断するための情報を特定する。作成されるファイル名やレジストリ名、通信先情報などが該当する。



今回は通信先情報  
の抽出を目的とする

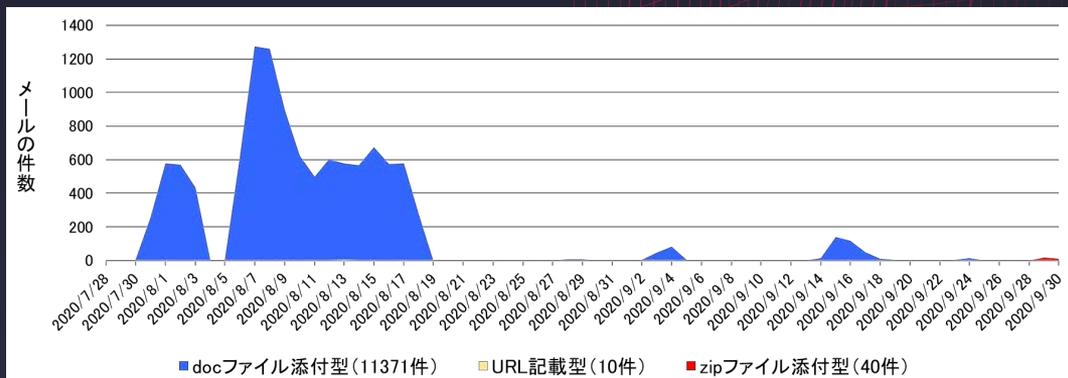
## Report

マルウェアの挙動・機能を含む解析レポート。Detection/IoCを含み、マルウェアの目的や背後の攻撃者について言及する。主に顧客向けに文章で記述される。

# EMOTET

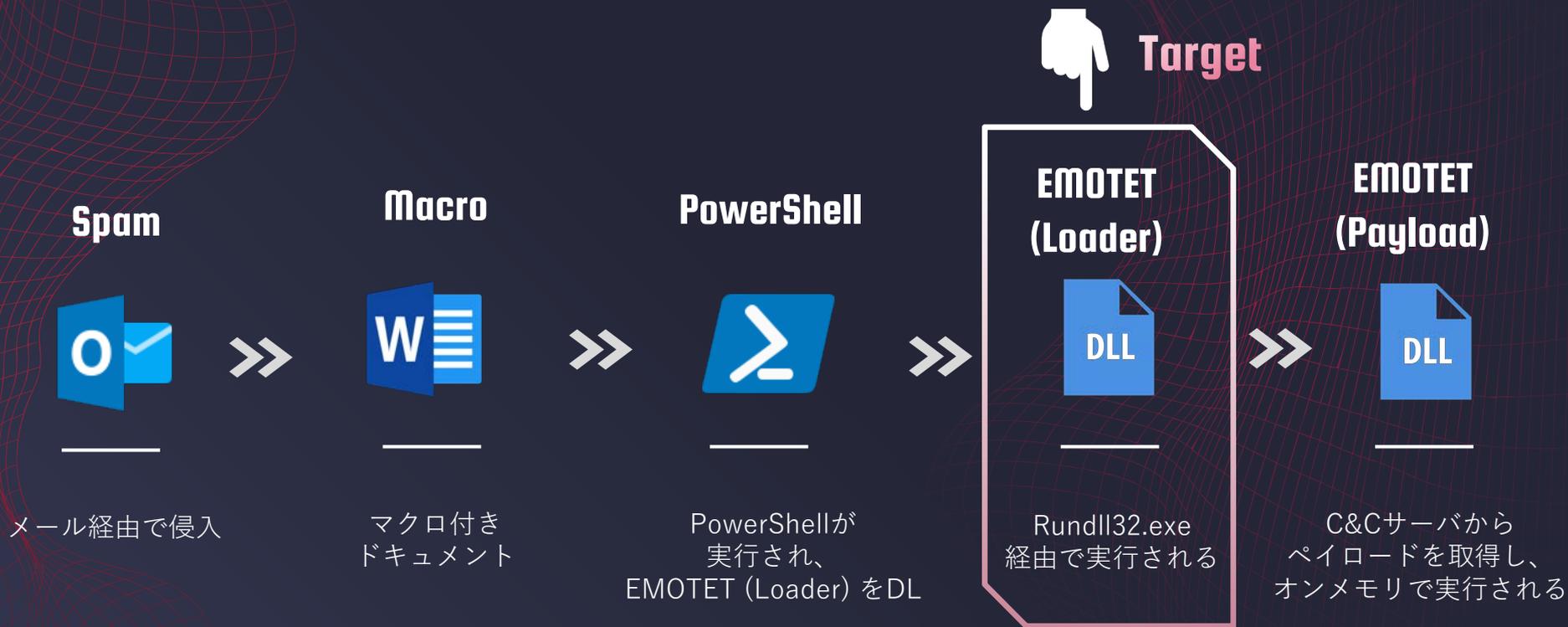
- 近年猛威をふるう、情報窃取兼ダウンロード型マルウェア
  - 大量のスパム経由で配布されることが知られている
  - Human Operated Ransomwareの起点となることもある
- とにかく数が多い
  - 大量の亜種が短時間で配布される
  - かつそれぞれの検体が大量のC&C通信先をもつ

## NICTによるEMOTETスパムの着信数推移



<https://blog.nictcr.jp/2020/10/emotet-mail-202007-202009/>

# How EMOTET works



# Analysis Steps

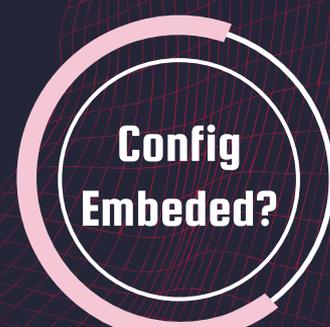
以下のような点を解析によって明らかにし、スクリプトに落とし込む



- ✓ パックされているか？
- ✓ パッカーのロジック



- ✓ 文字列は難読化・暗号化されているか？
- ✓ 鍵は？
- ✓ API呼び出しは難読化されているか？

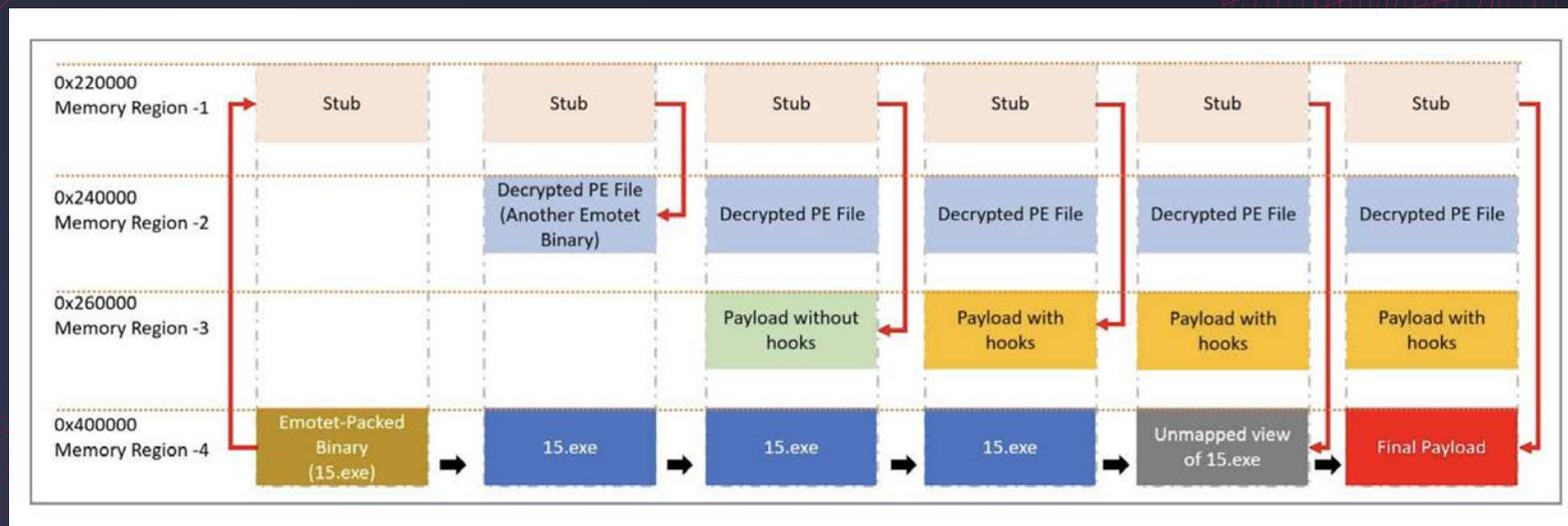


- ✓ C&Cサーバなどの情報はどこに/どのような形式で格納されているか？
- ✓ どうやって自動抽出するか？

# Step 0: Determine Packing Logic and Do Unpack

他の多くのマルウェアと同じく、EMOTETもパックされており、メモリ上でペイロードが復号されて実行される

(ある時期の) EMOTETのアンパッキングロジック



# Step 0: Determine Packing Logic and Do Unpack

アンパックの手法はさまざま

## ■ 手動アンパック

- デバッガーを使ってアンパック
- 実行後、マルウェア自身によってアンパックされた領域をダンプ

## ■ ツールによるアンパック

- マルウェアを実行、メモリ上からアンパック後の領域を機械的に抽出
  - ✓ PE-SIEVE / Hollows Hunter

## ■ 自動アンパック

- ツールの自動実行
  - ✓ tknk\_scanner
  - ✓ CAPE

```

C:\Users\user>cmd /c tasklist /v /fi "image file(*)" /s /b /u /p /q
Scanning workingset: 284 memory regions.
[*] Workingset scanned in 125 ms
[*] Report dumped to: process_3168
[*] Dumped module to: C:\Users\%user%\Desktop\process_3168\76440000.kerne132.dll as UNMAPPED
[*] Dumped module to: C:\Users\%user%\Desktop\process_3168\23200000.shc as VIRTUAL
[*] Report modified to: process_3168
[*] Report dumped to: process_3168
---
PID: 3168
---
SUMMARY:
Total scanned: 50
Skipped: 0
-
Hooked: 1
Replaced: 0
Hdrs Modified: 0
IAT Hooks: 0
Implanted: 1
Implanted PE: 1
Implanted shc: 0
Unreachable files: 0
Other: 0
Total suspicious: 2
  
```

The screenshot shows the tknk\_scanner web interface. At the top, there are navigation buttons for 'Scan' and 'Reset'. The main content area displays a large green checkmark and the word 'Success!'. Below this, there is a 'Submit File' button and a 'Dump Files' section. The 'Result' section shows a green checkmark and the text 'Success!'. The 'Submit File' section shows a file named 'test.exe' with a size of 476,000 bytes. The 'Dump Files' section shows a table of files with columns for 'File Name', 'Size', and 'Detect Rule'.

File Name	Size	Detect Rule
test.exe	476,000	PE-SIEVE
30753.exe	21,518	PE-SIEVE
520000.dll	128,268	PE-SIEVE
360000.dll	26,168	PE-SIEVE
280000.exe	32,388	PE-SIEVE
2300000.dll	396,628	PE-SIEVE

# Quick Unpack with Hollows Hunter

- パックされたEMOTETを実行し、Hollows Hunterでアンパック後のEMOTETをダンプ
- 具体的な手順はAppendixを参照

```
C:\Users\john>rundll32.exe C:\Users\john\Desktop\urcwzowo.xck,Control_RunDLL
```

```
C:\Users\john\Desktop>C:\Users\john\Desktop\hollows_hunter.exe /pid 1336
```

```
HollowsHunter v.0.2.6 (x64)
```

```
Built on: Apr 13 2020
```

```
using: PE-sieve v.0.2.6.0
```

```
>> Scanning PID: 1336 (rundll32.exe)
```

```
[-] Invalid address of relocations
```

```
[-] Invalid address of relocations
```

```
>> Detected: 1336
```

```
-----
```

```
SUMMARY:
```

```
Scan at: 01/13/21 03:29:17 (1610476157)
```

```
Finished scan in: 3516 milliseconds
```

```
[+] Total Suspicious: 1
```

```
[+] List of suspicious:
```

```
[0]: PID: 1336, Name: rundll32.exe
```

# Unpacked EMOTET

---

- 本ワークショップでは、アンパック済みのEMOTETを使用します

## Name

-  emotet.dll.gzf → **メイン解析用**
-  emotet-2.dll.gzf → **比較用**



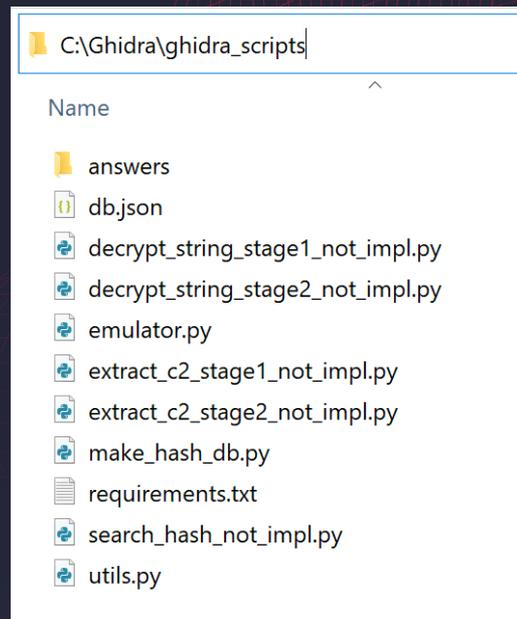
**Step I:**  
**Surface Analysis**

---

# Resources

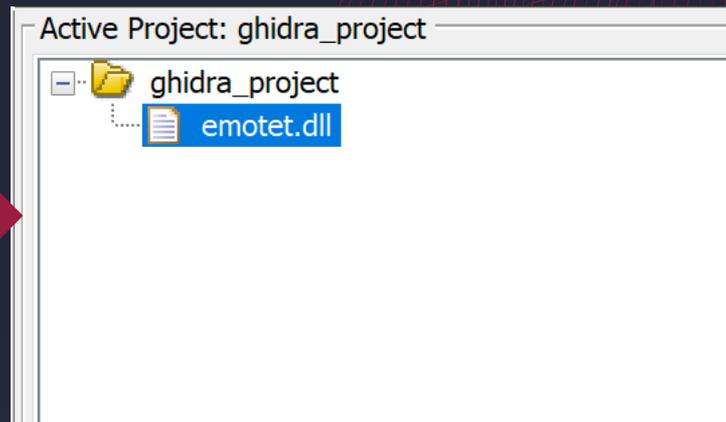
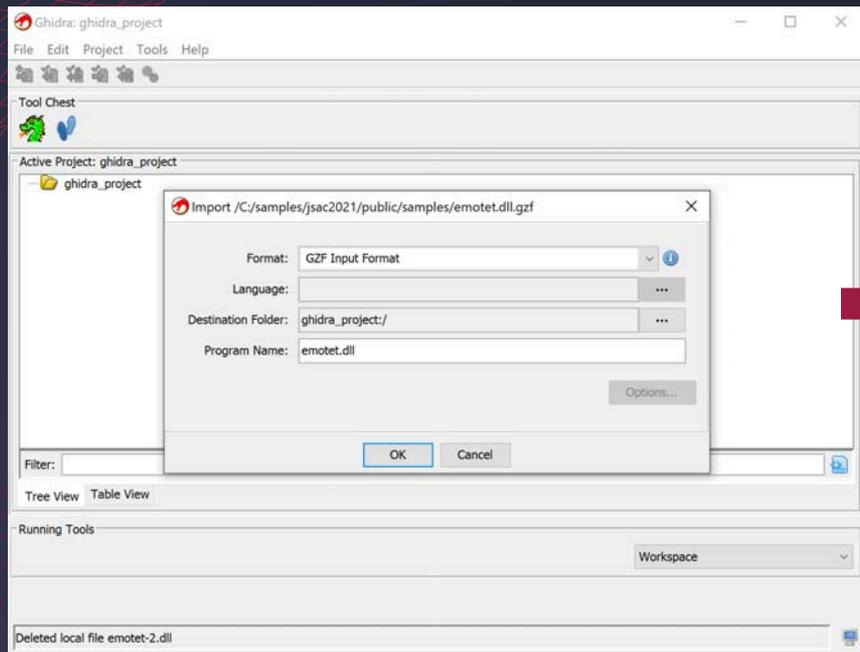
- 使用するサンプル等はJSAC参加者にのみSlackで共有済み
- 共有されたzipを展開し、以下手順を実施
  1. **scripts**: 配下のファイルをC:¥Ghidra¥ghidra\_scripts配下にコピー
    - scripts¥answers: 演習の回答なので、極力見ない
  2. **emotet.dll.gzf**: Ghidraにドラッグ&ドロップでインポート
  3. **emotet-2.dll.gzf**: Ghidraにドラッグ&ドロップでインポ
  4. **winapi\_32.gdt**: C:¥Ghidra直下にコピー

C:¥Ghidra¥ghidra\_scripts配下が以下のような構成になっていればOK



# Setup (sample)

- 配布したemotet.dll.gzfをGhidraの既存プロジェクトにドラッグ&ドロップしてインポート
  - 途中ポップアップが出たら「OK」



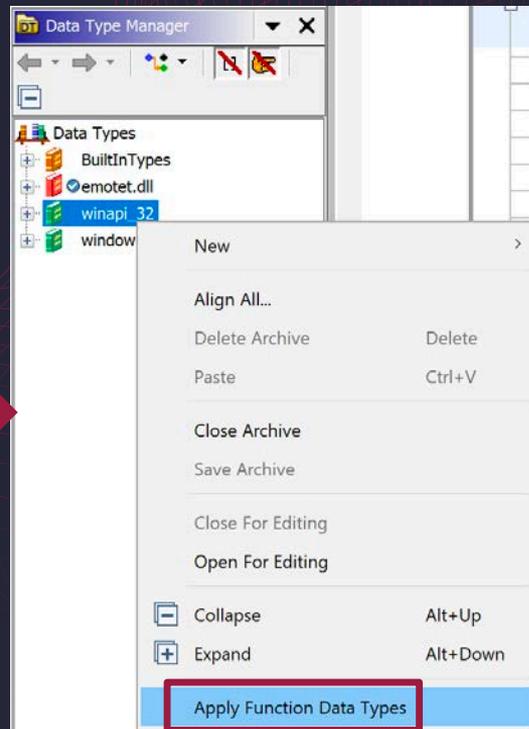
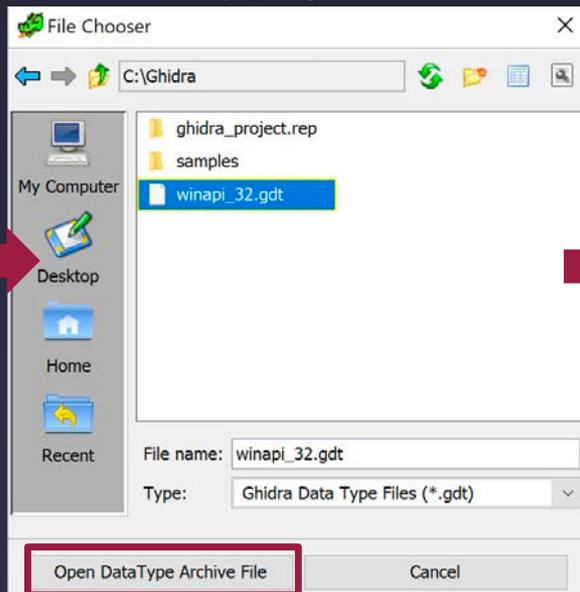
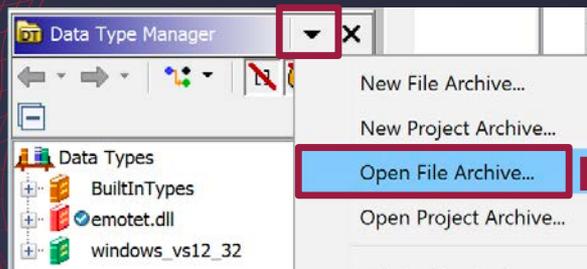
# Setup (GDT)

- Ghidraの標準型データにはwininet.h、winhttp.hのAPI情報がない
  - Data Type Managerでwinapi\_32.gdtをロード

ロード完了後、winapi\_32を  
右クリックして  
Apply Function Data Types

配布したファイルに含まれる  
winapi\_32.gdtをロード

▼ボタンを押してOpen File Archive



# Setup (scripts)

---

## スクリプトの設置

- 以下スクリプトが**C:¥Ghidra¥ghidra\_scripts**配下にコピーされているか念のため確認
  1. `utils.py`: 演習等で再利用可能なユーティリティ関数群
  2. `search_hash_not_impl.py`: 演習用穴あきスクリプト
  3. `make_hash_db.py`: APIハッシュ用DB作成スクリプト
  4. `db.json`: APIハッシュ用DB
  5. `emulator.py`: エミュレータを使ったハッシュ計算スクリプト
  6. `decrypt_string_stage1_not_impl.py`: 演習用穴あきスクリプト
  7. `decrypt_string_stage2_not_impl.py`: 演習用穴あきスクリプト
  8. `extract_c2_stage1_not_impl.py`: 演習用穴あきスクリプト
  9. `extract_c2_stage2_not_impl.py`: 演習用穴あきスクリプト

# Exports Functions

- DLL verのEmotetはrundll32.exeを使ってエクスポートされた関数から実行される

The screenshot displays the Windows Task Manager interface, showing a process tree. The 'Processes' tab is active, and the 'Only important' checkbox is checked. The process tree is expanded to show the following processes:

- 2668 WINWORD.EXE** /n "C:\Users\admin\AppData\Local\Temp\Déclarations prêtes Pour visualiser.doc" (3k files, 1k folders, 117 sub-processes)
- 2952 WMI cmd.exe** cmd cmd cmd /c msg %username% /v Word experienced an error trying to open the file. & P^Ow^er^she^L^L -w ... (163 files, 6 folders, 28 sub-processes)
- 3988 msg.exe** admin /v Word experienced an error trying to open the file. (42 files, 0 folders, 30 sub-processes)
- 1908 powershell.exe** -w hidden -ENCOD IAAgACQAbQBqAE8AIAA9ACAAIABbAFQAWQBwAEUAXQAoACIAewA1AH0AewAwAH... (2k files, 309 folders, 228 sub-processes)
- 3528 rundll32.exe** C:\Users\admin\Ph8oybt\Y3yg\_u0\V64L.dll Control\_RunDLL (387 files, 46 folders, 116 sub-processes)
- 3224 rundll32.exe** "C:\Users\admin\AppData\Local\Ozspuecmz\urcwzowo.xck",Control\_RunDLL (262 files, 21 folders, 70 sub-processes)

The 'emotet' process is also visible in the tree, associated with the 3224 rundll32.exe process.

