YARA Pretty Darby: Hunt the Spider Like a Champ

ALLSAFE

> whoami

ALLSAFE

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。
- 国内外のカンファレンスで発表経験あり。
- セキュリティ・キャンプやJSACでワークショップを実施。
- 技術系同人サークルAllsafeのプロデューサー。

Kohei Kawabata

- トレンドマイクロ株式会社にて、サイバークライムや標的型攻撃で利用されるマルウェアの解析に加え、脆弱性の研究に従事。
- 技術系同人サークルAllsafeの先輩。

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、スレッ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）。

TABLE OF CONTENTS

Announcement

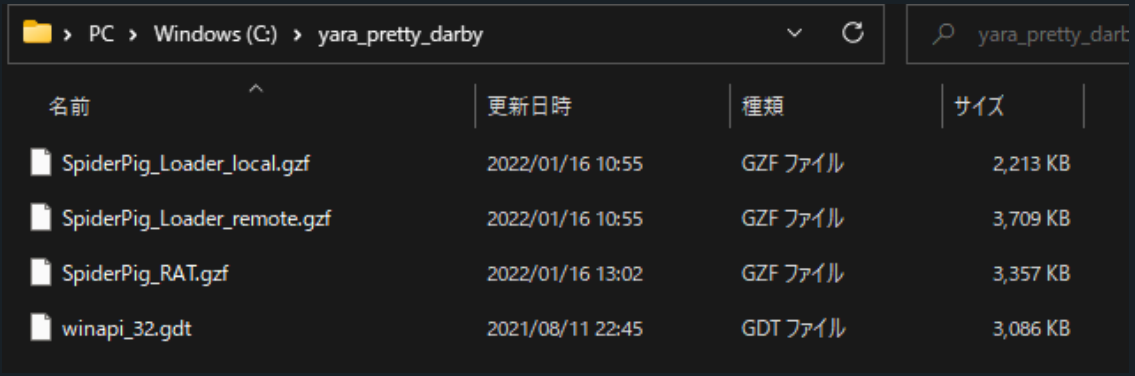
講義で使用するサンプルについて

- マルウェアのコードを含むため、指定した環境以外では取り扱わないこと
  - ワークショップ終了後は削除すること
- ワークショップ内で指定したファイル以外は実行はしないこと
- 第三者および外部ウェブサービス（VirusTotalなど）に、
  提供・アップロードしないこと

Setup

1. SlackからzipファイルをDL
2. 当日に配布したパスワードで下記パスにzipの中身を展開

   C:¥yara_pretty_darby



1

Introduction YARA

YARA

マルウェアを検知、分類するためのOSSツール

- シグネチャ（ルール）を定義し、合致するパターンを検出
- ファイルやメモリなど、さまざまな種類のデータに対して検索可能

A l l s a f e
“41 6c 6c 73 61 66 65”
が含まれるか？（=ルール）

スキャン

```
          00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F
0B6A0:  50 36 00 40 01 00 00 00  10 AC 00 40 01 00 00 00  P6.@.......@....
0B6B0:  00 80 00 40 01 00 00 00  00 00 00 00 00 00 00 00  ...@............
0B6C0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0B6D0:  41 6C 6C 73 61 66 65 00  48 65 6C 6C 6F 2C 20 66  Allsafe.Hello, f
0B6E0:  72 69 65 6E 64 21 00 00  00 00 00 00 00 00 00 00  riend!..........
0B6F0:  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
```

YARA for InfoSec

具体的にどのような場面で使用するのか？

- Analysis
  - 類似ファイルをみつけてファミリを特定する
  - マルウェア内で使われている手法（暗号方式・パッカーなど）を特定する
- Detection
  - IR中に見つけた不正ファイルと類似するファイル・プロセスを検索する
  - 特定の通信先やファイルパスが含まれるファイル・プロセスを検索する
- Hunting
  - 類似ファイルが外部サービス（VirusTotalなど）で公開されるのを監視する

YARA in Tools

- サンドボックス、EDR
  - e.g. Cuckoo Sandbox, CAPE
  - 多くの商用製品がYARAルールに対応している
- メモリフォレンジックツール
  - e.g. Volatility

Basics of YARA

YARA, rule them all

YARAルールは、独自の構文で記述される

- 文字列やバイナリパターンなどのシグネチャ (stringsセクション)と、それらの組み合わせ条件式 (conditionセクション)をもとに、検出ロジックを組み立てる