# Brief Analysis (code)

## ■ Garbage Code

- 実行に影響を及ぼさない無駄なコード
- Ghidraのデコンパイラがいい感じに端折ってくれる

```

FUN_1001900c                                     XREF[2]:
1001900c 55          PUSH     EBP
1001900d 8b ec      MOV     EBP,ESP
1001900f 83 e4 f9   AND     ESP,0xfffffff8
10019012 81 ec 40 ... SUB     ESP,0x240
10019018 83 a4 24 ... AND     dword ptr [ESP + local_a8*0x4],0x0
10019020 33 d2     XOR     EDX,EDX
10019022 83 a4 24 ... AND     dword ptr [ESP + local_a0],0x0
1001902a b9 0c 8f ... MOV     ECX,0x1e08f0c
1001902f c7 84 24 ... MOV     dword ptr [ESP + local_a8],0x3b3e34
1001903a c7 84 24 ... MOV     dword ptr [ESP + local_1c4],0x1bd7
10019045 81 8c 24 ... OR      dword ptr [ESP + local_1c4],0x47dfdea
10019050 81 84 24 ... ADD     dword ptr [ESP + local_1c4],0xc6ff
1001905b 6b 84 24 ... IMUL   EAX,dword ptr [ESP + local_1c4],0x5e
10019063 53        PUSH    EBX
10019064 55        PUSH    EBP
10019065 56        PUSH    ESI
10019066 57        PUSH    EDI
10019067 89 84 24 ... MOV     dword ptr [ESP + local_1c4],EAX
1001906e 81 b4 24 ... XOR     dword ptr [ESP + local_1c4],0x647d41bd
10019079 c7 84 24 ... MOV     dword ptr [ESP + local_bc],0xaf04
10019084 81 8c 24 ... OR      dword ptr [ESP + local_bc],0xe49f97e0
1001908f 81 b4 24 ... XOR     dword ptr [ESP + local_bc],0xe49f97e7
1001909a c7 84 24 ... MOV     dword ptr [ESP + local_140],0x8f5e
100190a5 81 b4 24 ... XOR     dword ptr [ESP + local_140],0x9f735941
100190b0 c1 a4 24 ... SHL     dword ptr [ESP + local_140],0x3
100190b8 81 b4 24 ... XOR     dword ptr [ESP + local_140],0xb9e8485
100190c3 c7 84 24 ... MOV     dword ptr [ESP + local_ec],0xbd1f
100190ce 81 84 24 ... ADD     dword ptr [ESP + local_ec],0xffff8eb6
100190d9 81 b4 24 ... XOR     dword ptr [ESP + local_ec],0x8ee9
100190e4 c7 84 24 ... MOV     dword ptr [ESP + local_16c],0x603f
100190ef 6b 84 24 ... IMUL   EAX,dword ptr [ESP + local_16c],0x30
100190f7 6a 58     PUSH    0x58

```

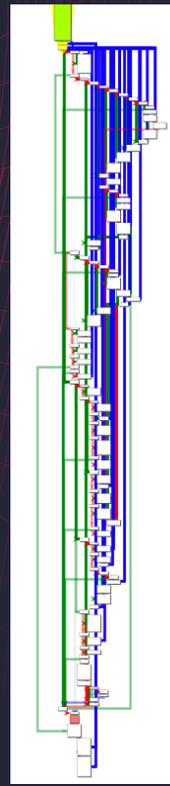
## ■ Control Flow Flattening

- Control Flow Obfuscationの手法
- 直感的に命令フローが識別しづらい

```

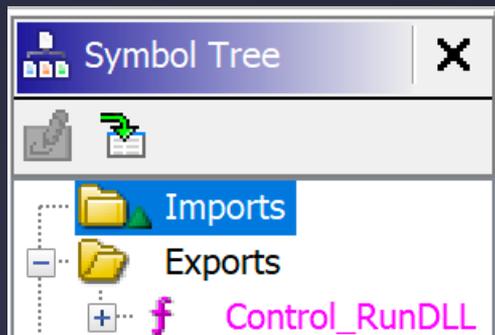
iVar4 = 0x1e08f0c;
iVar3 = 0x7e01;
uVar6 = 0x3b3e34;
LAB_1001a182:
do {
    iVar1 = iVar4;
    if (iVar4 < 0x14cb6629) {
        if (iVar4 == 0x14cb6628) {
            local_98 = FUN_10006f31(&local_94,0xc3,0x3b0a,(char)iVar4,0x5dd7,0x2768);
            FUN_1000e10b(&local_98);
            FUN_10007c8f(0x111b);
            iVar4 = 0x15c12bf9;
            goto LAB_1001a182;
        }
    }
    if (iVar4 < 0xa5688f5) {
        if (iVar4 == 0xa5688f4) {
            iVar4 = FUN_10014032();
            if (iVar4 == 0) {
                return;
            }
            iVar4 = 0x108d7d72;
            goto LAB_1001a182;
        }
    }
    if (iVar4 < 0x4ebc5d3) {
        if (iVar4 == 0x4ebc5d2) {
            free_mem(local_54[0],0x3d,0x59f8,0x72a5);
            iVar4 = 0x2eb577a4;
            goto LAB_1001a182;
        }
    }
    if (iVar4 == 0x13528f8) {

```



# Brief Analysis (API/strings)

- アンパック後のEMOTETは Import Tableが空
  - 動的にAPIを解決している可能性



- APIに関連する文字列もなし
  - API含む文字列が暗号化・難読化されている可能性 🙄

The image shows a 'Defined Strings' window with 9 items. The table lists memory locations, string values, and string representations.

Location	String Value	String Representation
10000000		
100000c8		
100001c0		
100001e8		
10000210		
10000238		
1001e04c	X.dll	"X.dll"
1001e052	Control_RunDLL	"Control_RunDLL"
1001e061	RunDLL	"RunDLL"

# Where to start?

- APIの動的解決や文字列の復号に使用されている関数は実行時に何度も呼び出される
- そのため、まずは各関数が呼び出されている回数を調べて、呼び出し回数が多い関数をみていくことにする

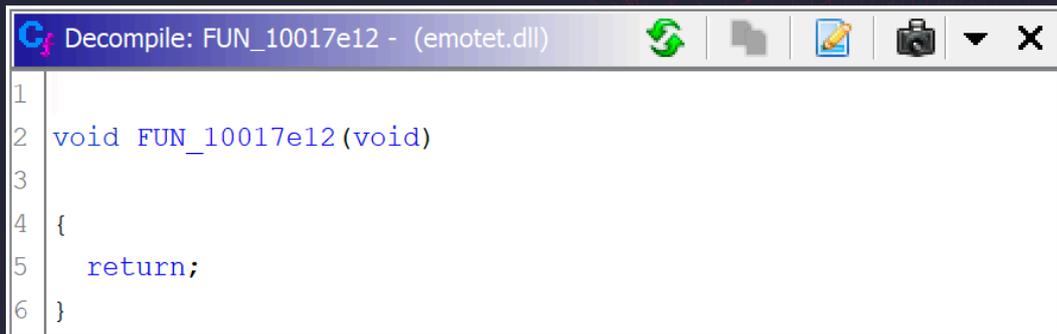
utils.pyに定義済みのget\_func\_xref\_count関数を実行し、呼び出し回数の多い関数を列挙

```
Python - Interpreter
Python Interpreter for Ghidra
Based on Jython version 2.7.2 (v2.7.2:925a3cc3b49d, Mar 21 2020, 10:03:58)
[OpenJDK 64-Bit Server VM (Oracle Corporation)]
Press 'F1' for usage instructions
>>> from utils import *
>>> top10 = get_func_xref_count(n=10)
```

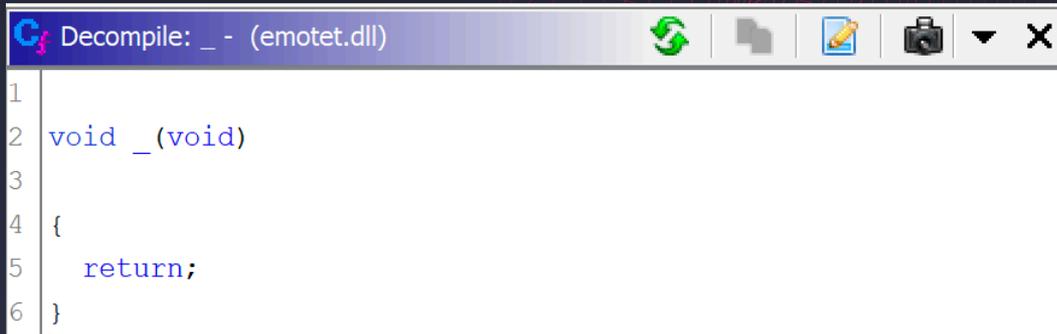
```
Python - Interpreter
>>> for called_func in top10:
...     print('{} is called {} times'.format(called_func['name'], called_func['count']))
...
FUN_10017e12 is called 160 times
FUN_10007d5b is called 107 times
FUN_10017f68 is called 34 times
FUN_1000d4e0 is called 33 times
FUN_10007c8f is called 28 times
FUN_1000732d is called 22 times
FUN_10002ec6 is called 16 times
FUN_100010cf is called 10 times
FUN_100150ba is called 8 times
FUN_10012199 is called 8 times
```

# FUN\_10017e12

- 最も多く呼び出されている関数
- RETするだけのジャンク関数
- 「\_」など適当な名前に変えておく



```
Decompile: FUN_10017e12 - (emotet.dll)
1
2 void FUN_10017e12(void)
3
4 {
5     return;
6 }
```



```
Decompile: _ - (emotet.dll)
1
2 void _(void)
3
4 {
5     return;
6 }
```

# FUN\_10007d5b

- 2番目に多く呼び出されている関数
- 4つの引数を受け取るが、実際は第二引数は未使用
- DAT\_1001fa20はグローバル変数の配列

FUN\_10007d5bの呼び出し例 (@0x10017e02)

```
pcVar1 = (code *) FUN_10007d5b(0xf5, 0xb, 0x90bdc3a8, 0xb7c2e83f);
```

Decompile: FUN\_10007d5b - (emotet.dll)

```

1
2 undefined4 __cdecl FUN_10007d5b(int param_1, undefined4 param_2, uint param_3, uint param_4)
3
4 {
5     int iVar1;
6     char *pcVar2;
7
8     if (*(int *)(&DAT_1001fa20 + param_1 * 4) == 0) {
9         iVar1 = FUN_10004ae6(param_3);
10        pcVar2 = FUN_10012794(0x7ef7, iVar1, 0x7a71, param_4);
11        *(char **)(&DAT_1001fa20 + param_1 * 4) = pcVar2;
12    }
13    return *(undefined4 *)(&DAT_1001fa20 + param_1 * 4);
14 }

```

各要素が4バイトの配列?

第一引数は配列のインデックス

# FUN\_10007d5b

- 変数名・型名の変更

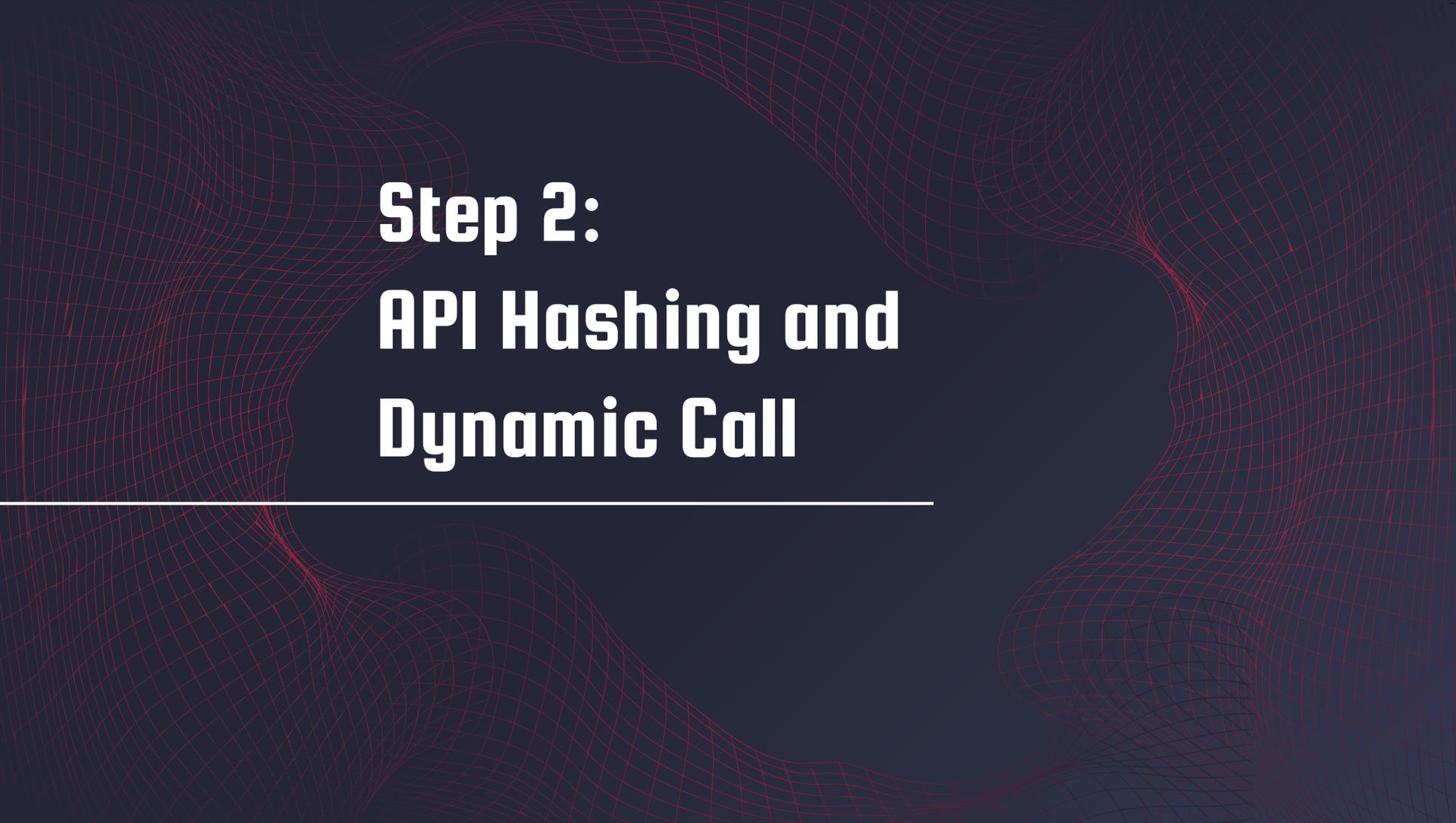
BEFORE	AFTER (name)	AFTER (type)
DAT_1001fa20	g_array	int[0x64]
param_1	index	N/A
param_2	_	N/A
param_3	hex_value1	N/A
param_4	hex_value2	N/A

少し見やすくしたFUN\_10007d5b

```

Decompile: FUN_10007d5b - (emotet.dll)
1
2 int __cdecl FUN_10007d5b(int index,undefined4 __,uint hex_value1,uint hex_value2)
3
4 {
5     int iVar1;
6     char *pcVar2;
7
8     if (g_array[index] == 0) {
9         iVar1 = FUN_10004ae6(hex_value1);
10        pcVar2 = FUN_10012794(0x7ef7,iVar1,0x7a71,hex_value2);
11        g_array[index] = (int)pcVar2;
12    }
13    return g_array[index];
14 }

```



**Step 2:**  
**API Hashing and**  
**Dynamic Call**

---

# How Windows APIs are resolved

## ■ 暗黙的な解決

- コンパイル時に、必要なAPIがインポートテーブルに定義され、実行時にOSが暗黙的にDLLをロードし、APIを解決する
- 普通にコンパイルするとこの方法で解決される

## ■ 明示的な解決（動的API解決）

- `LoadLibrary` / `GetProcAddress` を用いて実行時に動的にAPIを呼び出す
- API名から簡単に利用するAPIが識別されてしまう

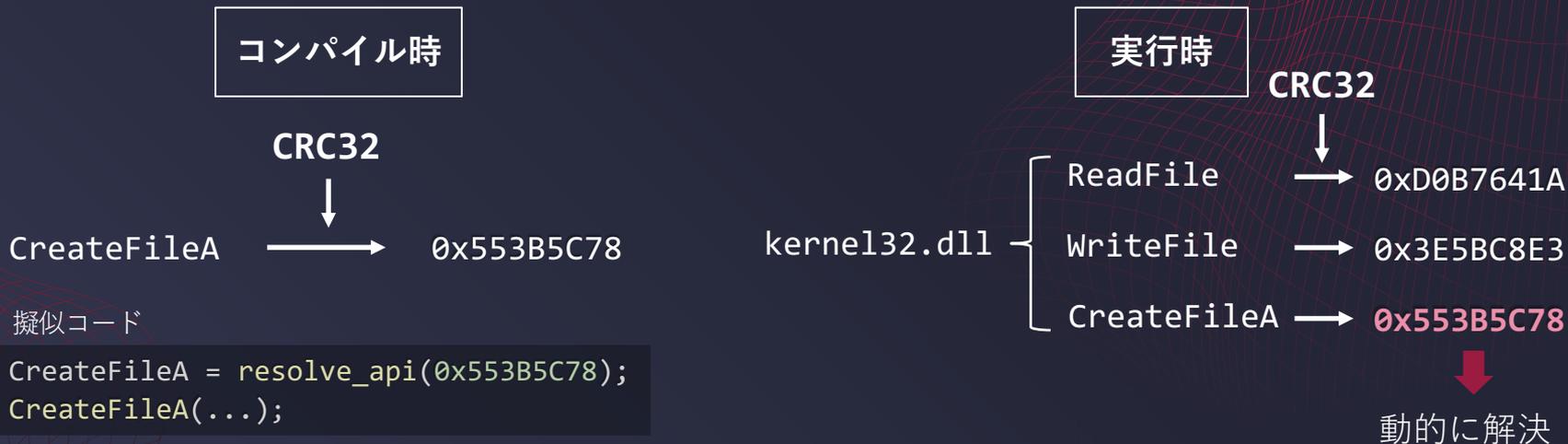
MessageBoxAを明示的に解決して呼び出す例

```
FARPROC MessageBoxA = GetProcAddress(LoadLibraryA("user32.dll"), "MessageBoxA");  
MessageBoxA(NULL, "Hello world", "Hello", MB_OK);
```

# API Hashing

- 動的にWindows APIを解決する際に用いられる耐解析手法の一つ
- 既知、もしくは独自実装のハッシュ関数でAPI名をハッシュ化して保持しておき、実行時に、エクスポートされたAPI名を列挙してハッシュ関数の計算結果が同じAPIを動的に呼び出す

例: CRC32を使用してAPI名をハッシュ化し、実行時に比較して動的に解決



# FUN\_10004ae6

- FUN\_10002b48の呼び出し
  - in\_FS\_OFFSET (=FSレジスタ)へのアクセス
  - x86 の場合、FSレジスタにはTEB (Thread Environment Block) へのポインタが格納されている
- FUN\_100180eaの呼び出し

```

Decompile: FUN_10004ae6 - (emotet.dll)
1
2 undefined4 __cdecl FUN_10004ae6(uint param_1)
3
4 {
5     int iVar1;
6     uint uVar2;
7     undefined4 *puVar3;
8     undefined4 *puVar4;
9
10    iVar1 = FUN_10002b48();
11    puVar4 = (undefined4 *) (*(int *) (iVar1 + 0xc) + 0xc);
12    puVar3 = (undefined4 *) *puVar4;
13    while( true ) {
14        if (puVar3 == puVar4) {
15            return 0;
16        }
17        uVar2 = FUN_100180ea((ushort *)puVar3[0xc], 2, 0x639b);
18        if ((uVar2 ^ 0x1fc325da) == param_1) break;
19        puVar3 = (undefined4 *) *puVar3;
20    }
21    return puVar3[6];
22 }
  
```

```

Decompile: FUN_10002b48 - ...
1
2 undefined4 FUN_10002b48(void)
3
4 {
5     int in_FS_OFFSET;
6
7     return *(undefined4 *) (in_FS_OFFSET + 0x30);
8 }
  
```

# Thread Environment Block / Process Environment Block

## ■ Thread Environment Block (TEB)

- 現在のスレッドに関する情報を保持する構造体
- FSセグメントレジスタからアクセス可能  
(x86 Windowsにおいて)

## ■ Process Environment Block (PEB)

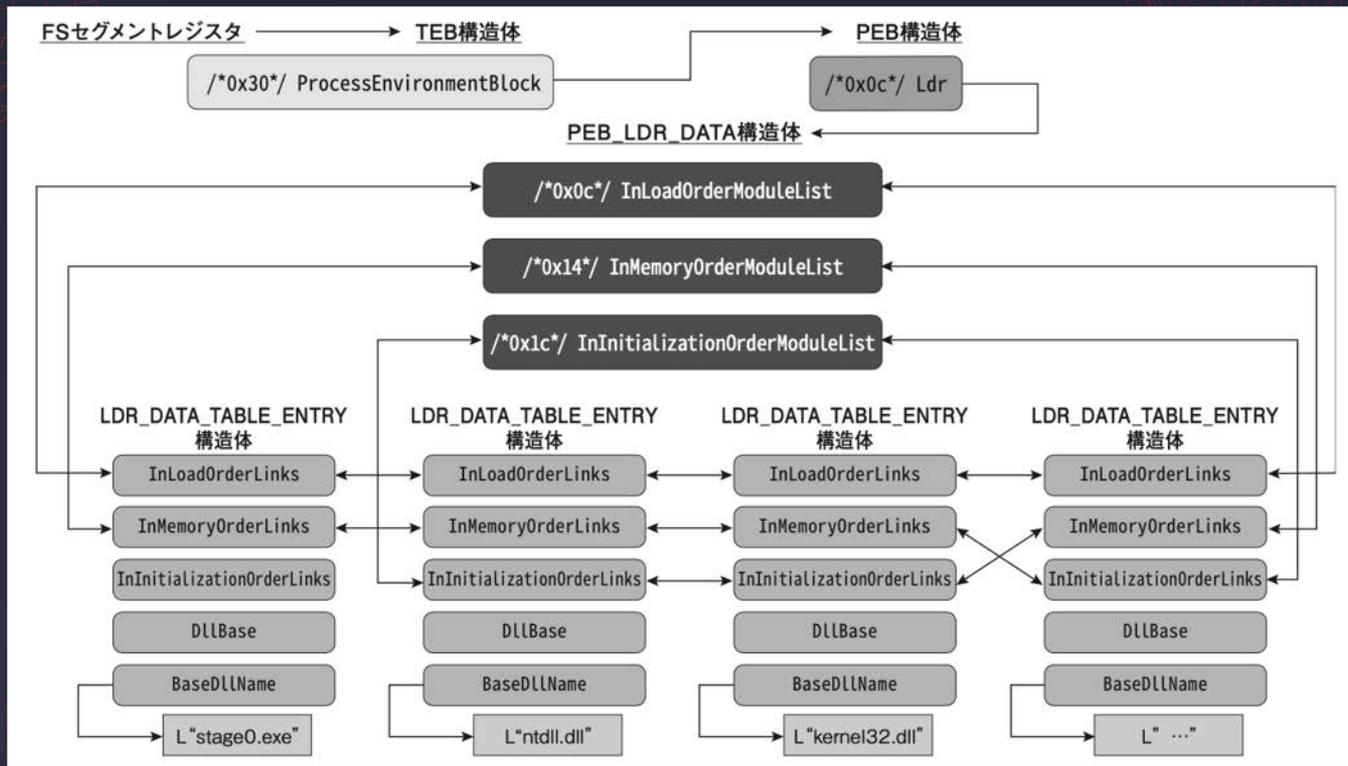
- 現在のプロセスに関する情報を保持する構造体
  - プロセスに読み込まれているDLLや、  
ヒープの状態、実行ファイルの情報など
- TEB構造体のオフセット0x30からアクセス可能

FSレジスタ経由でTEBにアクセスし、  
そこからPEBにアクセスしている

```
undefined4 FUN_10002b48(void)
{
    int in_FS_OFFSET;

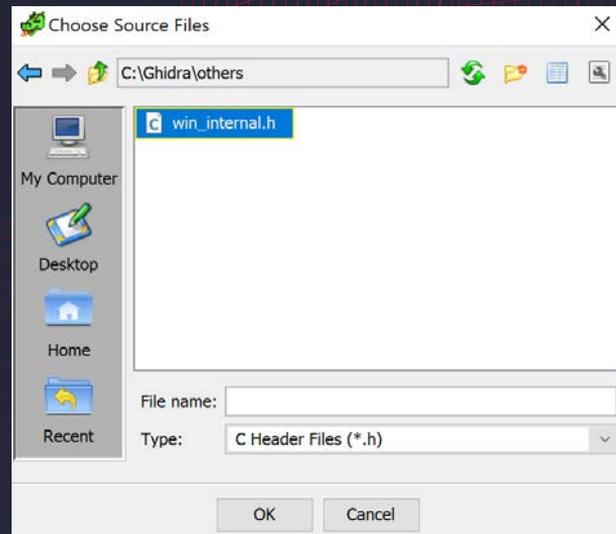
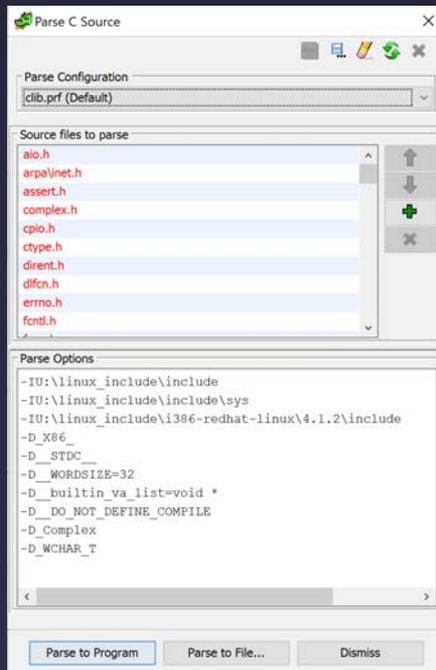
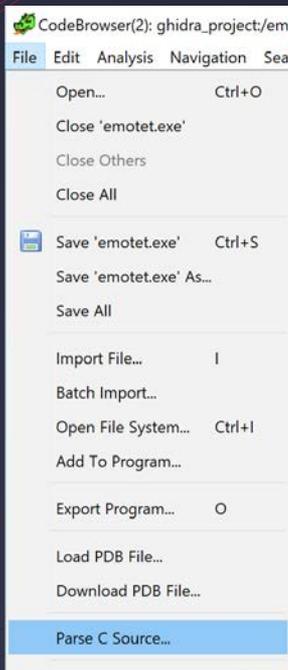
    return *(undefined4 *) (in_FS_OFFSET + 0x30);
}
```

# Thread Environment Block / Process Environment Block



# Apply Structure

- PEB構造体を事前定義したヘッダファイルをGhidraにインポート(すでに適用済み)
  - File > 「Parse C Source」 > win\_internal.hを選択 > 「Parse to Program」



# FUN\_10002b48

- PEBのポインタを取得する関数
  - in\_FS\_OFFSETをAuto Create Structure

BEFORE	AFTER (name)	AFTER (type)
FUN_10002b48	get_peg	PEB *
in_FS_OFFSET	teb	TEB *

```
Decompile: get_peg ...  
1  
2 PEB * get_peg(void)  
3  
4 {  
5     TEB *teb;  
6  
7     return teb->ProcessEnvironmentBlock;  
8 }
```

# FUN\_10004ae6

- 読み込まれているDLLのベースネームを列挙し、FUN\_100180eaに渡している
- FUN\_100180eaの戻り値を0x1fc325daとXOR
- その結果を引数の値と比較

BEFORE	AFTER (name)	AFTER (type)
iVar1	peb	PEB *
puVar3	current_module_entry	LDR_DATA_TABLE_ENTRY *
puVar4	next_module_entry	LDR_DATA_TABLE_ENTRY *

```

Decompile: FUN_10004ae6 - (emotet.dll)
1
2 DWORD __cdecl FUN_10004ae6(uint param_1)
3
4 {
5     PEB *peb;
6     uint uVar1;
7     LDR_DATA_TABLE_ENTRY *next_module_entry;
8     LDR_DATA_TABLE_ENTRY *current_module_entry;
9
10    peb = get_peb();
11    current_module_entry = (LDR_DATA_TABLE_ENTRY *) &peb->Ldr->InLoadOrderModuleList;
12    next_module_entry = (LDR_DATA_TABLE_ENTRY *) (current_module_entry->InLoadOrderLinks).Flink;
13    while( true ) {
14        if (next_module_entry == current_module_entry) {
15            return 0;
16        }
17        uVar1 = FUN_100180ea((ushort *) (next_module_entry->BaseDllName).Buffer, 2, 0x639b);
18        if ((uVar1 ^ 0x1fc325da) == param_1) break;
19        next_module_entry = (LDR_DATA_TABLE_ENTRY *) (next_module_entry->InLoadOrderLinks).Flink;
20    }
21    return next_module_entry->DllBase;
22 }

```

プロセスに読み込まれているDLLの列挙

DLLのベースネーム (ファイル名) を渡している

戻り値をXORして引数と比較

# FUN\_100180ea

1. この関数はハッシュ関数です、アルゴリズムを解析してください
2. 実際はFUN\_100180eaの戻り値を0x1fc325daとXORして比較していました。よって、次のように、DLLベースネームとXOR鍵の入力を受けて適切なハッシュ値を返す関数(calc\_hash)をGhidra Scriptで実装してください

Python - Interpreter

```
>>> hex(calc_hash('kernel32.dll', 0x1fc325da))  
'0x90bdc3a8L'
```

```
Decompile: FUN_100180ea - (emotet.dll)  
1  
2 int __fastcall FUN_100180ea(ushort *param_1, undefined param_2, undefined4 param_3)  
3  
4 {  
5     ushort uVar1;  
6     int iVar2;  
7     uint uVar3;  
8  
9     _();  
10    iVar2 = 0;  
11    uVar1 = *param_1;  
12    while (uVar1 != 0) {  
13        uVar3 = (uint)*param_1;  
14        if ((0x40 < uVar3) && (uVar3 < 0x5b)) {  
15            uVar3 = uVar3 + 0x20;  
16        }  
17        param_1 = param_1 + 1;  
18        iVar2 = uVar3 + iVar2 * 0x1003f;  
19        uVar1 = *param_1;  
20    }  
21    return iVar2;  
22 }
```

# FUN\_100180ea

- BaseDllName.Bufferの型はwchar\_t \* (=unsigned short \*)
- 1バイトずつ以下処理
  - 大文字ASCII (0x40 < x < 0x5b) の場合、小文字 (+0x20)に変更
  - 結果を状態(iVar2)に加算し、0x1003fを乗算
- 結果のint値を返す



ハッシュ関数であると判断できる  
(「calc\_hash\_case\_insensitive」に変更)

```
Decompile: calc_hash_case_insensitive - (emotet.dll)
1
2 int __fastcall calc_hash_case_insensitive(ushort *dll_basename,undefined __,undefined4 __)
3
4 {
5     uint char;
6     int value;
7     ushort wchar;
8
9     ::_();
10    value = 0;
11    wchar = *dll_basename;
12    while (wchar != 0) {
13        char = (uint)*dll_basename;
14        if ((0x40 < char) && (char < 0x5b)) {
15            char = char + 0x20;
16        }
17        dll_basename = dll_basename + 1;
18        value = char + value * 0x1003f;
19        wchar = *dll_basename;
20    }
21    return value;
22 }
```

# FUN\_100180ea

- DLLベースネームとXOR鍵を受け取り、ハッシュ値を返す関数

```
if ((uVar1 ^ 0x1fc325da) == param_1) break;
```

```
def calc_hash_lower(name, key):  
    name = name.lower()  
    value = 0  
    for c in name:  
        value = (ord(c) + value * 0x1003f) & 0xffffffff  
    return value ^ key
```

```
>>> hex(calc_hash_lower('Kernel32.dll', 0x1fc325da))  
'0x90bdc3a8L'
```

```
pcVar1 = (code *) FUN_10007d5b(0xf5, 0xb, 0x90bdc3a8, 0xb7c2e83f);
```

# FUN\_10004ae6

- `calc_hash_case_insensitive`関数の呼び出し元に戻る
- これまでの結果から、DLL名をハッシュ化した値を受け取り、ロード済みDLLの中から該当するDLLを探してベースアドレスを返す関数だとわかる
  - ベースアドレス: メモリにロードされているDLLのPEヘッダへのポインタ
- 関数名を「`find_lib`」に変えておく

BEFORE	AFTER (name)	AFTER (type)
param_1	hashed_dllname	N/A
uVar1	hashed	N/A
FUN_10004ae6	find_lib	N/A

```

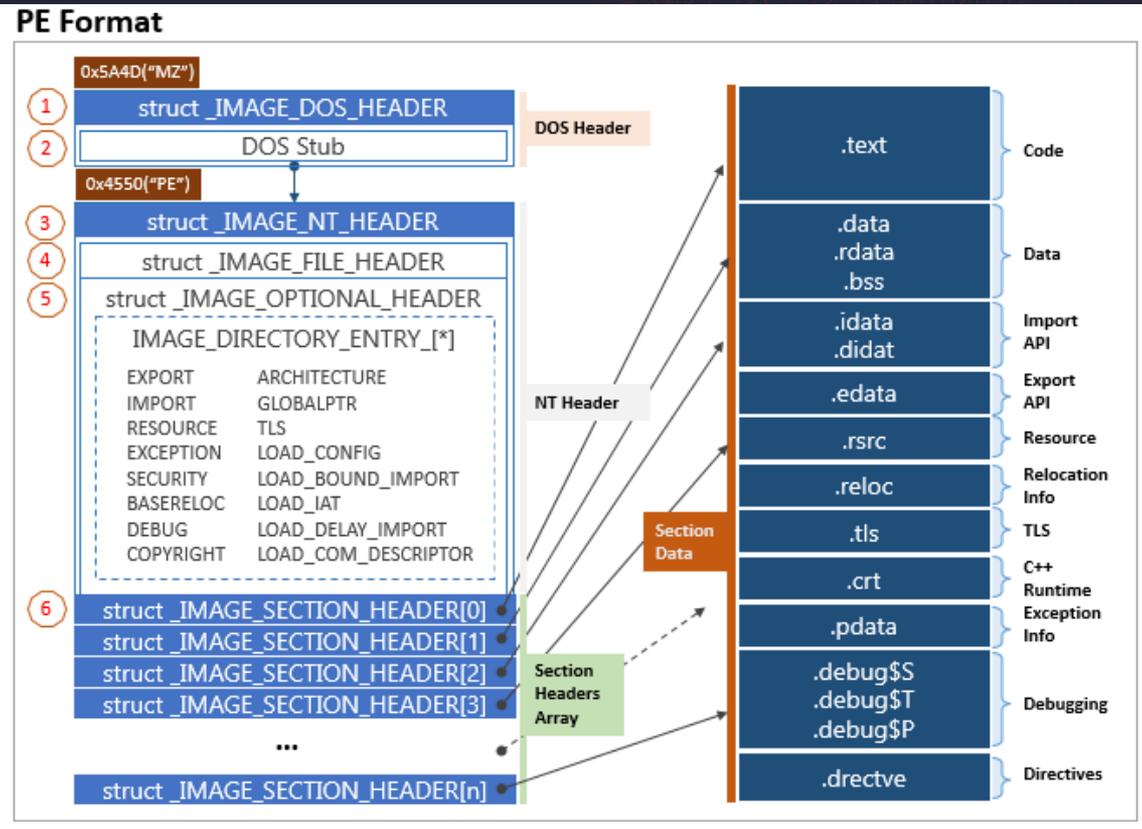
Decompile: find_lib - (emotet.dll)

1
2 DWORD __cdecl find_lib(uint hashed_dllname)
3
4 {
5     PEB *peb;
6     uint hashed;
7     LDR_DATA_TABLE_ENTRY *next_module_entry;
8     LDR_DATA_TABLE_ENTRY *current_module_entry;
9
10    peb = get_peb();
11    current_module_entry = (LDR_DATA_TABLE_ENTRY *)&peb->Ldr->InLoadOrderModuleList;
12    next_module_entry = (LDR_DATA_TABLE_ENTRY *)(current_module_entry->InLoadOrderLinks).Flink;
13    while( true ) {
14        if (next_module_entry == current_module_entry) {
15            return 0;
16        }
17        hashed = calc_hash_case_insensitive((ushort *) (next_module_entry->BaseDllName).Buffer, 2, 0x639b);
18        if ((hashed ^ 0x1fc325da) == hashed_dllname) break;
19        next_module_entry = (LDR_DATA_TABLE_ENTRY *) (next_module_entry->InLoadOrderLinks).Flink;
20    }
21    return next_module_entry->DllBase;
22 }
  
```

# PE Header Overview

Windows実行可能形式(PE)の  
ファイルフォーマット

- 実行ファイルに関する様々な情報を保持している
  - **Export Table**
    - エクスポート関数一覧
  - **Import Table**
    - 使用するAPI一覧



# FUN\_10012794



なにもわからない

```
Decompile: FUN_10012794 - (emotet.dll)
1
2 char * __fastcall FUN_10012794(undefined4 param_1,int param_2,undefined4 param_3,uint param_4)
3
4 {
5     int iVar1;
6     int iVar2;
7     int iVar3;
8     int iVar4;
9     uint uVar5;
10    uint uVar6;
11    char *pcVar7;
12    char *pcVar8;
13
14    _();
15    uVar6 = 0;
16    iVar1 = *(int *) (param_2 + 0x3c);
17    pcVar8 = (char *) (*(int *) (iVar1 + 0x78 + param_2) + param_2);
18    iVar2 = *(int *) (pcVar8 + 0x1c);
19    iVar3 = *(int *) (pcVar8 + 0x20);
20    iVar4 = *(int *) (pcVar8 + 0x24);
21    if (*(int *) (pcVar8 + 0x18) != 0) {
22        do {
23            uVar5 = FUN_1000df50((char *) (*(int *) (iVar3 + param_2 + uVar6 * 4) + param_2),0x92,0x2f3d,
24                0x30ee);
25            if ((uVar5 ^ 0x5a80eae) == param_4) {
26                pcVar7 = (char *) (*(int *) (iVar2 + param_2 +
27                    (uint)*(ushort *) (iVar4 + param_2 + uVar6 * 2) * 4) + param_2);
28                if (pcVar7 < pcVar8) {
29                    return pcVar7;
30                }
31                if (pcVar8 + *(int *) (iVar1 + 0x7c + param_2) <= pcVar7) {
32                    return pcVar7;
33                }
34                pcVar8 = (char *) FUN_10011d25(pcVar7);
35                return pcVar8;
36            }
37            uVar6 = uVar6 + 1;
38        } while (uVar6 < *(uint *) (pcVar8 + 0x18));
39    }
40    return (char *) 0x0;
41 }
```

# FUN\_10012794

- エクスポート関数の一覧を保持する  
エクスポートテーブルにアクセスしている

型を適用すると都度変数名が変わってしまうので、変更”前”の変数名で表記

BEFORE	AFTER (name)	AFTER (type)
param_2	base_address	N/A
param_4	hashed_api	N/A
iVar1	nt_headers	IMAGE_NT_HEADER32 *
pcVar8	export_table	IMAGE_EXPORT_DIRECTORY *
iVar2	functions_rva	N/A
iVar3	names_rva	N/A
iVar4	ordinals_rva	N/A
uVar5	value	N/A
uVar6	i	N/A
pcVar7	export_func	LPVOID

```

2 char * __fastcall FUN_10012794(undefined4 __,int base_address,undefined4 __,uint hashed_api)
3
4 {
5     uint value;
6     LPVOID *ppvVar1;
7     uint i;
8     LPVOID *export_func;
9     IMAGE_EXPORT_DIRECTORY *export_table;
10    IMAGE_NT_HEADERS32 *nt_header;
11    DWORD functions_rva;
12    DWORD names_rva;
13    DWORD ordinals_rva;
14
15    ::();
16    i = 0;
17
18    /* e_lfanew */
19    nt_header = *(IMAGE_NT_HEADERS32 **) (base_address + 0x3c);
20    export_table = (IMAGE_EXPORT_DIRECTORY *)
21        (*(int *) ((int) &(nt_header->OptionalHeader).DataDirectory[0].VirtualAddress +
22            base_address) + base_address);
23    functions_rva = export_table->AddressOfFunctions;
24    names_rva = export_table->AddressOfNames;
25    ordinals_rva = export_table->AddressOfNameOrdinals;
26    if (export_table->NumberOfNames != 0) {
27        do {
28            value = FUN_1000df50((char *) (*(int *) (names_rva + base_address + i * 4) + base_address),
29                0x92,0x2f3d,0x30ee);
30            if ((value ^ 0x5a90eae) == hashed_api) {
31                export_func = (LPVOID *)
32                    (*(int *) (functions_rva + base_address +
33                        (uint) * (ushort *) (ordinals_rva +
34                            base_address)));
35                if (export_func < export_table) {
36                    return (char *) export_func;
37                }
38                if ((LPVOID *)
39                    ((int) &export_table->Characteristics +
40                    *(int *) ((int) &(nt_header->OptionalHeader).DataDirectory[0].Size + base_address)) <=
41                    export_func) {
42                    return (char *) export_func;
43                }
44                ppvVar1 = (LPVOID *) FUN_10011d25((char *) export_func);
45                return (char *) ppvVar1;
46            }
47            i = i + 1;
48        } while (i < export_table->NumberOfNames);
49    }
50    return (char *) (LPVOID *) 0x0;
51 }

```

DLLのベースアドレス、つまり  
IMAGE\_DOS\_HEADERへの  
ポインタが引数に渡されている

OptionalHeader.DataDirectoryは  
さまざまなデータを格納する連想配列で  
インデックス0にはExport Tableへの  
オフセットが格納されている

↓

- エクスポート関数のRVAの配列
- エクスポート関数名のRVAの配列
- エクスポート関数の序数のRVAの配列

↓ エクスポート関数名を列挙

↓ エクスポート関数のアドレスを取得

↔ エクスポート関数のアドレスが  
フォワードされたものか確認

# FUN\_10012794

- 列挙したエクスポート関数をFUN\_1000df50に渡す
- 戻り値を0x5a80eaeとXOR
- XORした結果を第四引数と比較
- 合致した場合、対象APIのアドレスを返す
- Export ForwardされたAPIは別途APIを解決

```
25 if (export_table->NumberOfNames != 0) {
26     do {
27         value = FUN_1000df50((char *) (* (int *) (names_rva + base_address + i * 4) + base_address),
28                             0x92, 0x2f3d, 0x30ee);
29         if ((value ^ 0x5a80eae) == hashed_api) {
30             export_func = (LPVOID *)
31                 (* (int *) (functions_rva + base_address +
32                             (uint) * (ushort *) (ordinals_rva + base_address + i * 2) * 4) +
33                 base_address);
34             if (export_func < export_table) {
35                 return (char *) export_func;
36             }
37             if ((LPVOID *)
38                 ((int) & export_table->Characteristics +
39                  * (int *) ((int) & (nt_header->OptionalHeader).DataDirectory[0].Size + base_address)) <=
40                 export_func) {
41                 return (char *) export_func;
42             }
43             ppvVar1 = (LPVOID *) FUN_10011d25((char *) export_func);
44             return (char *) ppvVar1;
45         }
46         i = i + 1;
47     } while (i < export_table->NumberOfNames);
```

※export forwardとは

<https://docs.microsoft.com/en-us/windows/win32/debug/pe-format#export-address-table>

<https://devblogs.microsoft.com/oldnewthing/20060719-24/?p=30473>

# FUN\_1000df50

- API名が渡される関数
- ほぼ  
calc\_hash\_case\_sensitiveと同じ（小文字  
変換処理がない）

```
Decompile: FUN_1000df50 - (emotet.dll)
1
2 int __fastcall FUN_1000df50(char *param_1,undefined param_2,undefined4 param_3,undefined4 param_4)
3
4 {
5     char cVar1;
6     int iVar2;
7
8     _();
9     iVar2 = 0;
10    cVar1 = *param_1;
11    while (cVar1 != '\0') {
12        iVar2 = (int)*param_1 + iVar2 * 0x1003f;
13        param_1 = param_1 + 1;
14        cVar1 = *param_1;
15    }
16    return iVar2;
17 }
```



```
Decompile: calc_hash_case_sensitive - (emotet.dll)
1
2 int __fastcall
3 calc_hash_case_sensitive(char *api_name,undefined param_2,undefined4 param_3,undefined4 param_4)
4
5 {
6     char char;
7     int value;
8
9     _();
10    value = 0;
11    char = *api_name;
12    while (char != '\0') {
13        value = (int)*api_name + value * 0x1003f;
14        api_name = api_name + 1;
15        char = *api_name;
16    }
17    return value;
18 }
```

# Calculate Hash

FUN\_10012794関数内にハードコードされたXOR鍵

## ■ Pythonで実装

```

71 def calc_hash(name, key):
72     value = 0
73     for c in name:
74         value = (ord(c) + value * 0x1003f) & 0xffffffff
75     return value ^ key

```

```
if ((value ^ 0x5a80eae) == hashed_api)
```

```

Python - Interpreter
>>> hex(calc_hash('ExitProcess', 0x5a80eae))
'0xb7c2e83fL'

```

呼び出し元の例 (@0x10017e02)

合致している 🎉

```
pcVar1 = (code *)FUN_10007d5b(0xf5, 0xb, 0x90bdc3a8, 0xb7c2e83f);
```

# (Advanced) Calculate Hash by Emulator

- アルゴリズムの実装がわからなくても、Emulatorで実行してしまうことも可能

```
1001285a e8 f1 b6 ...   CALL      calc_hash_case_sensitive
1001285f 35 ae 0e ...   XOR       value,0x5a80eae
10012864 59           POP      param_1
10012865 59           POP      param_1
10012866 3b 44 24 34  CMP     value,dword ptr [ESP + hashed_api]
```

1. (エミュレータ内の) ECXに文字列を格納 (=引数に渡す)
2. 0x1001285aから0x10012864まで実行
3. EAXに格納されている値 (=ハッシュ化済みの文字列) を取得

# emulator.py

- Ghidraが提供している EmulatorHelperクラスを使用し、エミュレータを実装

```
1 from ghidra.app.emulator import EmulatorHelper
2
3 def init_emulator():
4     # initialize emulator
5     emu = EmulatorHelper(currentProgram)
6
7     # initialize stack
8     registry_size = emu.getStackPointerRegister().getBitLength()
9     stack_address = toAddr((((1 << (registry_size - 1)) - 1) ^ ((1 << (registry_size//2))-1)))
10    emu.writeRegister(emu.getStackPointerRegister(), stack_address.getOffset())
11
12    return emu
13
14
15 def emulate_calc_hash(entry_addr, param):
16    emu = init_emulator()
17
18    end_address = entry_addr.add(10)
19    emu.setBreakpoint(end_address)
20
21    # write parameter string into ECX
22    string_address = toAddr(0xa00000)
23    emu.writeMemory(string_address, param)
24    emu.writeRegister('ECX', string_address.getOffset())
25
26    # set address of entrypoint to execute at EIP
27    emu.writeRegister(emu.getPCRegister(), entry_addr.getOffset())
28
29    # run through until breakpoint
30    while monitor.isCancelled() is False:
31        current_address = emu.getExecutionAddress()
32        if current_address == end_address:
33            break
34        emu.step(monitor)
35
36    # get hashed value in EAX
37    result = emu.readRegister('EAX')
38    return result
```

# (Advanced) Calculate Hash by Emulator

GhidraのEmulatorでハッシュ関数をエミュレート

```
Python - Interpreter
>>> from emulator import emulate_calc_hash
>>> hashed_dll = emulate_calc_hash(toAddr(0x10004b98), 'kernel32.dll'.encode('utf-16le'))
>>> hex(hashed_dll)
'0x90bdc3a8L'
>>> hashed_api = emulate_calc_hash(toAddr(0x1001285a), 'ExitProcess')
>>> hex(hashed_api)
'0xb7c2e83fL'
```