ルール名

- 検出対象の名前（自由記述）
- マルウェア名やテクニック名など

シグネチャ

- 検索したいデータを定義
- $<任意の変数名> = <パターン>
- 文字列や Hexを指定可能
- 正規表現も利用可能

```
rule silent_banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

メタデータ

- ルールに関するメタ情報をキーバリュー形式で記述
- ルール作成者や作成日など
- 完全自由記述

条件式

- 検出のための条件式を記述
- stringsセクションで定義したデータの組み合わせ条件を指定

strings section

stringsセクションで指定可能なさまざまなパターン

- HEX文字列（{}でくくる）
  - ワイルドカード（?）
  - 可変調データ表現（[n-m]）
- テキスト文字列（""でくくる）
  - エンコーディング（ascii/wide）
  - ケースインセンシティブ（nocase）
- 正規表現（//でくくる）

```
rule strings_example
{
    strings:
        // "?"はワイルドカード
        // "?"1つにつき、4ビット
        $hex_string_1 = { E2 34 ?? C8 A? FB }

        // 合致させたいデータの長さが可変の場合
        $hex_string_2 = { F4 23 [4-6] 62 B4 }

        // 正規表現
        $regex_string = /Hello, \w+!/

        // エンコーディングを指定可能（ascii/wide）
        $text_string_1 = "Allsafe" wide

        // 大文字/小文字を区別して扱うかも指定可能
        // nocaseの場合、以下のようなケースも一致すると判断される
        //  - HELLO, World!
        //  - Hello, worLD!
        $text_string_2 = "hello, word!" nocase
    condition:
        all of them
}
```

condition section

conditionセクションで指定可能なさまざまなパターン

- 比較式
  - <=, >=, <, >, ==, !=
- 論理式
  - and, or
- セット
  - any of them, all of them, # of ($*)
- データへのアクセス
  - uint(8|16|32), int(8|16|32), etc
- キーワード
  - filesize

例: MZ/PEヘッダをチェックしてから条件式を評価

```
condition:
    uint16(0) == 0x5A4D and // MZ
    uint32(uint32(0x3c)) == 0x00004550 and // PE
    all of them
```

例: stringセクションで定義した変数名に対して、
ワイルドカードを使用して指定することも可能

```
condition:
    filesize < 50KB and
    $text_string_* and
    $hex_string_* and
    any of ($regex_*)
```

modules

機能拡張のためのモジュールも利用可能

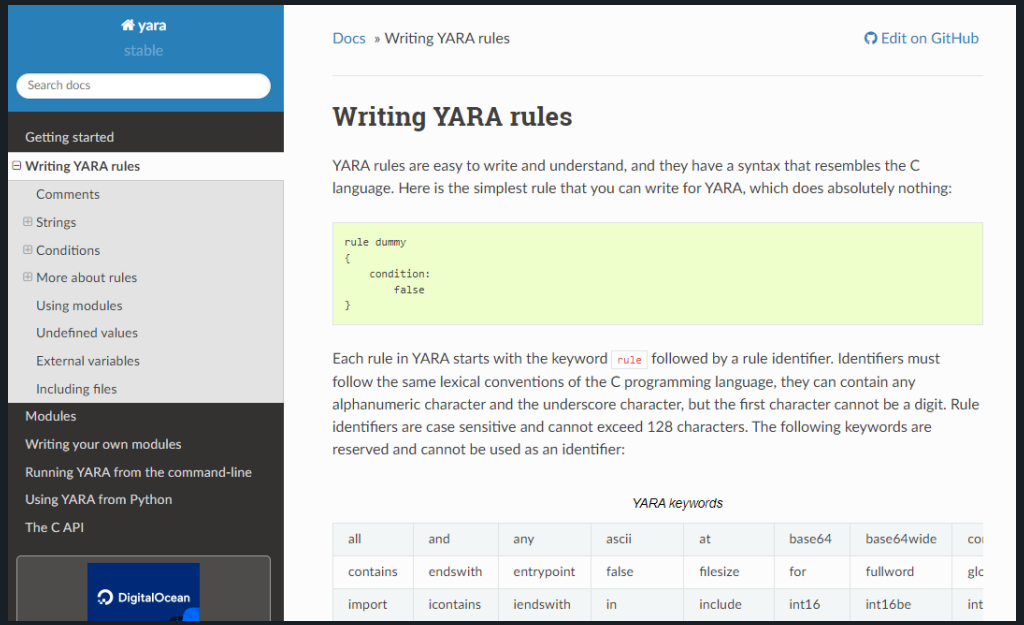
- pe: PEファイルの構造に簡単にアクセスするためのモジュール
- hash: ハッシュ値を計算するためのモジュール
- dotnet: .NETファイルの属性情報にアクセスするためのモジュール