実際に渡されているハッシュ値

```
pcVar1 = (code *) FUN_10007d5b(0xf5, 0xb, 0x90bdc3a8, 0xb7c2e83f);
```

# FUN\_10007d5b

引数のハッシュ値をもとに  
APIを動的解決し、グローバル  
変数にキャッシュ

## 変数

BEFORE	AFTER (name)
param_1	index
param_2	hashed_dll_name
param_3	hashed_api_name
g_array	g_api_table
pcVar1	api

## 関数

BEFORE	AFTER (name)
FUN_10007d5b	resolve_api
FUN_10004ae6	find_lib
FUN_10012794	find_api

```

Decompile: resolve_api - (emotet.dll)
1
2 int __cdecl resolve_api(int index,undefined4 __,uint hashed_dll_name,uint hashed_api_name)
3
4 {
5     int base_address;
6     char *api;
7
8     if (g_api_table[index] == 0) {
9         base_address = find_lib(hashed_dll_name);
10        api = find_api(0x7ef7,base_address,0x7a71,hashed_api_name);
11        g_api_table[index] = (int)api;
12    }
13    return g_api_table[index];
14 }

```

# Anti-API Hashing

---

プログラム内のハッシュ値一覧を取得し、API名を逆解決してコメントをつけたい

1. Hash DBの作成
  1. API Hashに使用するXOR鍵を特定
  2. System32配下の主要なDLLのエクスポート関数名をハッシュ化
2. ハッシュ値とDBを突合
  1. スタックに積まれているスカラー値を全て取得
  2. Hash DBに登録済みのハッシュ値と突合
3. コメントをつける
  1. 呼び出されるAPI名をコード上にコメント

# Create Hash DB

## make\_hash\_db.py

- %windir%\System32\\*.dll  
(今回は簡略化のため一部DLLのみ)に定義されているエクスポート関数を列挙
- 検体内にハードコードされていたXOR鍵で関数名をエンコードし、↓形式のjsonで格納
  - <HASH>: <ORIG\_NAME>

```

21 def calc_hash(name, key):
22     value = 0
23     for c in name:
24         value = (ord(c) + value * 0x1003f) & 0xffffffff
25     return value ^ key
26
27 def get_export_api(dllpath):
28     pe = pefile.PE(dllpath)
29     if ((not hasattr(pe, "DIRECTORY_ENTRY_EXPORT")) or (pe.DIRECTORY_ENTRY_EXPORT is None)):
30         raise RuntimeError(f'{dllpath} doesn\'t have export table')
31
32     for sym in pe.DIRECTORY_ENTRY_EXPORT.symbols:
33         if sym.name is not None:
34             yield sym.name.decode('utf-8')
35
36 def main():
37     import argparse
38
39     p = argparse.ArgumentParser()
40     p.add_argument('--lib-key', dest='lib_key', default=LIB_XOR_KEY, type=lambda x: int(x, 0), help='XOR key for lib name hashing')
41     p.add_argument('--api-key', dest='api_key', default=API_XOR_KEY, type=lambda x: int(x, 0), help='XOR key for API hashing')
42
43     args = p.parse_args()
44
45     # partially apply calc_hash function with XOR key
46     emotet_xor_for_lib = partial(calc_hash, key=args.lib_key)
47     emotet_xor_for_api = partial(calc_hash, key=args.api_key)
48
49     results = defaultdict(dict)
50
51     # enumerate all .dll files in %windir%\system32
52     target_path = os.path.join(os.environ.get('windir'), 'system32')
53     for dll_filepath in glob.glob(os.path.join(target_path, '*.dll')):
54         dllname = os.path.basename(dll_filepath)
55
56         # filter uninteresting dlls for easy use
57         if dllname.lower() not in INTERESTING_DLLS:
58             continue
59
60         # calc hash of dll name
61         hashed_dllname = emotet_xor_for_lib(dllname.lower())
62         results[hex(hashed_dllname)] = dllname
63
64         # calc hash of each API name in DLL
65         for api in get_export_api(dll_filepath):
66             hashed_api = emotet_xor_for_api(api)
67             results[hex(hashed_api)] = api
68
69     # dump to json file
70     with open('db.json', 'w') as f:
71         json.dump(results, f, indent=4)

```

# Create Hash DB

make\_hash\_db.py

- 出力されたdb.json

```
1  {
2      "0x2de3bdc6": "advapi32.dll",
3      "0xfd8a9a46": "A_SHAFinal",
4      "0x862245b0": "A_SHAInit",
5      "0xf996c519": "A_SHAUpdate",
6      "0x4927aac2": "AbortSystemShutdownA",
7      "0x4927aa2c": "AbortSystemShutdownW",
8      "0xa0ea950a": "AccessCheck",
9      "0xf2ea6e16": "AccessCheckAndAuditAlarmA",
10     "0xf2ea6e60": "AccessCheckAndAuditAlarmW",
11     "0x233ec35b": "AccessCheckByType",
12     "0x24dd2927": "AccessCheckByTypeAndAuditAlarmA",
13     "0x24dd2931": "AccessCheckByTypeAndAuditAlarmW",
14     "0x7e40889e": "AccessCheckByTypeResultList",
15     "0xa8efafea": "AccessCheckByTypeResultListAndAuditAlarmA",
16     "0xb86cc6eb": "AccessCheckByTypeResultListAndAuditAlarmByHandleA",
17     "0xb86cc6f5": "AccessCheckByTypeResultListAndAuditAlarmByHandleW",
18     "0xa8efaff4": "AccessCheckByTypeResultListAndAuditAlarmW",
19     "0x4fed1fce": "AddAccessAllowedAce",
20     "0x3ff7c7d": "AddAccessAllowedAceEx",
21     "0x8e1182f": "AddAccessAllowedObjectAce",
22     "0x5f75502d": "AddAccessDeniedAce",
23     "0x437d50d8": "AddAccessDeniedAceEx",
24     "0xeed4718a": "AddAccessDeniedObjectAce",
25     "0x73006e2c": "AddAce",
26     "0xe5dbf5cb": "AddAuditAccessAce",
27     "0xa78fa4f6": "AddAuditAccessAceEx",
28     "0xee7a7a8": "AddAuditAccessObjectAce",
```

# Lookup Hashes

- DLL名とAPI名のハッシュ値は`resolve_api`関数の引数に渡されている
- 値はスタック経由で渡されている
- 雑だが、PUSH命令の引数に渡されているスカラー値が取得できればよさそう

## Decompile View at FUN\_10017d5a

```
pcVar1 = (code *)resolve_api(0xf5,0xb,0x90bdc3a8,0xb7c2e83f);
(*pcVar1)(0,uVar2,uVar3,uVar4,uVar5,uVar6,uVar7,uVar8);
return;
```

```
undefined4 __cdecl resolve_api(int index, undefined4 _, uint hashed_dll_name, uint hashed_api_name)
undefined4 EAX:4 <RETURN>
int Stack[0x4]:4 index
undefined4 Stack[0x8]:4 _
uint Stack[0xc]:4 hashed_dll_name
uint Stack[0x10]:4 hashed_api_name
```

## Listing View at FUN\_10017d5a

10017df2	68 3f e8 ...	PUSH	0xb7c2e83f
10017df7	68 a8 c3 ...	PUSH	0x90bdc3a8
10017dfc	51	PUSH	ECX
10017dfd	68 f5 00 ...	PUSH	0xf5
10017e02	e8 54 ff ...	CALL	resolve_api

# Lookup Hashes

- プログラム内の全命令を列挙
- PUSH命令の引数に渡されたスカラー値を取得
- 値をDB内のハッシュ値と比較
- 合致したらコメント・ブックマークをつける

```
9 def add_bookmark_comment(addr, comment):
10     cu = currentProgram.getListing().getCodeUnitAt(addr)
11     createBookmark(addr, "hashed_name", comment)
12     cu.setComment(CodeUnit.EOL_COMMENT, comment)

39 def main():
40     db_path = askFile("DB for hashes", "import").getPath()
41
42     # for Headless usage, you can get argument via command line
43     # db_path = getScriptArgs()[0]
44
45     db = HashDB(db_path)
46
47     # get all instructions in program
48     instructions = currentProgram.getListing().getInstructions(True)
49
50     # process each instruction
51     for inst in instructions:
52         value = get_scalar_argument(inst)
53         if value is not None:
54             try:
55                 # then lookup DB
56                 orig_name = db.lookup(str(value))
57
58                 # if it exists, add comment and add bookmark
59                 if orig_name:
60                     print('[*] {} at {}: {}'.format(str(value), inst.getAddress(), orig_name))
61                     add_bookmark_comment(inst.getAddress(), orig_name)
62             except Exception as e:
63                 print(e)
```

# Lookup Hashes

- 与えられた命令がスカラー値をPUSHするものかチェックする関数を実装し、`search_hash_not_impl.py`を完成させてください

`search_hash_not_impl.py`

```
28 def get_scalar_argument(inst):
29     ''' **** IMPLEMENT HERE ****
30
31     if given instruction is like 'PUSH <SCALAR>',
32     retrun scalar value.
33     Hint:
34     | - check API document of Instruction interface
35     '''
36     raise NotImplementedError('not implemented')
```

# Lookup Hashes

- 与えられた命令がスカラー値をPUSHするものかチェックする関数を実装し、`search_hash_not_impl.py`を完成させてください
- ヒント:
  - `inst.getMnemonicString()`
  - `Inst.getOpObjects()`

## search\_hash\_not\_impl.py

```
28 def get_scalar_argument(inst):
29     ''' **** IMPLEMENT HERE ****
30
31     if given instruction is like 'PUSH <SCALAR>',
32     retrun scalar value.
33     Hint:
34     | - check API document of Instruction interface
35     ...
36     raise NotImplementedError('not implemented')
```

# Lookup Hashes

- 与えられた命令がスカラー値をPUSHするものかチェックする関数を実装し、`search_hash_not_impl.py`を完成させてください

実装した`search_hash.py`

```
28 def get_scalar_argument(inst):
29     # get mnemonic in one instruction
30     mnemonic = inst.getMnemonicString()
31
32     if mnemonic == 'PUSH':
33         # get operand in index 1
34         value = inst.getOpObjects(0)[0]
35         if isinstance(value, Scalar) and value.bitLength() == 32:
36             return value
```

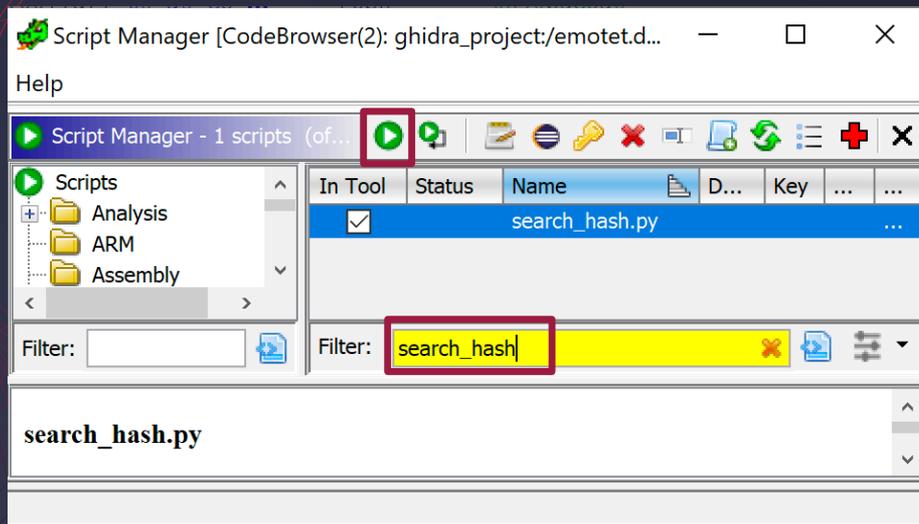
# Run Ghidra Script

## ■ Script Managerで実行

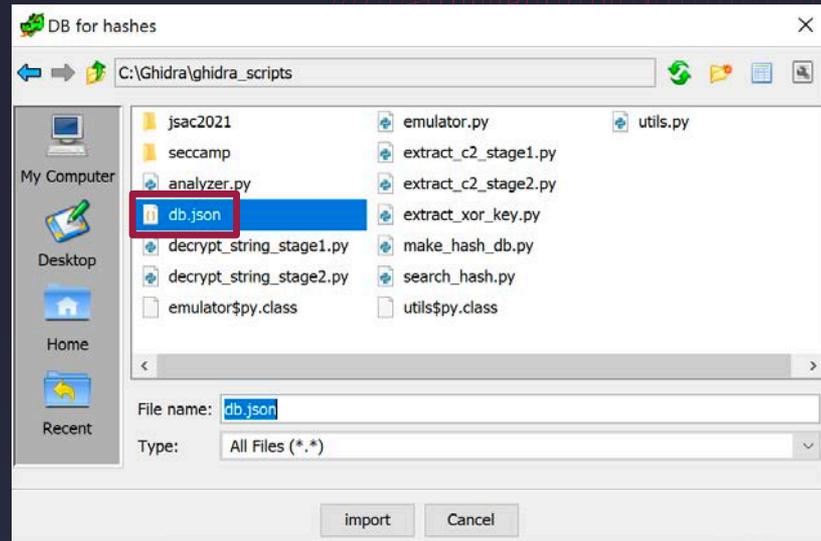
### 1. Script Manager起動



### 2. スクリプト名を検索して実行



### 3. 参照するDBを指定 (make\_hash\_db.pyで作成したjson)



# Run Ghidra Script

コンソールに実行結果が表示されている

```

Console - Scripting
0xb is not found in DB
[*] 0xb7c2e83f at 10017df2: ExitProcess
[*] 0x90bdc3a8 at 10017df7: kernel32.dll
0xf5 is not found in DB
0x0 is not found in DB

```

API名・DLL名のコメントがついている

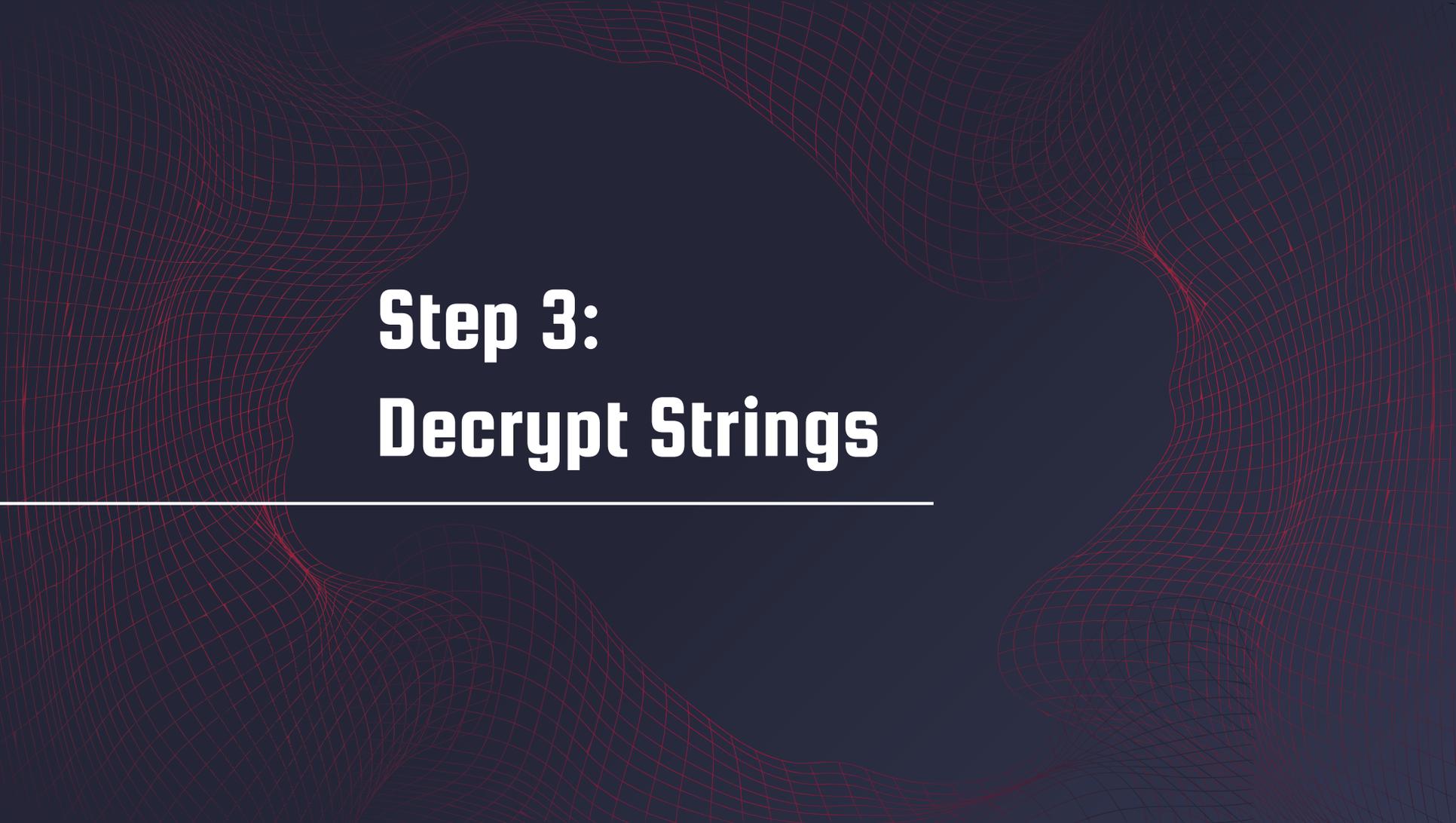
10017df2	68 3f e8 ...	PUSH	0xb7c2e83f	ExitProcess	21	/* ExitProcess */
10017df7	68 a8 c3 ...	PUSH	0x90bdc3a8	kernel32.dll	22	/* kernel32.dll */
10017dfc	51	PUSH	ECX		23	pcVar1 = (code *)resolve_api(0xf5,0xb,0x90bdc3a8,0xb7c2e83f);
10017dfd	68 f5 00 ...	PUSH	0xf5		24	(*pcVar1)(0,uVar2,uVar3,uVar4,uVar5,uVar6,uVar7,uVar8);
10017e02	e8 54 ff ...	CALL	resolve_api	undefined4 res	25	return;

※コメントが表示されない場合、  
Edit > Tool Options > Decompiler > Display の  
「Display EOL Comments」が有効化されていない可能性

# Other API Hash Search Script

---

- EMOTETは、ハッシュ関数に使用されるXOR鍵が検体ごとに異なるため、検体ごとにDBを作成する必要があるが、既知のアルゴリズムが使用されている場合は、既成のDBを使用してルックアップすれば、ハッシュ関数の詳細をしらなくてもAPI Hashを解決できる場合がある
  - `shellcode_hash_search.py` (Ghidra)
    - [https://github.com/AllsafeCyberSecurity/ghidra\\_scripts/blob/master/shellcode\\_hash\\_search.py](https://github.com/AllsafeCyberSecurity/ghidra_scripts/blob/master/shellcode_hash_search.py)
  - `shellcode_hashes_search_plugin.py` (IDA)
    - [https://github.com/fireeye/flare-ida/tree/master/shellcode\\_hashes](https://github.com/fireeye/flare-ida/tree/master/shellcode_hashes)



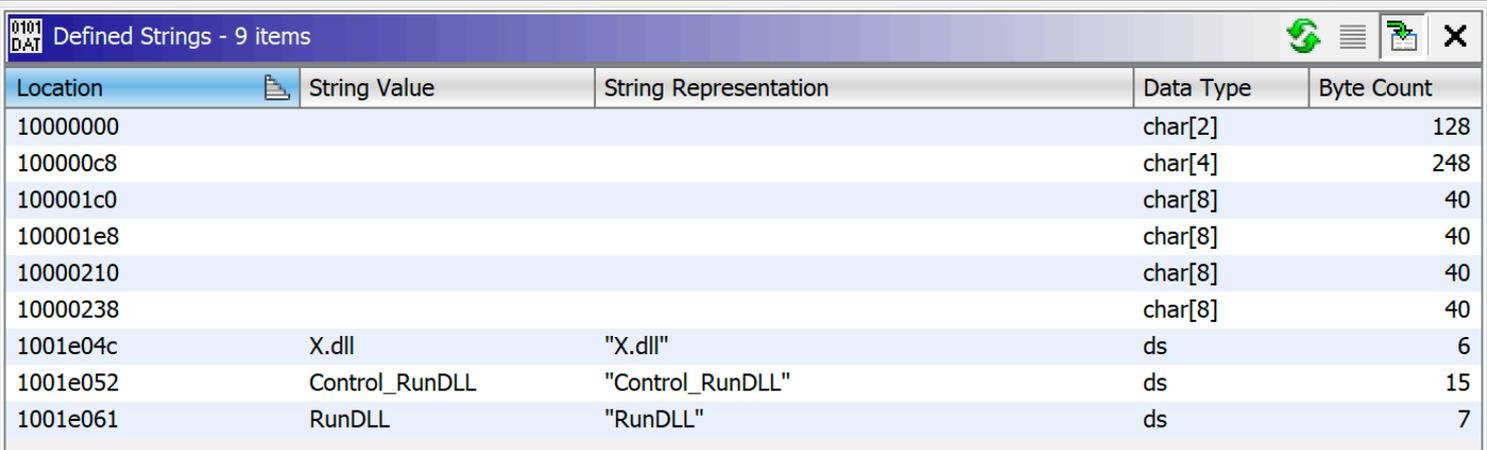
# **Step 3:** **Decrypt Strings**

---

# Defined Strings

- アンパック後のEMOTETには、不正活動に関連する可読文字列が存在しない
- 文字列も暗号化・難読化されている可能性がある

Window > Defined Strings から  
プログラム内の可読文字列を表示できる

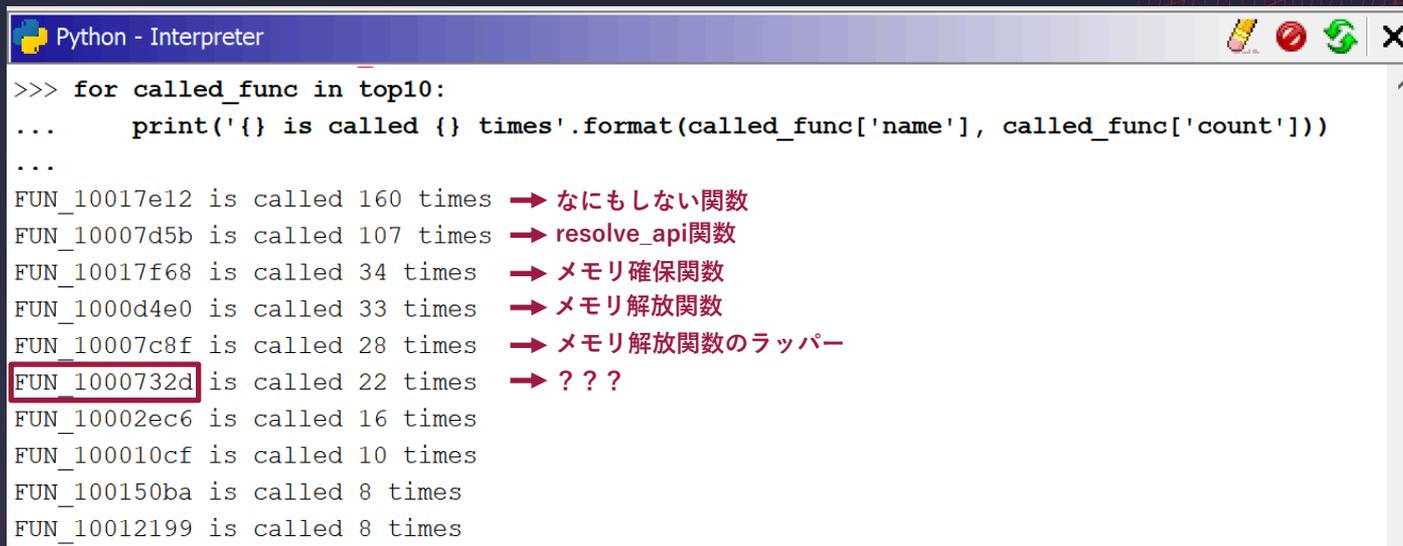


The screenshot shows a window titled "Defined Strings - 9 items". The window contains a table with the following columns: Location, String Value, String Representation, Data Type, and Byte Count. The table lists several memory locations, most of which are empty, and three entries with specific string values: "X.dll", "Control\_RunDLL", and "RunDLL".