```
import "pe"

rule vnc_dll {
  condition:
    pe.Dll and
    pe.imports("ws_32.dll","socket") and
    pe.exports("VncStartServer") and
    pe.exports("VncStopServer")
}
```

peモジュールを使用し、特定のAPIのインポートなどを検出条件にしている

YARA Docs

- 細かい仕様は公式ドキュメントで確認する
  - https://yara.readthedocs.io/en/stable/writingrules.html



For more detail

プロダクションレベルのルールを読むのが一番

- https://github.com/Yara-Rules/rules
- https://github.com/Neo23x0/signature-base/tree/master/yara
- https://github.com/kevoreilly/CAPEv2/tree/master/data/yara
- https://github.com/eset/malware-ioc/
- https://github.com/JPCERTCC/MalConfScan/blob/master/yara/rule.yara
- https://github.com/InQuest/awesome-yara

2

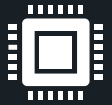
Make YARA rules with Ghidra

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、強力な機能を提供

Ghidra’s basic features

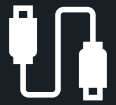


Disassembler/Decompiler

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

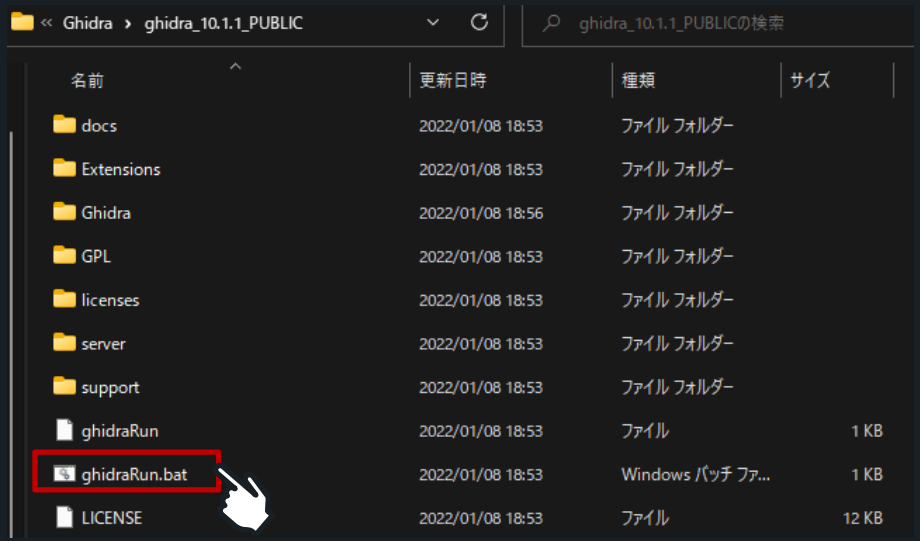