Location	String Value	String Representation	Data Type	Byte Count
10000000			char[2]	128
100000c8			char[4]	248
100001c0			char[8]	40
100001e8			char[8]	40
10000210			char[8]	40
10000238			char[8]	40
1001e04c	X.dll	"X.dll"	ds	6
1001e052	Control_RunDLL	"Control_RunDLL"	ds	15
1001e061	RunDLL	"RunDLL"	ds	7

# Where is Decryption Function?

- 文字列が暗号化されている場合、実行時に復号関数が呼び出されているはず
- 復号関数は何度も呼び出されているはず



```
Python - Interpreter
>>> for called_func in top10:
...     print('{} is called {} times'.format(called_func['name'], called_func['count']))
...
FUN_10017e12 is called 160 times → なんもしない関数
FUN_10007d5b is called 107 times → resolve_api関数
FUN_10017f68 is called 34 times → メモリ確保関数
FUN_1000d4e0 is called 33 times → メモリ解放関数
FUN_10007c8f is called 28 times → メモリ解放関数のラッパー
FUN_1000732d is called 22 times → ???
FUN_10002ec6 is called 16 times
FUN_100010cf is called 10 times
FUN_100150ba is called 8 times
FUN_10012199 is called 8 times
```

# FUN\_1000732d

## 比較的複雑な関数

### 1. ロジックを解析してください

- ✓ 17~24、35~50行に集中する  
といいかも

### 2. 復号スクリプト

decrypt\_string\_stage1\_not\_impl.pyのdecrypt\_string関数を実装してください

```
Decompile: FUN_1000732d - (emotet.dll)
2  ushort * __fastcall FUN_1000732d(undefined4 param_1, uint *param_2, undefined4 param_3)
3
4  {
5      uint *puVar1;
6      uint uVar2;
7      ushort uVar3;
8      uint uVar4;
9      ushort *puVar5;
10     uint uVar6;
11     ushort *puVar7;
12     uint uVar8;
13     uint uVar9;
14     uint *puVar10;
15
16     _();
17     uVar2 = *param_2;
18     puVar10 = param_2 + 2;
19     uVar4 = param_2[1] ^ uVar2;
20     uVar8 = uVar4 + 1;
21     uVar9 = uVar8;
22     if ((uVar8 & 3) != 0) {
23         uVar9 = (uVar8 & 0xffffffff) + 4;
24     }
25     puVar5 = (ushort *)FUN_10017f68(uVar8 & 0xffffffff03, (void *) (uVar9 * 2));
26     if (puVar5 != (ushort *)0x0) {
27         uVar8 = 0;
28         puVar1 = (uint *) ((int)puVar10 + (uVar9 & 0xffffffff));
29         uVar9 = (uint) ((int)puVar1 + (3 - (int)puVar10) >> 2);
30         if (puVar1 < puVar10) {
31             uVar9 = 0;
32         }
33         puVar7 = puVar5;
34         if (uVar9 != 0) {
35             do {
36                 uVar6 = *puVar10;
37                 puVar10 = puVar10 + 1;
38                 uVar6 = uVar6 ^ uVar2;
39                 *puVar7 = (ushort)uVar6 & 0xff;
40                 puVar7[1] = (ushort)(uVar6 >> 8) & 0xff;
41                 uVar3 = (ushort)(uVar6 >> 0x10);
42                 uVar8 = uVar8 + 1;
43                 puVar7[2] = uVar3 & 0xff;
44                 puVar7[3] = uVar3 >> 8;
45                 puVar7 = puVar7 + 4;
46             } while (uVar8 < uVar9);
47         }
48         puVar5[uVar4] = 0;
49     }
50     return puVar5;
51 }
```

```
13 def decrypt_string(enc_data_addr):
14     """ **** IMPLEMENT HERE ****
15
16     decrypt string at given address and return it as string
17     Hint:
18     1. use 'getInt' to get 4 bytes in little endian format
19     2. use 'getBytes' to get specific size of bytes
20     3. use 'Address#add' method to add/sub address object
21         >>> addr = toAddr(0x401000)
22         >>> new_addr = addr.add(4)
23     4. use 'xor_with_multi_bytes' to xor bytes with DWORD
24     5. in jython, you can convert array to string by 'toString' method
25         >>> getBytes(0x408000)
26         array('b', [116, 101, 115, 116])
27         >>> getBytes(0x408000).toString()
28         'test'
29     """
30     raise NotImplementedError('not implemented')
```

# FUN\_1000732d

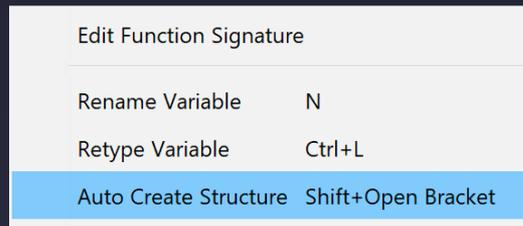
## 見やすくするヒント ①

- オフセットでアクセスされている場合、構造体を使用している可能性がある
- Ghidraは独自構造体を自動で識別できない
- Auto Create Structureで構造体を自動定義し、指定した変数に適用できる

```
17  uVar2 = *param_2;  
18  puVar10 = param_2 + 2;  
19  uVar4 = param_2[1] ^ uVar2;
```



param\_2を右クリック > Auto Create Structure



変更後

```
17  uVar2 = param_2->field_0x0;  
18  puVar10 = &param_2->field_0x8;  
19  uVar4 = param_2->field_0x4 ^ uVar2;
```

# FUN\_1000732d

## 見やすくするヒント ②

- 同じレジスタが異なる用途で使いまわされている場合、変数名が付けづらい問題
- Split Out As New Variable で変数を分割すると見やすくなる

```

20 uVar8 = uVar4 + 1;
21 uVar9 = uVar8;
22 if ((uVar8 & 3) != 0) {
23     uVar9 = (uVar8 & 0xffffffff) + 4;
24 }
25 puVar5 = (ushort *)FUN_10017f68(uVar8 & 0xfffff03, (void *) (uVar9 * 2));
26 if (puVar5 != (ushort *)0x0) {
27     uVar9 = 0;
28     puVar1 = (uint *) ((int)puVar10 + (uVar9 & 0xffffffff));
29     uVar9 = (uint) ((int)puVar1 + (3 - (int)puVar10) >> 2;
30     if (puVar1 < puVar10) {
31         uVar9 = 0;
32     }
33     puVar7 = puVar5;
34     if (uVar9 != 0) {
35         do {
36             uVar6 = *puVar10;
37             puVar10 = puVar10 + 1;
38             uVar6 = uVar6 ^ uVar2;
39             *puVar7 = (ushort)uVar6 & 0xff;
40             puVar7[1] = (ushort) (uVar6 >> 8) & 0xff;
41             uVar3 = (ushort) (uVar6 >> 0x10);
42             uVar8 = uVar8 + 1;

```

uVar8は途中で0で初期化され、別の用途で再利用されている

uVar8を右クリック > Split Out As New Variable

```

21 uVar10 = uVar4 + 1;
22 uVar10 = uVar8;
23 if ((uVar8 & 3) != 0) {
24     uVar10 = (uVar8 & 0xffffffff) + 4;
25 }
26 puVar5 = (ushort *)FUN_10017f68(uVar8 & 0xfffff03, (void *) (uVar10 * 2));
27 if (puVar5 != (ushort *)0x0) {
28     uVar9 = 0;
29     puVar1 = (uint *) ((int)puVar11 + (uVar10 & 0xffffffff));
30     uVar10 = (uint) ((int)puVar1 + (3 - (int)puVar11) >> 2;
31     if (puVar1 < puVar11) {
32         uVar10 = 0;
33     }
34     puVar7 = puVar5;
35     if (uVar10 != 0) {
36         do {
37             uVar6 = *puVar11;
38             puVar11 = puVar11 + 1;
39             uVar6 = uVar6 ^ uVar2;
40             *puVar7 = (ushort)uVar6 & 0xff;
41             puVar7[1] = (ushort) (uVar6 >> 8) & 0xff;
42             uVar3 = (ushort) (uVar6 >> 0x10);
43             uVar9 = uVar9 + 1;

```

後者がuVar9として再定義された

Edit Function Signature

Rename Variable N

Retype Variable Ctrl+L

Split Out As New Variable

# FUN\_1000732d

- 第2引数に、「暗号化された文字列に関する情報を保持する構造体」へのポインタが渡されている
  - オフセット+0: XOR鍵
  - オフセット+4: XOR鍵でエンコードされた、復号後文字列のサイズ
  - オフセット+8: 暗号データ
- 4バイトで割り切れるサイズに調整
- 復号文字列を格納するために、②のサイズ\*2バイト分のバッファ確保
- enc\_dataがemptyでないかチェック
- 復号処理
  - 4バイトを1チャンクとして処理
  - 4バイト分ポインタを進める
  - XOR鍵でデコード
  - 下位バイトから順にバッファにコピー
  - 4バイト分ポインタを進める
- 復号後のデータ末尾にNULLを追記し返す

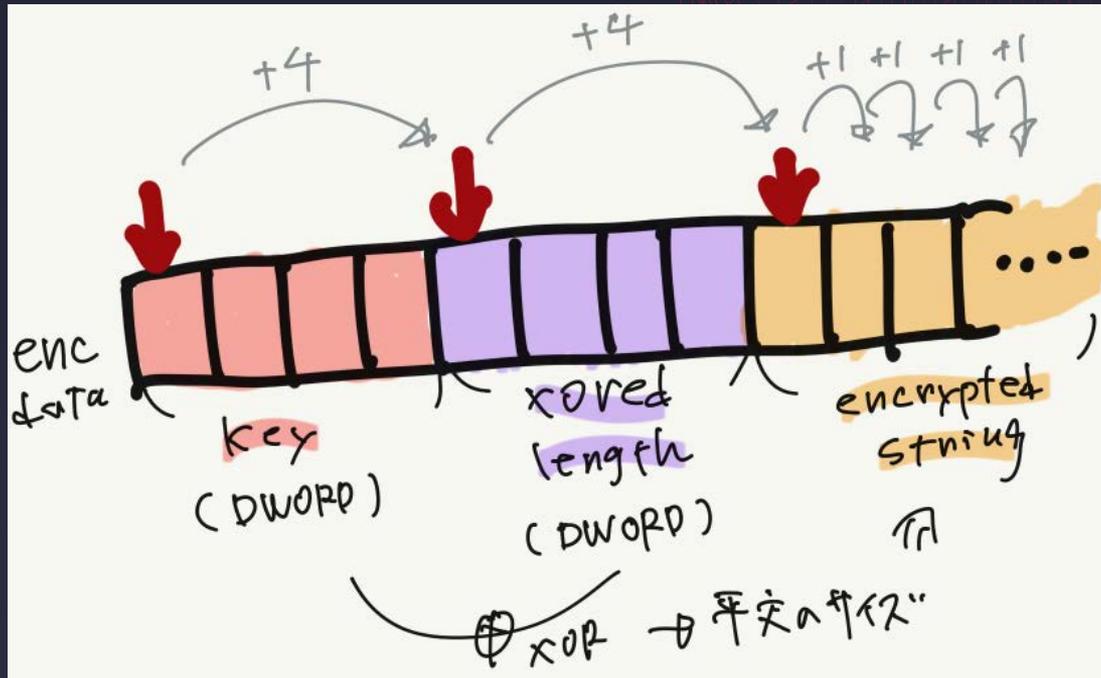
```

Decompile: FUN_1000732d - (emotet.dll)
18  xor_key = data->key;
19  enc_data = &data->enc_data;
20  orig_length = data->xored_length ^ xor_key;
21  uVar8 = orig_length + 1;
22  size = uVar8;
23  if ((uVar8 & 3) != 0) {
24      size = (uVar8 & 0xffffffffc) + 4;
25  }
26  buffer = (ushort *)alloc_memory(uVar8 & 0xfffff03, (void *) (size * 2));
27  if (buffer != (ushort *)0x0) {
28      i = 0;
29      puVar1 = (uint *) ((int)enc_data + (size & 0xffffffffc));
30      size = (uint) ((int)puVar1 + (3 - (int)enc_data) >> 2;
31      if (puVar1 < enc_data) {
32          size = 0;
33      }
34      decrypted = buffer;
35      if (size != 0) {
36          do {
37              chunk = *enc_data;
38              enc_data = enc_data + 1;
39              chunk = chunk ^ xor_key;
40              *decrypted = (ushort)chunk & 0xff;
41              decrypted[1] = (ushort)(chunk >> 8) & 0xff;
42              high_chunk = (ushort)(chunk >> 0x10);
43              i = i + 1;
44              decrypted[2] = high_chunk & 0xff;
45              decrypted[3] = high_chunk >> 8;
46              decrypted = decrypted + 4;
47          } while (i < size);
48      }
49      buffer[orig_length] = 0;
50  }
51  return buffer;

```

# String Decryption Algorithm

- 先頭4バイトがXOR鍵 (keyとする)
- オフセット+4から4バイトがXORされた (平文の) サイズ (xored\_lengthとする)
- $\text{key} \wedge \text{xored\_length}$  が平文のサイズとなる (orig\_lengthとする)
- オフセット+8以降から orig\_length分が暗号文



# Decrypt Strings: Logic

暗号文字列のデータはEDX経由で復号関数 (FUN\_1000732d) に渡されている

```
1000ddb9 ba 80 f5 ... MOV     EDX, DAT_1001f580
1000ddbe e8 6a 95 ... CALL   FUN_1000732d
```

解析結果をもとにGhidra Scriptで復号処理を実装

Address	Disassembly	Comment	Value
1001f580	98 1e 9c 71 ddw	key	719C1E98h
1001f584	b5 1e 9c 71 ddw	xored_length	719C1EB5h
1001f588	cb	enc	CBh
1001f589	51		51h Q
1001f58a	da		DAh
1001f58b	25		25h %
1001f58c	cf		CFh
1001f58d	5f		5Fh -
1001f58e	ce		CEh
1001f58f	34		34h 4
1001f590	c4		C4h
1001f591	53		53h S
1001f592	f5		F5h
1001f593	12		12h
1001f594	ea		EAh
1001f595	71		71h q
1001f596	ef		EFh
1001f597	1e		1Eh
1001f598	f0		F0h

```
Python - Interpreter
>>> import struct
>>> from itertools import cycle
>>>
>>> enc_data_addr = toAddr(0x1001f580)
>>> key = getInt(enc_data_addr)
>>> hex(key)
'0x719c1e98'
>>> xored_length = getInt(enc_data_addr.add(4))
>>> hex(xored_length)
'0x719c1eb5'
>>> original_length = key ^ xored_length
>>> original_length
45
>>> enc = getBytes(enc_data_addr.add(8), original_length).tostring()
>>> enc
'\xcbQ\xda%\xcf_\xce4\xc4S\xf5\x12\xeaq\xef\x1e\xfej\xc0&\xf1p\xf8\x1e\xefm\xc02\xed1\xec'
>>> ''.join([chr(ord(k) ^ ord(v)) for k, v in zip(cycle(struct.pack('<I', key)), enc)])
'SOFTWARE\Microsoft\Windows\CurrentVersion\Run'
```



# Decrypt Strings: Logic

---

- 穴埋め部分を実装

実装したdecrypt\_string\_stage1.pyのdecrypt\_string関数

```
def decrypt_string(enc_data_addr):  
    key = getInt(enc_data_addr)  
    xored_length = getInt(enc_data_addr.add(4))  
    original_length = key ^ xored_length  
    enc = getBytes(enc_data_addr.add(8), original_length).tostring()  
    return xor_with_multi_bytes(enc, key)
```

# Decrypt Strings: How to get arguments?

- ここまでで、暗号データのアドレスがわかれば文字列を復号できることがわかった
- 暗号データのアドレスはEDXレジスタ経由で復号関数 (FUN\_1000732d) の第2引数として渡されている

FUN\_1000732dの参照元を辿ると、EDX経由でアドレスが渡されている

```
1000282a ba c0 f5 ... MOV EDX, DAT_1001f5c0
1000282f e8 f9 4a ... CALL FUN_1000732d
```

```
100156aa ba e0 f9 ... MOV EDX, DAT_1001f9e0
100156af 89 b4 24 ... MOV dword ptr [ESP + local_44], ESI
100156b6 e8 72 1c ... CALL FUN_1000732d
```

```
10001c25 ba 30 f9 ... MOV EDX, DAT_1001f930
10001c2a 89 45 e4 ... MOV dword ptr [EBP + local_20], EAX
10001c2d 81 75 e4 ... XOR dword ptr [EBP + local_20], 0x36d9
10001c34 ff 75 fc ... PUSH dword ptr [EBP + local_8]
10001c37 8b 4d f0 ... MOV this, dword ptr [EBP + local_14]
10001c3a e8 ee 56 ... CALL FUN_1000732d
```

# Decrypt Strings: How to get arguments?

ただし、例外もある

- FUN\_1000732dの「呼び出し元の関数の引数」としてアドレスが渡されているケース
- 本来は対応すべきだが、今回は無視することにする

```
void __fastcall FUN_10004ccb(uint *param_1, int param_2)
{
    ushort *puVar1;
    undefined4 uVar2;

    puVar1 = FUN_1000732d(0x9aa, (astruct *)param_1, 0x3966);
}
```

暗号データのアドレスが呼び出し元関数の引数として渡されている

```
if (uVar1 == 0x1f775c77) {
    FUN_10004ccb((uint *)&DAT_1001f720 0);
    uVar1 = 0x1eb12525;
    goto LAB_10012bb2;
}
if (uVar1 == 0x4024269) {
    FUN_10004ccb((uint *)&DAT_1001f6c0 3);
    uVar1 = 0x2440893e;
    goto LAB_10012bb2;
}
if (uVar1 == 0x66a140d) {
    FUN_10004ccb((uint *)&DAT_1001f780 2);
    uVar1 = 0x4024269;
    goto LAB_10012bb2;
}
if (uVar1 == 0xe1cbd22) {
    FUN_10004ccb((uint *)&DAT_1001f760 6);
    uVar1 = 0x37073aea;
    goto LAB_10012bb2;
}
if (uVar1 == 0x1eb12525) {
    FUN_10004ccb((uint *)&DAT_1001f740 1);
    uVar1 = 0x66a140d;
    goto LAB_10012bb2;
}
```

# Decrypt Strings: How to get arguments?

- 復号関数 (FUN\_1000732d) の呼び出し元の直前の命令を取得し、EDXにMOVされているアドレスを取得
- 特定のアドレスの直前の命令を取得するためのユーティリティ関数

get\_instructions\_beforeがutils.pyに定義済み

```
Python - Interpreter
>>> from utils import *
>>> insts = get_instructions_before(toAddr(0x1000ddbe), 3)
>>> for inst in insts[::-1]:
...     print(inst)
...
PUSH dword ptr [ESP + 0x4c]
MOV ECX,dword ptr [ESP + 0x34]
MOV EDX,0x1001f580
```

1000ddb1	ff 74 24 4c	PUSH	dword ptr [ESP + local_28]
1000ddb5	8b 4c 24 34	MOV	ECX,dword ptr [ESP + local_44]
1000ddb9	ba 80 f5 ...	MOV	EDX,DWORD_1001f580
1000ddbe	e8 6a 95 ...	CALL	FUN_1000732d



EDXにMOVしてる命令だけフィルター

```
>>> def is_mov_edx(inst):
...     return str(inst).startswith('MOV EDX,0x')
...
>>> for inst in filter(is_mov_edx, insts):
...     print(inst)
...
MOV EDX,0x1001f580
```

# Decrypt Strings: Assemble codes

- 復号関数のアドレスはとりあえずハードコードしておく

decrypt\_string\_stage1.pyのmain関数

```
36 def main():
37
38     decrypt_string_addr = toAddr(0x1000732d)
39
40     for xref in getReferencesTo(decrypt_string_addr):
41         # get instructions before callee address
42         insts = get_instructions_before(xref.getFromAddress(), 50)
43
44         # find instruction that passes
45         # address of encrypted data via EDX
46         for inst in insts:
47             if is_mov_edx(inst):
48                 # get encrypted data address and decrypt it
49                 data_addr = inst.getAddress(1)
50                 if data_addr:
51                     decrypted_str = decrypt_string(data_addr)
52                     # add comment
53                     print('[*] found at {}: {!r}'.format(inst.getAddress(), decrypted_str))
54                     add_bookmark_comment(inst.getAddress(), decrypted_str)
```

# Decrypt Strings: Run

- Script Manager経由で  
decrypt\_string\_stage1.pyを実行

実行結果はコンソールに出力される

```

Console - Scripting
decrypt_string_stage1.py> Running...
[*] found at 1000282a: '%s\\rundll32.exe "%s\\%s",Control_RunDLL'
[*] found at 10004103: '%s\\%s'
[*] found at 10005938: '%s\\%s'
[*] found at 10008d16: '%u.%u.%u.%u'
  
```

復号後の文字列がコメントされている

```

uVar6 = 0x752c;
        /* SOFTWARE\Microsoft\Windows\CurrentVersion\Run */
puVar1 = FUN_1000732d(0x5eec, (astruct *) &DWORD_1001f580, 0x752c);

        /* %u.%u.%u.%u */
puVar4 = FUN_1000732d(0x4857, (astruct *) &DAT_1001f7e0, 0x2227);
  
```

Window > Bookmark からもコメントを確認できる



Type	Category	Description	Location
Note	decrypted_str	%s\rundll32.exe "%s\\%s",Control_RunDLL	1000282a
Note	decrypted_str	%s\\%s	10004103
Note	decrypted_str	%s\\%s	10005938
Note	decrypted_str	%u.%u.%u.%u	10008d16
Note	decrypted_str	DNT: 0Referer: %s/%sContent-Type: multipart/form...	10008e88
Note	decrypted_str	SOFTWARE\Microsoft\Windows\CurrentVersion\Run	1000ddb9
Note	decrypted_str	POST	1001058e
Note	decrypted_str	%s\rundll32.exe "%s\\%s",Control_RunDLL	100136d7
Note	decrypted_str	%s\\%s	10014329
Note	decrypted_str	%s:Zone.Identifier	1000b3ae
Note	decrypted_str	%s\rundll32.exe "%s",Control_RunDLL %s	1000f2e6
Note	decrypted_str	%s\\%s%x	1001c33e
Note	decrypted_str	%s\\%s	1001c3c0

# Decrypt Strings Stage2

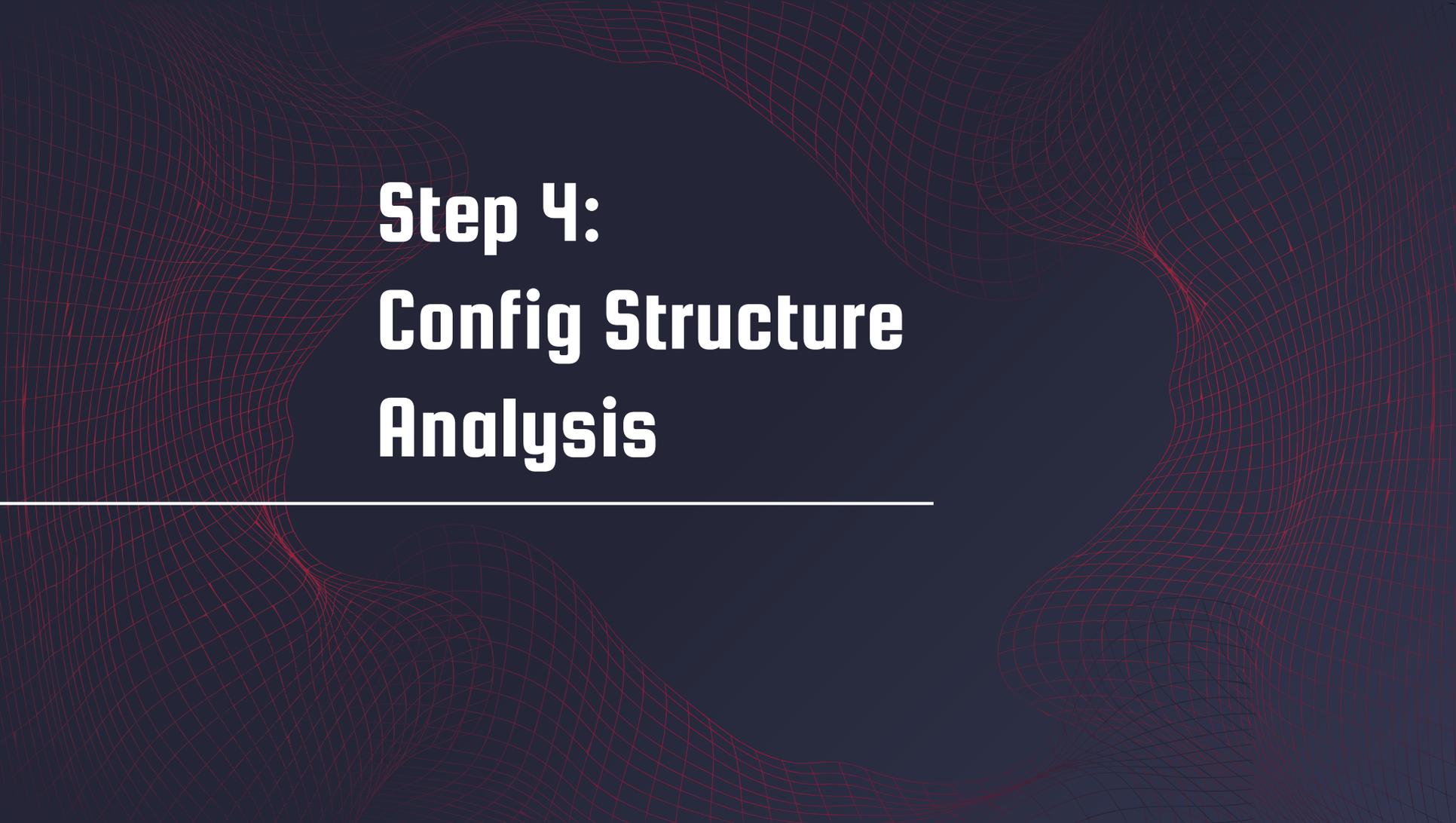
- 復号関数のアドレスは常に固定とは限らない 🤔
- 復号関数を都度手動で解析して特定しては自動化の意味がない 🤔



複数サンプルを解析し、  
復号関数内の共通する特徴的な命令列を見つけ出し、  
それをもとに復号関数を自動特定する

本演習は自主学習にします。

Appendixに課題と解法が書いてあるので興味のある方はトライしてみてください。



**Step 4:**  
**Config Structure**  
**Analysis**

---

# Config Structure Analysis

- EMOTETは通信先を検体内にハードコードしている
- 通信関連のAPIを辿り、ハードコードされた通信先情報の取得を試みる

Bookmarksから、`search_hash.py`で見つけた  
通信関連のAPIを確認できる

✔ Bookmarks - (240 bookmarks)			
Type	Category	Description	Location
Note	hashed_name	InternetCloseHandle	10018d90
Note	hashed_name	InternetConnectW	10002c02
Note	hashed_name	InternetOpenW	10005e2a
Note	hashed_name	InternetReadFile	10005fd7
Note	hashed_name	InternetSetOptionW	10002550

# Find Hostname & Port

## ■ InternetConnectW API

```
void InternetConnectW(
    HINTERNET      hInternet,
    LPCWSTR        lpszServerName,
    INTERNET_PORT  nServerPort,
    LPCWSTR        lpszUserName,
    LPCWSTR        lpszPassword,
    DWORD          dwService,
    DWORD          dwFlags,
    DWORD_PTR      dwContext
);
```

- InternetConnectWを呼び出しているFUN\_10002b4fの第七引数にホスト名、第4引数にポート番号が渡されている

```
Decompile: FUN_10002b4f - (emotet.dll)
1
2 void FUN_10002b4f(undefined4 param_1,undefined4 param_2,undefined param_3,undefined4 param_4,
3     undefined4 param_5,undefined4 param_6,undefined4 param_7,undefined4 param_8,
4     undefined4 param_9,undefined param_10,undefined4 param_11)
5
6 {
7     code *pcVar1;
8
9     _();
10
11     /* InternetConnectW */
12     /* wininet.dll */
13     pcVar1 = (code *)resolve_api(0x71,0x7b,0x2175dc,0x26e080);
14     (*pcVar1)(param_6,param_7,param_4,0,0,param_8,0,0);
15     return;
```

pcVar1をRetype Variableで  
InternetConnectW \*型に変更した後、  
引数名や型を手動で変更する



```
Decompile: call_internetconnectw - (emotet.dll)
1
2 void call_internetconnectw
3     (undefined4 param_1,undefined4 param_2,undefined param_3,INTERNET_PORT port,
4     undefined4 param_5,HINTERNET param_6,LPCWSTR hostname,DWORD param_8,
5     undefined4 param_9,undefined param_10,undefined4 param_11)
6
7 {
8     InternetConnectW *pcVar1;
9
10    _();
11
12    /* InternetConnectW */
13    /* wininet.dll */
14    pcVar1 = (InternetConnectW *)resolve_api(0x71,0x7b,0x2175dc,0x26e080);
15    (*pcVar1)(param_6,hostname,port,(LPCWSTR)0x0,(LPCWSTR)0x0,param_8,0,0);
16    return;
```

# Find Hostname & Port

- `call_internetconnectw` (`FUN_10002b4f`) の呼び出し元を辿ると、`FUN_1000f70c`が呼び出していることがわかる
- ホスト名が渡されている第七引数は、`FUN_1000f70c`の第引数から渡されている
- ポート番号が渡されている第4引数には、`in_stack_00000024`という変数が渡されている
  - 実際はポート番号も `FUN_1000f70c`の引数として渡されているのだが、Ghidraが関数の解析に失敗していて、解釈がおかしくなっている

`FUN_1000f70c`内で`call_internetconnectw`を呼び出している部分

```
local_140 = call_internetconnectw
                                     ↓ ポート番号
                                     (0x76f3,0x62c4, (char)local_144, in_stack_00000024 0x5b4,local_1c,
ホスト名→ local_10 3,0x32bd, (char)local_144,0x65b6);
```

`call_internetconnectw`に渡されているホスト名は`FUN_1000f70c`の第二引数から渡される

```
2 undefined4 __fastcall
3 FUN_1000f70c(undefined4 param_1,int param_2,undefined4 param_3,int **param_4,undefined4 *param_5,
4   undefined4 param_6,undefined4 param_7)
5
6 {
7   undefined4 uVar1;
8   int iVar2;
9   void *this;
10  uint uVar3;
11  LPCWSTR pWVar4;
12  undefined extraout_CL;
13  undefined4 extraout_ECX;
14  undefined4 uVar5;
15  undefined4 uVar6;
16  int iVar7;
17  INTERNET_PORT in_stack_00000024;
18  void *local_144;
19  int local_140;
20  int local_1c;
21  undefined4 local_18;
22  undefined4 local_14;
23  LPCWSTR local_10;
24  ushort *local_c;
25  undefined4 local_8;
26  undefined4 local_4;
27
28  local_10 = (LPCWSTR)param_2;
```

# Fix Function: FUN\_1000f70c

- FUN\_1000f70cの引数の解釈にGhidraが失敗している
- そのため、Edit Function Signatureで引数の数を調整する

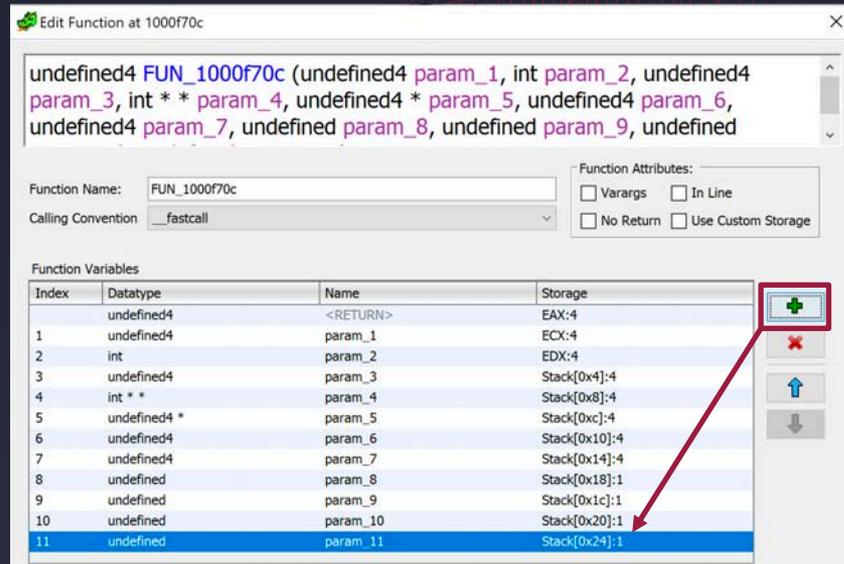
FUN\_1000f70cの呼び出し元を見てみると  
ECX, EDX, PUSH\*9で引数を渡している

```

10008a0a ff b4 24 ...   PUSH    dword ptr [ESP + local_b2c]
10008a11 83 a4 24 ...   AND     dword ptr [ESP + local_b34], 0x0
10008a19 8d 84 24 ...   LEA    EAX=>local_800, [ESP + 0x46c]
10008a20 ff 74 24 50    PUSH    dword ptr [ESP + local_c1c]
10008a24 83 a4 24 ...   AND     dword ptr [ESP + local_b30], 0x0
10008a2c 8d 94 24 ...   LEA    EDX=>local_b00, [ESP + 0x170]
10008a33 ff 74 24 7c    PUSH    dword ptr [ESP + local_bf4]
10008a37 ff 74 24 30    PUSH    dword ptr [ESP + local_c44]
10008a3b 50           PUSH    EAX
10008a3c ff b4 24 ...   PUSH    dword ptr [ESP + local_b78]
10008a43 8b 4c 24 6c    MOV    ECX, dword ptr [ESP + local_c14]
10008a47 8d 84 24 ...   LEA    EAX=>local_b3c, [ESP + 0x144]
10008a4e 50           PUSH    EAX
10008a4f 8d 84 24 ...   LEA    EAX=>local_b34, [ESP + 0x150]
10008a56 50           PUSH    EAX
10008a57 8d 84 24 ...   LEA    EAX=>local_a00, [ESP + 0x288]
10008a5e 50           PUSH    EAX
10008a5f e8 a8 6c ...   CALL   FUN_1000f70c

```

FUN\_1000f70cを右クリック > Edit Function Signatureで編集  
Add Parameterで引数の数が合計11になるまで追加



# Fix Function: FUN\_1000f70c

- 正しく引数の数を調整すると、FUN\_1000f70cの第1引数にポート番号が渡され、call\_internetconnectwの第4引数に渡されていることがわかる

```
local_140 = call_internetconnectw
           (0x76f3, 0x62c4, (char)local_144, (INTERNET_PORT) param_11, 0x5b4,
           local_1c, local_10, 3, 0x32bd, (char)local_144, 0x65b6);
```

- FUN\_1000f70cの呼び出し元でも引数が反映されている（名前は手動で変更）

```
iVar7 = FUN_1000f70c(0x58a3, (int)hostname, local_a00, &local_b34, &local_b3c, 0x4e48, local_800
                    , 0x98, 0x53, 0x9b, (char)port);
```

# FUN\_10007e95

- FUN\_1000f70cを呼び出している関数がFUN\_10007e95
- portへのアサインは見えるが、hostnameがどこからアサインされているかわからない
- ポート番号はDAT\_1001fa08からのオフセット0x24+0x4に格納されている
  - → 構造体として定義しておく

portがアサインされている箇所

```

/* %u.%u.%u.%u */
puVar4 = decrypt_string(0x4857, (astruct *)&DAT_1001f7e0, 0x2227);
bVar1 = *(byte *) (*(int *) (DAT_1001fa08 + 0x24) + 1);
FUN_10017f0f((uint)*(byte *) (*(int *) (DAT_1001fa08 + 0x24) + 2), 0x1abe, bVar1, 0x62f4, 0x16d7,
             (uint)**(byte **) (DAT_1001fa08 + 0x24), 0x57f1, 0x190a, 0x4e52, (uint)bVar1,
             puVar4, (uint)*(byte *) (*(int *) (DAT_1001fa08 + 0x24) + 3));
FUN_10007c8f(0xcd4);
iVar7 = 0x2351b248;
port = (uint)*(ushort *) (*(int *) (DAT_1001fa08 + 0x24) + 4);

```



DAT\_1001fa08に対し、Create Auto Structureで構造体定義

```

/* %u.%u.%u.%u */
puVar4 = decrypt_string(0x4857, (astruct *)&DAT_1001f7e0, 0x2227);
bVar1 = PTR_1001fa08->field_0x24[1];
FUN_10017f0f((uint)PTR_1001fa08->field_0x24[2], 0x1abe, bVar1, 0x62f4, 0x16d7,
             (uint)*PTR_1001fa08->field_0x24, 0x57f1, 0x190a, 0x4e52, (uint)bVar1, puVar4,
             (uint)PTR_1001fa08->field_0x24[3]);
FUN_10007c8f(0xcd4);
iVar7 = 0x2351b248;
port = (uint)*(ushort *) (PTR_1001fa08->field_0x24 + 4);

```

# FUN\_10017f0f

- フォーマット文字列を第 1 1 引数として受け取る
  - IPアドレスのようなフォーマット
- FUN\_10017f0fは、実質 `_snwprintf` を呼び出すだけ

```
int _snwprintf(
    wchar_t *buffer,
    size_t count,
    const wchar_t *format [,
    argument] ...
);
```

IPアドレスのようなフォーマット文字列が渡される

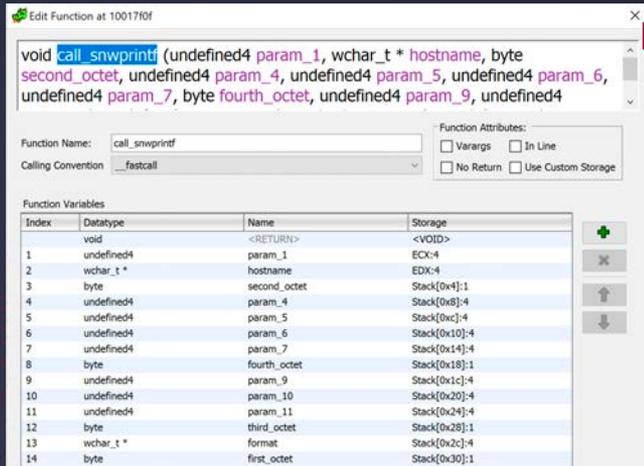
```
/* %u.%u.%u.%u */
puVar4 = decrypt_string(0x4857, (astruct *)&DAT_1001f7e0, 0x2227);
bVar1 = PTR_1001fa08->field_0x24[1];
FUN_10017f0f((uint)PTR_1001fa08->field_0x24[2], 0x1abe, bVar1, 0x62f4, 0x16d7,
             (uint)*PTR_1001fa08->field_0x24, 0x57f1, 0x190a, 0x4e52, (uint)bVar1, puVar4,
             (uint)PTR_1001fa08->field_0x24[3]);
```

```
Decompile: FUN_10017f0f - (emotet.dll)
1
2 void __cdecl
3 FUN_10017f0f(undefined4 param_1, undefined4 param_2, undefined4 param_3, undefined4 param_4,
4             undefined4 param_5, undefined4 param_6, undefined4 param_7, undefined4 param_8,
5             undefined4 param_9, undefined4 param_10, undefined4 param_11, undefined4 param_12)
6
7 {
8     code *pcVar1;
9     undefined4 extraout_ECX;
10    undefined4 extraout_EDX;
11    undefined4 uVar2;
12    undefined4 uVar3;
13
14    _();
15    uVar3 = 0x40;
16    uVar2 = extraout_EDX;
17
18    /* _snwprintf */
19    pcVar1 = (code *)FUN_10018202(extraout_ECX, 0x8440e238, 0x34c);
20    (*pcVar1)(uVar2, uVar3, param_11, param_12, param_1, param_10, param_6);
21    return;
```

# Fix Function: FUN\_10017f0f

- 一見問題ないようだが、実は呼び出し規約と引数の数が間違っている
  - `__cdecl` -> `__fastcall`
  - 引数の数 -> 14

Edit Function Signatureで修正



関数シグネチャ修正後、変数名・型を変更した結果 (param\_2とextraout\_EDXは実際同一だが、変数名が同期されず、今回は無視)

```

C:\Decompile: call_snwprintf - (emotet.dll)
1
2 void __fastcall
3 call_snwprintf(undefined4 param_1, wchar_t *hostname, byte second_octet, undefined4 param_4,
4               undefined4 param_5, undefined4 param_6, undefined4 param_7, byte fourth_octet,
5               undefined4 param_9, undefined4 param_10, undefined4 param_11, byte third_octet,
6               wchar_t *format, byte first_octet)
7
8 {
9     _snwprintf *_snwprintf;
10    undefined4 extraout_ECX;
11    wchar_t *extraout_EDX;
12    undefined3 in_stack_00000005;
13    undefined3 in_stack_00000019;
14    undefined3 in_stack_00000029;
15    undefined3 in_stack_00000031;
16    wchar_t *_Dest;
17    size_t _Count;
18
19    _();
20    _Count = 0x40;
21    _Dest = extraout_EDX;
22
23    /* _snwprintf */
24    _snwprintf = (_snwprintf *)FUN_10018202(extraout_ECX, 0x8440e238, 0x34c);
25    (*_snwprintf)(_Dest, _Count, format, first_octet, second_octet, third_octet, fourth_octet);
26    return;
27 }
  
```

# Config Structure

- call\_snwprintfでホスト名 (IP) が生成されることがわかった
- IPアドレスの各オクテットがcall\_snwprintfの引数に渡されている
- ここまでの情報から、0x1001fa08には次のような構造体へのポインタが存在していると推測できる
  - ✓ オフセット+0x24にはIPアドレスへのポインタが格納されているはず
  - ✓ IPアドレスはリトルエンディアンの4バイト (1オクテット1バイト)
  - ✓ また、オフセット+0x24+0x4にポート番号が格納されているはず

```
/* %u.%u.%u.%u */ 第2オクテット
format = decrypt_string(0x4857, (astruct *)&DAT_1001f7e0, 0x2227);
uVar9 = PTR_1001fa08->field_0x24[1]; 第3オクテット
call_snwprintf(0xc6a, hostname, PTR_1001fa08->field_0x24[2], 0x1abe, (uint)uVar9, 0x62f4, 0x16d7
  第1オクテット *PTR_1001fa08->field_0x24, 0x57f1, 0x190a, 0x4e52, uVar9, (wchar_t *)format,
  PTR_1001fa08->field_0x24[3]);
FUN_10007c8f(0xcd4); 第4オクテット
iVar5 = 0x2351b248; ポート番号
port = (uint)*(ushort *) (PTR_1001fa08->field_0x24 + 4);
```

# Config Structure

- ここまでの情報をまとめて、次のような構造体を定義

擬似構造体

```
typedef struct {
    byte ip_address[4];
    WORD port;
} C2Info;
```

```
typedef struct {
    ...
    // at offset 0x24
    C2Info *c2;
} Config;
```

Data Type Manager > emotet.dll 右クリック > New > Structure でC2Infoを新規作成し、次にPTR\_1001fa08を右クリック > Edit Data Type からオフセット36をC2Info\*に変更

The image shows two screenshots of the Structure Editor in Visual Studio. The top screenshot shows the 'C2Info' structure being defined with fields 'ip\_address' (4 bytes) and 'port' (2 bytes). The bottom screenshot shows the 'Config' structure being edited, where the 'c2' field at offset 36 is set to be a pointer to 'C2Info'.

**Structure Editor - C2Info (emotet.dll)**

Offset	Length	Mnemonic	Data Type	Name	Comment
0	4	db[4]	byte[4]	ip_address	
4	2	dw	word	port	

Search:

Byte Offset: 5 4 3 2 1 0

Component Bits: port ip\_address

Name: C2Info

Description:

Category: emotet.dll/auto

Size: 6

**Structure Editor - Config (emotet.dll)**

Offset	Length	Mnemonic	Data Type	Name	Comment
33	1	??	undefined		
34	1	??	undefined		
35	1	??	undefined		
36	4	C2Info *	C2Info *	c2_info	

Search:

Byte Offset: 39 38 37 36 35 34 33

Component Bits: c2\_info

Name: Config

Description:

Category: emotet.dll/auto\_structs

Size: 40 Alignment: 1  Align

# Config Structure

- 構造体が適用されると見やすくなる
- ここで生成されたIPアドレスとポート番号が、先ほど解析したFUN\_1000f70cに渡され、通信が発生する
- つまり、これらのIPアドレスとポート番号がC&Cサーバの情報となることがわかる

```
        /* %u.%u.%u.%u */  
format = decrypt_string(0x4857, (astruct *)&DAT_1001f7e0, 0x2227);  
uVar9 = g_config->c2_info->ip_address[1];  
call_snwprintf(0xc6a, hostname, g_config->c2_info->ip_address[2], 0x1abe, (uint)uVar9, 0x62f4,  
              0x16d7, g_config->c2_info->ip_address[0], 0x57f1, 0x190a, 0x4e52, uVar9,  
              (wchar_t *)format, g_config->c2_info->ip_address[3]);  
FUN_10007c8f(0xcd4);  
iVar5 = 0x2351b248;  
port = (uint)g_config->c2_info->port;
```

# Config Structure

- `g_config(PTR_1001fa08)` はポインタを保持するだけのグローバル変数なので、`Config`構造体の実態へのポインタは実行時に格納される
- では、いつ・どこで初期化されるのか？

`g_config`の参照元一覧を確認すると、WRITEしているのは0x100e258の一箇所のみ

Uses of "Config \*" (DataType) - 34 locations [CodeBrowser(2): ghidra\_project:/emotet.dll]

Help

Uses of "Config \*" (DataType) - 34 locations

Loca...	Label	Code Unit	Context	Function Name
10008d21		MOV ECX,dword ptr [g_config]	READ	FUN_10007e95
10008d29		MOV EDX,dword ptr [ECX + 0x24]	174: third_octet = g_config->c2_info->ip_ad	FUN_10007e95
10008d29		MOV EDX,dword ptr [ECX + 0x24]	177: third_octet,(wchar_t *)puVar3,g_config-	FUN_10007e95
10008d50		MOV EAX,[g_config]	READ	FUN_10007e95
10008d55		MOV EAX,dword ptr [EAX + 0x24]	176: 0x62f4,0x16d7,g_config->c2_info->ip_ad	FUN_10007e95
10008d6c		MOV EAX,[g_config]	READ	FUN_10007e95
10008d78		MOV EAX,dword ptr [EAX + 0x24]	175: call_snwprintf(0xc6a,hostname,g_config->	FUN_10007e95
10008d9a		MOV EAX,[g_config]	READ	FUN_10007e95
10008da7		MOV EAX,dword ptr [EAX + 0x24]	180: port = (uint)g_config->c2_info->port;	FUN_10007e95
1000e10b	FUN_1...	PUSH EBP	5: Config *pCVar1;	FUN_1000e10b
1000e11f		CALL _	11: if (pCVar1 != (Config *)0x0) {	FUN_1000e10b
1000e24e		CALL alloc_memory	9: pCVar1 = (Config *)alloc_memory(0x25,&DA	FUN_1000e10b
1000e24e		CALL alloc_memory	9: pCVar1 = (Config *)alloc_memory(0x25,&DA	FUN_1000e10b
1000e258		MOV dword ptr [g_config],EDX	WRITE	FUN_1000e10b
1000e25e		TEST EDX,EDX	11: if (pCVar1 != (Config *)0x0) {	FUN_1000e10b
1000e260		JZ LAB_1000e2b1	10: g_config = pCVar1;	FUN_1000e10b
1000e260		JZ LAB_1000e2b1	10: g_config = pCVar1;	FUN_1000e10b

# FUN\_1000e10b

## Config構造体を初期化する関数

1. Config構造体用のバッファを確保し、g\_configへポインタをコピー
2. DAT\_1001f000からC2Info構造体のポインタをコピー
3. DAT\_1001f000には、1要素あたり8バイトの配列が格納されている

pcVar1を右クリック > Auto Fill in Structure して  
Config構造体をアップデートした後のコード

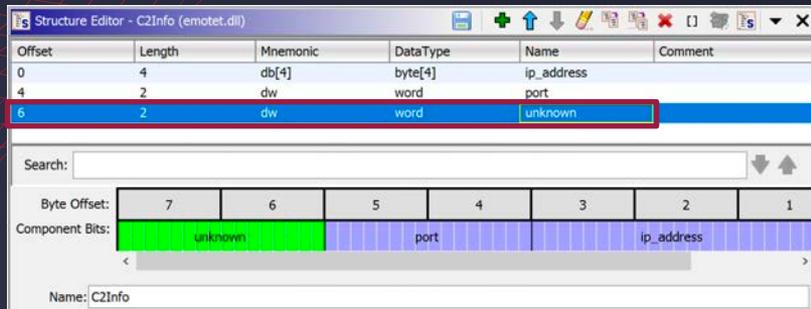
```

Decompile: FUN_1000e10b - (emotet.dll)
1
2 undefined4 __cdecl FUN_1000e10b(undefined4 *param_1)
3
4 {
5     Config *pCVar1;
6     int iVar1;
7
8     _();
9     pCVar1 = (Config *)alloc_memory(0x25,&DAT_0000002c);
10    g_config = pCVar1; ①
11    if (pCVar1 != (Config *)0x0) {
12        iVar1 = pCVar1->field_0x8;
13        pCVar1->field_0x28 = &DAT_1001f000; ②
14        pCVar1->c2_info = (C2Info *)&DAT_1001f000; ③
15        pCVar1->field_0x10 = 0;
16        while (*(int *)(&DAT_1001f000 + iVar1 * 8) != 0) {
17            iVar1 = iVar1 + 1;
18            pCVar1->field_0x8 = iVar1;
19        }
20        iVar1 = FUN_1001829c(param_1,0x35,0x2807,0x110f);
21        if (iVar1 != 0) {
22            return 1;
23        }
24        FUN_1000d4e0(g_config,0xd0,0x3dc4,0x573f);
25    }
26    return 0;
27 }

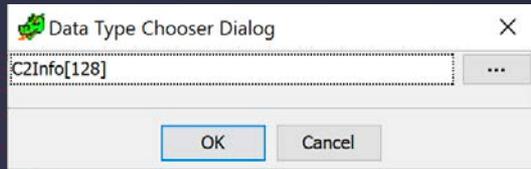
```

# FUN\_1000e10b

- 現在C2Info構造体は6バイトなので、（使用用途は不明だが）2バイト分追加しておく



- DAT\_1001f000(g\_c2info\_array)をRetype Globalし、C2Info構造体の配列として定義



## 編集後のFUN\_1000e10b

```
Decompile: FUN_1000e10b - (emotet.dll)
1
2 undefined4 __cdecl FUN_1000e10b(undefined4 *param_1)
3
4 {
5     Config *tmp;
6     int i;
7
8     _();
9     tmp = (Config *)alloc_memory(0x25, &DAT_0000002c);
10    g_config = tmp;
11    if (tmp != (Config *)0x0) {
12        i = tmp->field_0x8;
13        tmp->field_0x28 = (undefined *)g_c2info_array;
14        tmp->c2_info = g_c2info_array;
15        tmp->field_0x10 = 0;
16        while (*(int *)g_c2info_array[i].ip_address != 0) {
17            i = i + 1;
18            tmp->field_0x8 = i;
19        }
20    }
```

# Array of C2Info

- DAT\_1001f000(g\_c2info\_array) にはIPアドレス、ポート番号が平文で格納されている
- このデータをパースすれば通信先情報を取得できそう

```

g_c2info_array XREF[6]:
1001f000 3c d7 00 ... C2Info[1...
C2Info 1001f000 3c d7 00 7d 50...C2Info[0]
1001f000 3c d7 00 7d db[4] ip_address
1001f000 [0] 3Ch, D7h, 0h, 7Dh ip_address
1001f004 50 00 dw 50h port port
1001f006 e4 d4 dw D4E4h unknown
C2Info 1001f008 b4 cc 35 a3 bb...C2Info
1001f008 b4 cc 35 a3 db[4] ip_address [1] ip_address
1001f008 [0] B4h, CCh, 35h, A3h ip_address
1001f00c bb 01 dw 1BBh port port
1001f00e 49 ef dw EF49h unknown
C2Info 1001f010 8d d2 a3 59 90...C2Info
1001f010 8d d2 a3 59 db[4] ip_address [2] ip_address
1001f010 [0] 8Dh, D2h, A3h, 59h ip_address
1001f014 90 1f dw 1F90h port port
1001f016 34 57 dw 5734h unknown
1001f018 09 98 9d cb a8...C2Info [3]

```

# Protocol of Connection

IPとポートは判明したが、プロトコル(HTTP/HTTPS)は？

- HTTPSを使う場合、WinInet系APIではHttpOpenRequestのdwFlagsでINTERNET\_FLAG\_SECURE(0x800000)が指定される

HttpOpenRequestの関数シグネチャ

```
void HttpOpenRequestW(  
    HINTERNET hConnect,  
    LPCWSTR   lpszVerb,  
    LPCWSTR   lpszObjectName,  
    LPCWSTR   lpszVersion,  
    LPCWSTR   lpszReferrer,  
    LPCWSTR   *lplpszAcceptTypes,  
    DWORD      dwFlags,  
    DWORD_PTR dwContext  
);
```

INTERNET\_FLAG\_SECURE

0x00800000

Uses secure transaction semantics. This translates to using Secure Sockets Layer/Private Communications Technology (SSL/PCT) and is only meaningful in HTTP requests. This flag is used by [HttpOpenRequest](#) and [InternetOpenUrl](#), but this is redundant if https:// appears in the URL. The [InternetConnect](#) function uses this flag for HTTP connections; all the request handles created under this connection will inherit this flag.

<https://docs.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-httpopenrequestw>

# Protocol of Connection

HttpOpenRequestWの呼び出し元を辿ると、INTERNET\_FLAG\_SECURE(0x800000)のビットは立っていないので、プロトコルはHTTPが使用されることがわかる

```
Decompile: call_httpopenrequestw - (emotet.dll)
1
2 void __cdecl
3 call_httpopenrequestw
4     (undefined4 param_1, undefined4 param_2, LPCWSTR param_3, undefined4 param_4,
5     undefined4 param_5, HINTERNET param_6, undefined4 param_7, DWORD dwFlags, undefined4 param_9,
6     undefined param_10, undefined4 param_11, LPCWSTR param_12)
7
8 {
9     HttpOpenRequestW *pcVar1;
10    undefined4 _;
11
12    ::_();
13
14    /* HttpOpenRequestW */
15    /* wininet.dll */
16    pcVar1 = (HttpOpenRequestW *) resolve_api(0x379, __, 0x2175dc, 0x89d4a7f4);
17    (*pcVar1)(param_6, param_3, param_12, (LPCWSTR) 0x0, (LPCWSTR) 0x0, (LPCWSTR *) 0x0, dwFlags, 0);
18    return;
19 }
```

```
pWVar4 = (LPCWSTR) call_httpopenrequestw
(0x5c18, 0x844cc300, local_c, 0x7505, 0x1cf0, local_140, 0x844cc300,
0x844cc300, 0x3639, 0, 0x6b8d, param_3);
```

# Extract C2 in Action

extract\_c2\_stage1\_not\_impl.py  
のparse\_single\_configを実装して  
ください

- アドレス0x1001f000以降に存在しているIPアドレスとポートを以下形式で出力する
  - http://<ip-address>:<port>

g\_c2info\_array XREF[6]:

Address	Value	Type
1001f000	3c d7 00 ...	C2Info[1...]
1001f000	3c d7 00 7d 50...	C2Info [0]
1001f000	3c d7 00 7d	db[4] ip_address
1001f000	[0]	3Ch, D7h, 0h, 7Dh ip_address
1001f004	50 00	dw 50h port
1001f006	e4 d4	dw D4E4h unknown
1001f008	b4 cc 35 a3 bb...	C2Info [1]
1001f008	b4 cc 35 a3	db[4] ip_address
1001f008	[0]	B4h, CCh, 35h, A3h ip_address
1001f00c	bb 01	dw 1BBh port
1001f00e	49 ef	dw EF49h unknown
1001f010	8d d2 a3 59 90...	C2Info [2]
1001f010	8d d2 a3 59	db[4] ip_address
1001f010	[0]	8Dh, D2h, A3h, 59h ip_address
1001f014	90 1f	dw 1F90h port
1001f016	34 57	dw 5734h unknown
1001f018	09 98 9d cb a8...	C2Info [3]

# Extract C2 in Action

1. `g_c2info_array`の先頭アドレスから処理を進める
2. 先頭4バイトをIPアドレス、オフセット+4から2バイトをポート番号としてパース
3. 8バイト(`C2Info`構造体のサイズ)分アドレスを進める

```
Python - Interpreter
>>> from extract_c2_stage1 import *
>>> extract_c2(toAddr(0x1001f000))
http://125. [REDACTED] 80
http://163. [REDACTED] 0:443
http://89.1 [REDACTED] 1:8080
http://203. [REDACTED] :7080
http://157. [REDACTED] 7:443
```



## 実装したextract\_c2\_stage1.py

```

14 def parse_single_config(addr):
15     # get 4 bytes from top of the given address
16     # this should be IP address
17     ip = convert_dword_to_ip(getInt(addr))
18
19     # '0.0.0.0' is a mark of the end of config
20     if ip == '0.0.0.0':
21         raise InvalidConfig('Invalid Config or found Config end')
22
23     # get 2 bytes from offset 4 of the given address,
24     # this should be port number
25     port = getShort(addr.add(SIZE_OF_IP))
26     return ip, port
27
28 def iterate_config(config_addr):
29     while True:
30         try:
31             ip, port = parse_single_config(config_addr)
32             config_addr = config_addr.add(SIZE_OF_SINGLE_CONFIG)
33             yield (ip, port)
34         except InvalidConfig:
35             # when parse_single_config throw InvalidConfig exceptoin,
36             # it should be that it finished iteration of config
37             break
38         except Exception as e:
39             print('[!] {}'.format(e))
40
41 def extract_c2(config_addr):
42     for (ip, port) in iterate_config(config_addr):
43         print('http://{}:{}'.format(ip, port))
44

```

## Extract C2 Stage2

---

- コンフィグのアドレスは常に固定とは限らない 🤔
- コンフィグのアドレスを都度手動で解析・特定しては自動化の意味がない 🤔



複数サンプルを解析し、コンフィグ初期化関数内で  
共通する特徴的な命令列を見つけ出し、  
それをもとにコンフィグのアドレスを自動特定する

# How to compare samples?

## Version Tracking

- 2つのバイナリの差分を解析するGhidraの機能
  - パッチの差分解析
  - 共通する命令列の可視化

差分表示の例

Source: **resolve\_api()** in /emotet.exe

```

00402db0 55      PUSH    ESP
00402db1 8b ec   MOV     EBP, ESP
00402db3 56      PUSH    ESI
00402db4 8b 75 08 MOV     ESI, dword ptr [EBP + index]
00402db7 57      PUSH    EDI
00402db8 8b fa   MOV     EDI, hex_value2
00402dba 83 3c b5 CMP     dword ptr [ESI*0x4 + g_array], 0x0
          90 9a 40
          00 00
00402dc2 75 15   JNZ     LAB_00402dd9
00402dc4 e8 24 ff CALL    find_lib
          ff ff
00402dc9 8b d7   MOV     hex_value2, EDI
00402dcb 8b c8   MOV     hex_value1, EAX
00402dcd e8 86 fe CALL    FUN_00402c58
          ff ff
00402dd2 89 04 b5 MOV     dword ptr [ESI*0x4 + g_array], EAX
          90 9a 40 00
LAB_00402dd9                                     XREF[1]: 0040
00402dd9 8b 04 b5 MOV     EAX, dword ptr [ESI*0x4 + g_array]
          90 9a 40 00
00402de0 5f      POP     ESI
  
```

Destination: **FUN\_00402e99()** in /emotet-2.exe

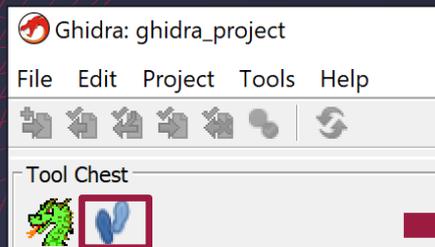
```

00402e99 56      PUSH    ESI
00402e9a 8b 74 24 08 MOV     EDI, dword ptr [ESP + param_3]
00402e9e 57      PUSH    EDI
00402e9f 8b fa   MOV     EDI, param_2
00402ea1 83 3c b5 CMP     dword ptr [ESI*0x4 + DAT_00409a30], 0x0
          30 9a 40
          00 00
00402ea9 75 15   JNZ     LAB_00402ec0
00402eab e8 26 ff CALL    FUN_004023d6
          ff ff
00402eb0 8b d7   MOV     param_2, EDI
00402eb2 8b c8   MOV     param_1, EAX
00402eb4 e8 88 fe CALL    FUN_00402d41
          ff ff
00402eb9 89 04 b5 MOV     dword ptr [ESI*0x4 + DAT_00409a30], EAX
          30 9a 40 00
LAB_00402ec0                                     XREF[1]: 0040
00402ec0 8b 04 b5 MOV     EAX, dword ptr [ESI*0x4 + DAT_00409a30]
          30 9a 40 00
00402ec7 5f      POP     EDI
00402ec8 5e      POP     ESI
00402ec9 c3      RET
  
```

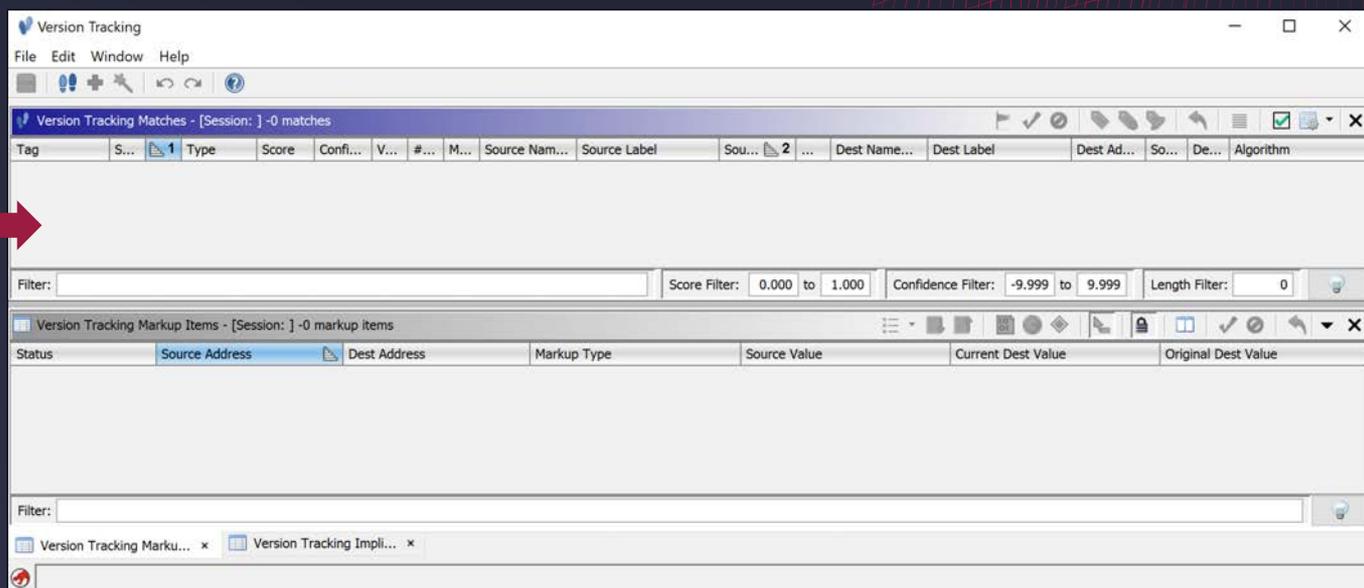
# Version Tracking: Main Window

- Version Trackingで、関数を比較したりする際に使用するウィンドウ
- 最初はなにも表示されない

Ghidraのプロジェクト画面から開く



Version Tracking Windowの初期画面

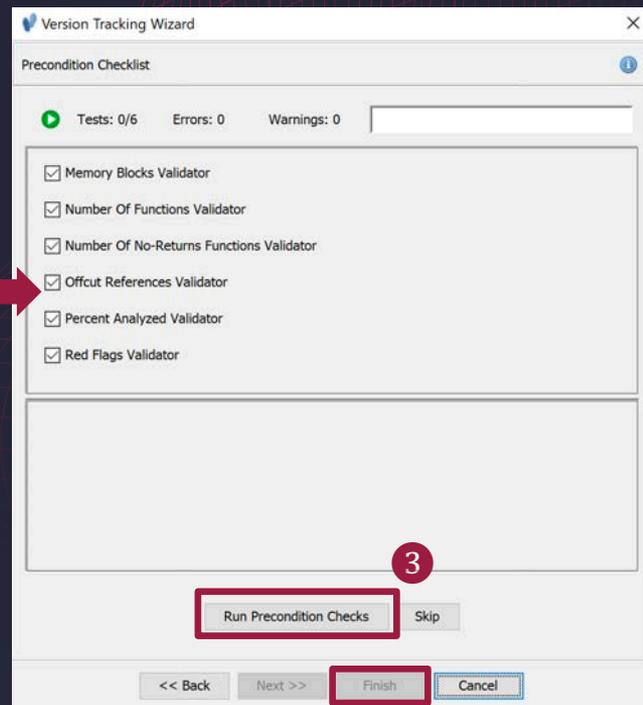
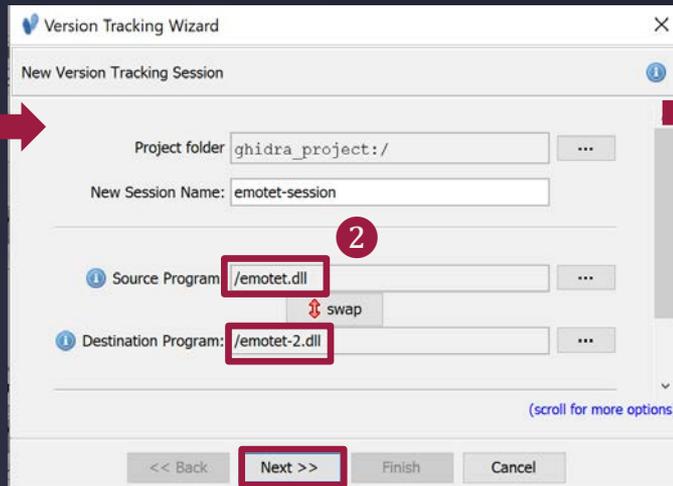
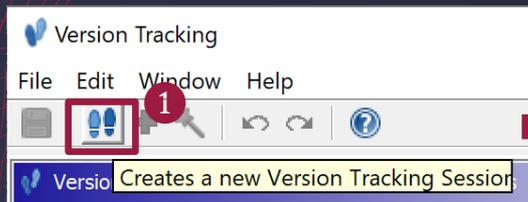


# Version Tracking: Session

バイナリの比較結果を**Session**という情報として保持

## ■ Sessionの作成手順

1. Version Tracking Window > [Create a new Version Tracking Session]
2. Source ProgramとDestination Programを指定
3. [Run Precondition Checks]を実行



# Version Tracking: Compare Functions

Version Tracking Functionsを使うと、比較したい関数を入力すると差分が可視化できる  
(emotet.dllの0x1000e10bとemotet-2.dllの0x1001a094の比較例)

Version Tracking: emotet-session

File Edit Window Help

Version Tracking Functions

Version Tracking Implied Matches

Version Tracking Markup Items

Version Tracking Functions - [Session: emotet-session] - Source Functions - 1 functions (of 243) / Destination Functions - 1 functions (of 238)

Source	Destination
Label: FUN_1000e10b, Location: 1000e10b, Function Signature: undefined4 FUN_1000e10b(undefi...	Label: FUN_1001a094, Location: 1001a094, Function Signature: undefined4 FUN_1001a094(undefi...

Filter: 1000e10b

Filter: 1001a094

Decompile View Listing View

Source: FUN\_1000e10b() in /emotet.dll

```

FUN_1000e10b XREF[1]:
1000e10b 55      PUSH     EBX
1000e10c 8b ec    MOV     EBP,ESP
1000e10e 83 ec 28  SUB     EBX,0x28
1000e111 56      PUSH     ESI
1000e112 ff 75 0c PUSH     dword ptr [EBP + Stack[0x0]]
1000e115 be 00 f0 MOV     ESI,DAT_1001f000
01 10
1000e11a ff 75 08 PUSH     dword ptr [EBP + param_1]
1000e11d 56      PUSH     ESI->DAT_1001f000
1000e11e 51      PUSH     EAX
1000e11f e8 ee 9c CALL    [00 00]
c7 45 dc MOV     dword ptr [EBP + local_28],0x6155
55 61 00 00
1000e12b 33 d2    XOR     EBX,EBX
1000e12d c1 6d dc SHR     dword ptr [EBP + local_28],0x2
  
```

Destination: FUN\_1001a094() in /emotet-2.dll

```

FUN_1001a094 XREF[1]:
1001a094 55      PUSH     EBX
1001a095 8b ec    MOV     EBP,ESP
1001a097 83 ec 30  SUB     EBX,0x30
1001a09a 53      PUSH     EBX
1001a09b 56      PUSH     ESI
1001a09c 57      PUSH     EDI
1001a09d ff 75 14 PUSH     dword ptr [EBP + param_6]
1001a0a0 8b f2    MOV     EBX,param_2
1001a0a2 bb 00 f2 MOV     EBX,DAT_1001f200
01 10
1001a0a7 ff 75 10 PUSH     dword ptr [EBP + param_5]
1001a0aa ff 75 0c PUSH     dword ptr [EBP + param_4]
1001a0ad ff 75 08 PUSH     dword ptr [EBP + param_3]
1001a0b0 56      PUSH     ESI
1001a0b1 53      PUSH     EBX->DAT_1001f200
1001a0b2 e8 ba 40 CALL    FUN_1000e171
  
```

**ブルー:** 一致していないコードユニット

**グレー:** 部分的に一致していないコードユニット

**グリーン:** バイト列、ニーモニック、オペランドのいずれかが異なっている箇所

**ホワイト:** 完全一致しているコードユニット

# Decrypt Strings Stage2: Find Function

extract\_c2\_stage2\_not\_impl.pyのfind\_config\_addrssを実装する

1. Version Trackingで複数サンプルの命令列を比較
  - Source -> emotet.dll: 0x1000e10b (Version Tracking Functionsで検索するときは0xをとる)
  - Destination -> emotet-2.dll: 0x1001a094 (Version Tracking Functionsで検索するときは0xをとる)
2. 共通する“特徴的な”命令列を特定
3. 特徴的な命令列を含む関数一覧をスクリプトで取得

extract\_c2\_stage2\_not\_impl.pyのfind\_config\_addrss

```
42 def find_config_address():
43     ''' **** IMPLEMENT HERE ****
44
45     search config embedded address and return address of it.
46     Hint:
47     | 1. use Version Tracking to configure the characteristic
48     |   instructions related to config
49     | 2. use `findBytes` method to search bytes/instructions
50     '''
51     raise NotImplementedError('not implemented')
```

# Decrypt Strings Stage2: Find Function

- Version Trackingで複数サンプル内の命令列を比較
- 共通する特徴的な命令列を特定する

たとえばこのあたり

1000e267	89	72 28	MOV	dword ptr [EDX + 0x28], ESI=>DAT_1001f000	1001a1d9	89	58 28	MOV	dword ptr [EAX + 0x28], EBX=>DAT_1001f200
1000e26a	89	72 24	MOV	dword ptr [EDX + 0x24], ESI=>DAT_1001f000	1001a1dc	89	58 08	MOV	dword ptr [EAX + 0x8], EBX=>DAT_1001f200
1000e26d	89	4a 10	MOV	dword ptr [EDX + 0x10], ECX	1001a1df	89	78 3c	MOV	dword ptr [EAX + 0x3c], EDI
1000e270	eb	04	JMP	LAB_1000e276	→ 1001a1e2	eb	04	JMP	LAB_1001a1e8

デコンパイル結果

```
*(undefined **) ((int)pvVar1 + 0x28) = &DAT_1001f000;
*(undefined **) ((int)pvVar1 + 0x24) = &DAT_1001f000;
*(undefined4 *) ((int)pvVar1 + 0x10) = 0;
while (*(int *)(&DAT_1001f000 + iVar2 * 8) != 0) {
```

# Decrypt Strings Stage2: Find Function

- 特徴的な命令列を検索し、その命令の引数に渡されているアドレスを取得
  - `Address[] findBytes(Address start, java.lang.String byteString, int matchLimit)`
    - 正規表現をつかった検索が可能
  - `Instruction getInstructionAt(Address address)`
    - 引数addressで指定したアドレスの命令を取得

1000e267	89 72 28	MOV	dword ptr [EDX + 0x28], ESI=>DAT_1001f000
1000e26a	89 72 24	MOV	dword ptr [EDX + 0x24], ESI=>DAT_1001f000
1000e26d	89 4a 10	MOV	dword ptr [EDX + 0x10], ECX
1000e270	eb 04	JMP	LAB_1000e276

```

Python - Interpreter
>>> asm = b'\\x89.{1}\\x28\\x89.{2}\\x89.{2}\\xeb\\x04'
>>> found = findBytes(None, asm, -1)
>>> found
array(ghidra.program.model.address.Address, [1000e267])
>>> inst = getInstructionAt(found[0])
>>> config_addr = inst.getAddress(1)
>>> config_addr
1001f000
  
```

レジスタや変数など、コンパイラ等によって可変なものは「.」（任意の一文字）にして一般化する

# Decrypt Strings Stage2: Fully Automated

find\_config\_addressを実装したextract\_c2\_stage2.pyとして保存して実行

```
def find_config_address():
    asm = b'\\x89.{1}\\x28\\x89.{2}\\x89.{2}\\xeb\\x04'
    found = findBytes(None, asm, -1)
    if not found:
        raise RuntimeError('Config not found')
    elif len(found) != 1:
        raise ValueError('False detection of config')

    inst = getInstructionAt(found[0])
    config_addr = inst.getAddress(1)
    return config_addr
```

emotet.dllおよびemotet-2.dllで自動抽出が可能に

```
Console - Scripting
extract_c2_stage2.py> Running...
[*] Config Address: 1001f000
http://[redacted]:80
http://[redacted]:180:443
http://[redacted]:141:8080
http://[redacted]:2.9:7080
http://[redacted]:.87:443
http://[redacted]:17:443
http://[redacted]:202:80
http://[redacted]:15:8080
http://[redacted]:243:80
http://[redacted]:.94:8080
http://[redacted]::80
http://[redacted]:8:443
http://[redacted]:.180:7080
http://[redacted]:182:80
http://[redacted]:9:8080
```





# Summary

# Ghidra Script for EMOTET

---

1. APIハッシュの突合
  - `search_hash.py`
2. 文字列の復号
  - `decrypt_string_stage1.py`
3. C&C情報の抽出
  - `extract_c2_stage1.py`
  - `extract_c2_stage2.py`

# (Advanced) It's Automatic!

- `extract_c2_stage2.py`をHeadless対応した`analyzer.py`をHeadless Analyzerで実行すればバッチ処理も可能



Headless AnalyzerでCLI経由で`analyzer.py`を実行

```
cmd
C:\Ghidra>%GHIDRA_INSTALL_DIR%\support\analyzeHeadless.bat C:\Ghidra ghidra_project -process
emotet.dll -scriptPath C:\Ghidra\ghidra_scripts\ -postScript analyzer.py result.json
```

出力された`result.json` (抜粋)

```
Ghidra > {} results.json > [ ] url
1  {
2    "url": [
3      "http://75.211:80",
4      "http://69.214:8080",
5      "http://67.243:8080",
6      "http://190.126:80",
7      "http://183.5.193:80",
8      "http://137.107:8080",
9      "http://168.138:7080",
10     "http://209.232:80",
11     "http://180.110:8080",
12     "http://167.214:8080",
13     "http://85.138:80",
14     "http://95.64:8080",
15     "http://83.251:8080",
16     "http://157.39:8080",
17     "http://79.9:8080",
18     "http://70.250:8080",
19     "http://93.71:8080",
20     "http://174.180:80",
21     "http://24.119:80",
22     "http://41.34:80",
23     "http://188.2:443",
24     "http://91.52:7080",
```

# Conclusion

---

- 解析自体を目的とするのではなく、自動化を目的にするとスケールする
- 解析コードの共有により、知識をサイロ化させない・再利用可能にする
  - 解析結果だけでなく、再利用可能なコードとして共有する (Analysis as a Code)
  - Ghidra Scriptは誰でも再利用可能なので、有効な選択肢の一つ
  - 解析者にもGithubificationが広がると、コミュニティ全体の能力が向上する
    - The Githubification of InfoSec: <https://medium.com/@johnlatwc/the-githubification-of-infosec-afbdbfaad1d1>

**THANKS!**

---

**ALLSAFE**

if you have any, please let us know, friend.

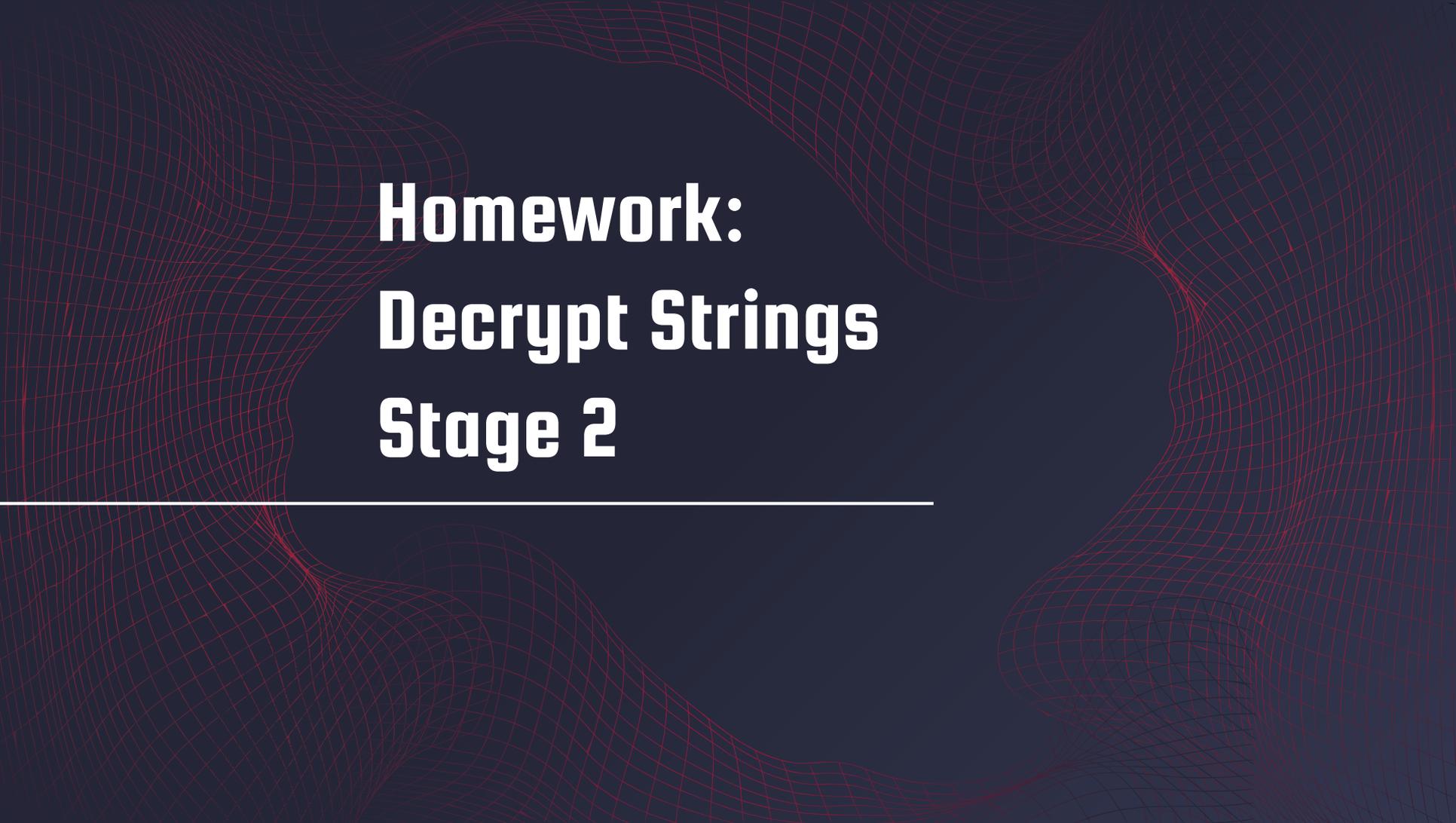
contact: @PINKSAWTOOTH

# CREDITS

- Ghidra Logo by NSA
- Presentation template by Slidesgo
- Icons by Flaticon
- Twemoji by Twitter, Inc and other contributors is licensed under CC-BY 4.0
- Allsafe Logo inspired by Mr.Robot



# Appendix



**Homework:**  
**Decrypt Strings**  
**Stage 2**

---

# Decrypt Strings Stage2

---

- 復号関数のアドレスは常に固定とは限らない 🤔
- 復号関数を都度手動で解析して特定しては自動化の意味がない 🤔



複数サンプルを解析し、  
復号関数内の共通する特徴的な命令列を見つけ出し、  
それをもとに復号関数を自動特定する

# Decrypt Strings Stage2: Find Function

decrypt\_string\_stage2\_not\_impl.pyのfind\_decrypt\_string\_funcを実装する

1. Version Trackingで複数サンプルの命令列を比較
  - Source -> emotet.dll: 0x1000732d (Version Tracking Functionsで検索するときは0xをとる)
  - Destination -> emotet-2.dll: 0x10006aba (Version Tracking Functionsで検索するときは0xをとる)
2. 共通する“特徴的な”命令列を特定
3. 特徴的な命令列を含む関数一覧をスクリプトで取得

decrypt\_string\_stage2\_not\_impl.pyのfind\_decrypt\_string\_func

```
32 def find_decrypt_string_func():
33     ''' **** IMPLEMENT HERE ****
34         return list of possible decrypt_string functions
35         Hint:
36             1. use Version Tracking to compare two samples
37             2. use `findBytes` to search instructions/binaries
38             3. use `getFunctionContaining` to get function that contains specific address
39     '''
40     raise NotImplementedError('not implemented')
```

# Decrypt Strings Stage2: Find Function

- Version Trackingで複数サンプル内の命令列を比較
- 共通する特徴的な命令列を特定する

たとえばこのあたり

Decompile View	Listing View
Source: <b>FUN_1000732d()</b> in /emotet.dll	Destination: <b>FUN_10006aba()</b> in /emotet-2.dll
10007410 8b d3 MOV data,EBX	10006b9c 8b d3 MOV EDX,EBX
10007412 8d 0c b7 LEA param_1,[EDI + ESI*0x4]	10006b9e 8d 0c b7 LEA ECX,[EDI + ESI*0x4]
10007415 8b f1 MOV ESI,param_1	10006ba1 8b f1 MOV ESI,ECX
10007417 2b f7 SUB ESI,EDI	10006ba3 2b f7 SUB ESI,EDI
10007419 83 c6 03 ADD ESI,0x3	10006ba5 83 c6 03 ADD ESI,0x3
1000741c c1 ee 02 SHR ESI,0x2	10006ba8 c1 ee 02 SHR ESI,0x2
1000741f 3b f9 CMP EDI,param_1	10006bab 3b f9 CMP EDI,ECX
10007421 0f 47 f0 CMOVA ESI,buffer	10006bad 0f 47 f0 CMOVA ESI,EAX

デコンパイル結果

```
puVar1 = (uint *) ((int)enc_data + (size & 0xffffffff));  
size = (uint) ((int)puVar1 + (3 - (int)enc_data)) >> 2;
```

# Decrypt Strings Stage2: Find Function

- 特徴的な命令列を検索し、それらの命令列を含んでいる関数一覧を取得
  - `Address[] findBytes(Address start, java.lang.String byteString, int matchLimit)`
    - 正規表現をつかった検索が可能
  - `Function getFunctionContaining(Address address)`
    - 指定したアドレスを含む関数を取得

```
Source: FUN_1000732d() in /emotet.dll
10007410 8b d3      MOV     data,EBX
10007412 8d 0c b7   LEA    param_1,[EDI + ESI*0x4]
10007415 8b f1      MOV     ESI,param_1
10007417 2b f7      SUB     ESI,EDI
10007419 83 c6 03   ADD     ESI,0x3
1000741c c1 ee 02   SHR     ESI,0x2
1000741f 3b f9      CMP     EDI,param_1
10007421 0f 47 f0   CMOVA  ESI,buffer
```

Python - Interpreter

```
>>> asm = b'\x8b.{1}\x2b.{1}\x83.{1}\x03\xc1.{1}\x02'
>>> found = findBytes(None, asm, -1)
>>> found
array(ghidra.program.model.address.Address, [1000704b, 10007415, 1000d219])
>>> suspicious_decrypt_string_func = set([getFunctionContaining(sus_inst) for sus_inst in found])
>>> suspicious_decrypt_string_func
set([FUN_1000732d, FUN_1000d126, FUN_10006f31])
```

レジスタや変数など、コンパイラ等によって  
可変なものは「.」（任意の一文字）にして一般化する

# Decrypt Strings Stage2: Fully Automated

実装したスクリプトをdecrypt\_string\_stage2.pyとして保存して実行

```

32 def find_decrypt_string_func():
33     asm = b'\x8b.{1}\x2b.{1}\x83.{1}\x03\xc1.{1}\x02'
34     found = findBytes(None, asm, -1)
35     if not found:
36         raise RuntimeError('decrypt_string function is not found')
37     suspicious_decrypt_string_func = set([getFunctionContaining(sus_inst) for sus_inst in found])
38     return list(suspicious_decrypt_string_func)
39
40 def main():
41     decrypt_string_funcs = find_decrypt_string_func()
42     for decrypt_string_func in decrypt_string_funcs:
43         print('[*] decrypt_string at {}'.format(decrypt_string_func.getEntryPoint()))
44         # get callee address of decrypt_string function
45         for xref in getReferencesTo(decrypt_string_func.getEntryPoint()):
46             # get instructions before callee address
47             insts = get_instructions_before(xref.getFromAddress(), 3)
48
49             # find instruction that passes
50             # address of encrypted data via ECX
51             for inst in insts:
52                 if is_mov_edx(inst):
53                     # get encrypted data address and decrypt it
54                     data_addr = inst.getOpObjects(1)[0].getValue()
55                     if data_addr:
56                         decrypted_str = decrypt_string(toAddr(data_addr))
57                         # add comment
58                         print('[*] found at {}: {!r}'.format(inst.getAddress(), decrypted_str))
59                         add_bookmark_comment(inst.getAddress(), decrypted_str)

```

複数の復号関数を見つけ、  
引数に渡されている文字列を復号している

```

Console - Scripting
decrypt_string_stage2.py> Running...
[*] decrypt_string at 1000732d
[*] found at 1000282a: '%s\\rundll32.exe "%s\\%s",Control_RunDLL'
[*] found at 10004103: '%s\\%s'
[*] found at 10005938: '%s\\%s'
[*] found at 10008d16: '%u.%u.%u.%u'
[*] found at 10008e88: 'DNT: 0\r\nReferer: %s/%s\r\nContent-Type: multipart/form-data; boundary=%s\r\n'
[*] found at 1000ddb9: 'SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run'
[*] found at 1001050e: 'POST'
[*] found at 100136d7: '%s\\rundll32.exe "%s\\%s",Control_RunDLL'
[*] found at 10014329: '%s\\%s'
[*] found at 1000f2e6: '%s\\rundll32.exe "%s",Control_RunDLL %s'
[*] found at 1001c33e: '%s\\%s%x'
[*] found at 1001c3c0: '%s\\%s'
[*] found at 100156aa: 'WinSta0\\Default'
[*] found at 10004971: '%s\\%s'

```



# **How to get unpacked samples by yourself**

---

# How to get unpacked samples by yourself

---

- 必要なもの

- 仮想環境 (VMWare/VirtualBox/Hyper-V等)
- Windows OS (ゲストOS)
- ANY.RUNアカウント
- Process Explorer / Process Monitor (なくても可)
- Hollow Hunter

- [https://github.com/hasherezade/hollows\\_hunter/release](https://github.com/hasherezade/hollows_hunter/release)

- 以後の作業は全て構築した仮想環境内でおこなう

# How to get unpacked samples by yourself

以下ANY.RUNのページを開き、実行されているDLLをダウンロード

- urcwzowo.xck (=packed emotet.dll) MD5:19b0124f2e4f223113bb11a84765a6c3
  - <https://app.any.run/tasks/4060b18e-8132-42c0-a0bc-65f398d6bfb2/>
- eclh.gfk (=packed emotet-2.dll) MD5:714cdae2b20896e72d92e28dc831b81b
  - <https://app.any.run/tasks/b9be3b5e-368d-4248-87c2-e06b3d66769e/>

The screenshot shows the ANY.RUN interface. At the top, there's a 'Files modification' table with columns: Timeshift, PID, Process name, Filename, and Content. A red box highlights the 'Download' button in the top right corner of the interface. Below the table, a detailed view of the file 'urcwzowo.xck' is shown, including its status ('Dropped from process'), a 'Download' button, and a 'Hashes' section with MD5, SHA1, SHA256, and SSDEEP hashes.

Timeshift	PID	Process name	Filename	Content
3532 ms	1908	powershell.exe	YS1PV9.temp	Not available
3532 ms	1908	powershell.exe	C:\Users\admin\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations\d93f411851d7c929.customDestinations-ms~RF154e74.TMP	7.83 Kb binary
3532 ms	1908	powershell.exe	C:\Users\admin\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations\d93f411851d7c929.customDestinations-ms	7.83 Kb binary
6297 ms	1908	powershell.exe	C:\Users\admin\PhBovbt\Y3va_u0\V64L.dll	Not available
6563 ms	3528	rundll32.exe	C:\Users\admin\AppData\Local\Ozspuecmz\urcwzowo.xck	197 Kb executable

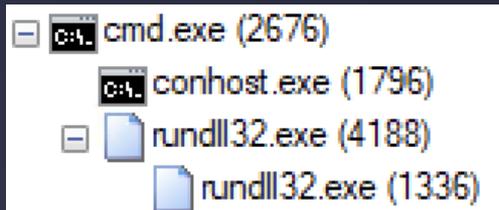
<p>urcwzowo.xck</p> <p>⚠ Dropped from process</p> <p>🔍 Look up on VirusTotal</p>		<p>Submit to analysis</p> <p>Download</p> <p>Mime: application/x-dosexec</p> <p>Size: 197.00 Kb</p>
<p>TrID - File Identifier</p> <p>76.4% Win64 Executable (generic)</p> <p>12.4% Win32 Executable (generic)</p> <p>5.5% DOS Executable Generic</p>		<p>Hashes</p> <p>MD5 19B0124F2E4F223113BB11A84765A6C3</p> <p>SHA1 D27BFE2481C74FE0C213456AD3906E96097AB4C6</p> <p>SHA256 BF274F8C9BA0A2E9B51CC341688A1BC827E21E3D52F152BF49380123F70B2A59</p> <p>SSDEEP 3072:7zr1NwFbuQ+12ro9Ux4huw/mY2EeTyDqqsAX8QaCQIS39mLSnwK1:7zPkBvor0GIRe+7sAXMCQL3ImwK</p>

# How to get unpacked samples by yourself

仮想環境を外部接続不可な状態にし、EMOTETを適切な引数で実行

```
C:\Users\john>rundll32.exe C:\Users\john\Desktop\urcwzowo.xck,Control_RunDLL
```

このとき、Process ExplorerやProcess Hackerなどで実行されたrundll32.exeのプロセスシーケンスを確認しておく



# How to get unpacked samples by yourself

実行後5-10秒ほど待機し、子プロセスのPIDをHollows Hunterに渡して実行

- 成功すると実行フォルダ配下にprocess\_<PID>フォルダが作成され、中にアンパック後のEMOTETのDLLが保存される

```
C:\Users\john\Desktop>C:\Users\john\Desktop\hollows_hunter.exe /pid 1336
HollowsHunter v.0.2.6 (x64)
Built on: Apr 13 2020

using: PE-sieve v.0.2.6.0
>> Scanning PID: 1336 (rundll32.exe)
[-] Invalid address of relocations
[-] Invalid address of relocations
>> Detected: 1336
-----
SUMMARY:
Scan at: 01/13/21 03:29:17 (1610476157)
Finished scan in: 3516 milliseconds
[+] Total Suspicious: 1
[+] List of suspicious:
[0]: PID: 1336, Name: rundll32.exe
```