Ghidra Usage

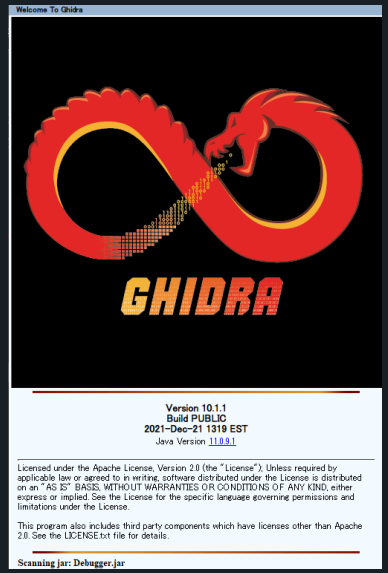
Set up, Ghidra...

- Ghidraのインストール・セットアップについては、事前に共有したURLを参考にして進めてください

Wake up, Ghidra...

C:¥Ghidra¥ghidra_10.1.1_PUBLIC¥ghidraRun.bat 実行





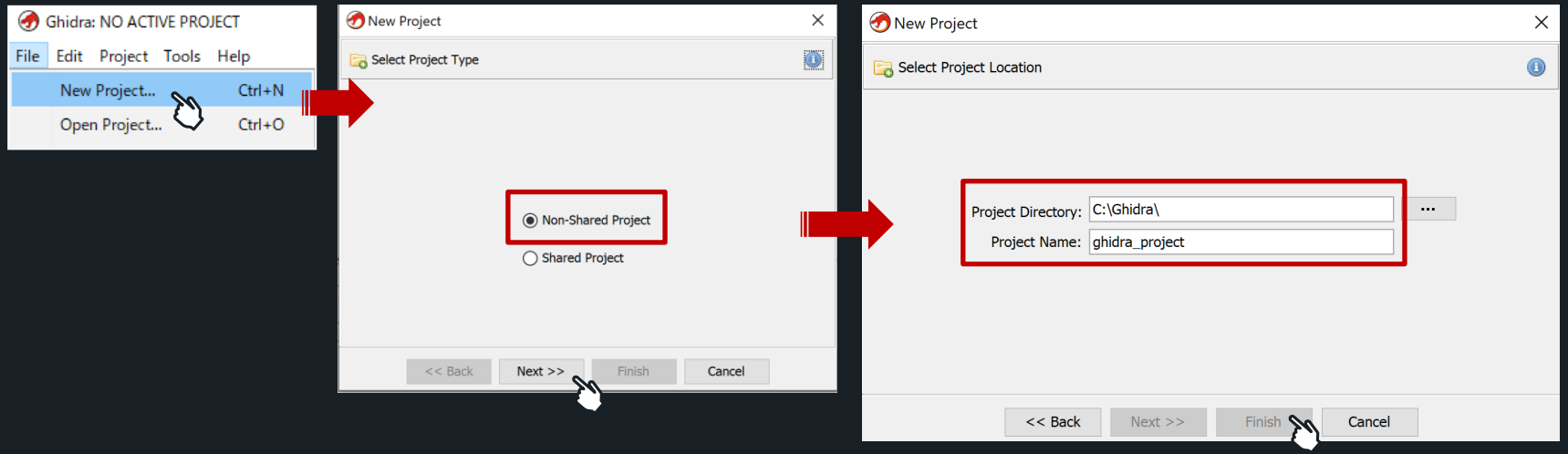
Create Ghidra Project

- Ghidraは解析結果をプロジェクトという単位で管理
- 初回起動時にはプロジェクトを新規作成する必要がある（すでに作成済みであれば不要）

❶ File -> New Project を選択

❷ Non-Shared Projectにチェックボックスが入っていることを確認してNext

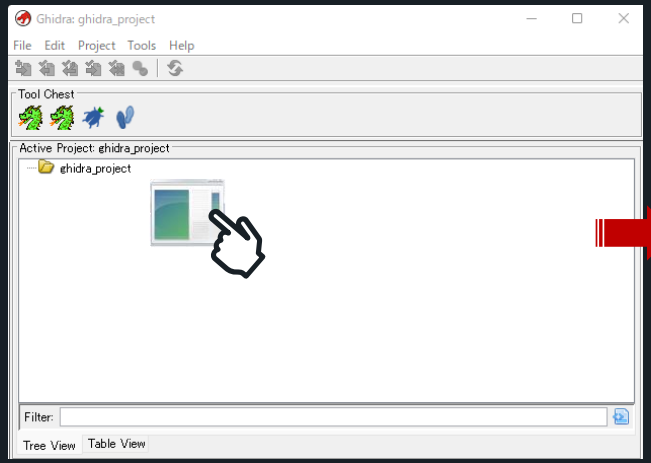
❸ 遷移した先のSelect Project Location画面で、以下設定でプロジェクトの作成（講義ではこのディレクトリをプロジェクトのディレクトリとして扱う）



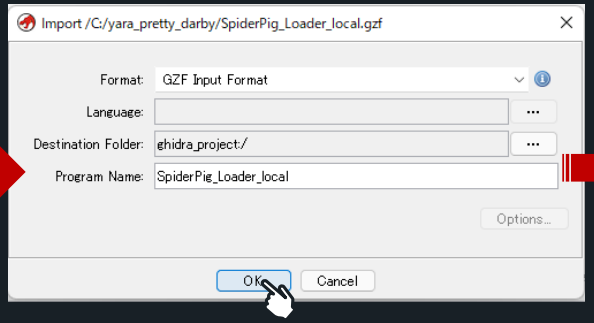
import file

- Ghidraでは、解析対象となるファイルをProjectに「import」して作業する
  - 「C:¥yara_pretty_darby¥*ファイル名*.gzf」を使用

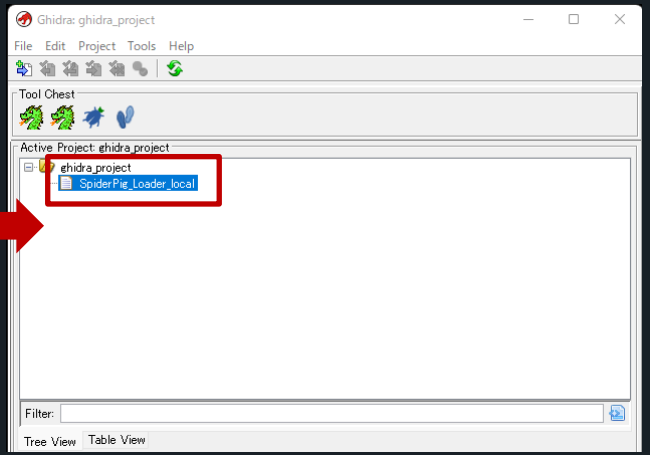
❶ 作成したプロジェクト画面にファイルをD&D



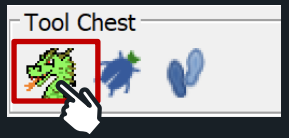
❷ ポップアップが表示されたら「OK」



❸ プロジェクト画面にファイルが表示されていればインポート完了



Code Browser



開く

Listing View

ディスアセンブル結果・データを表示

Decompile View

Listing Viewのデコンパイル結果を表示

Program Tree

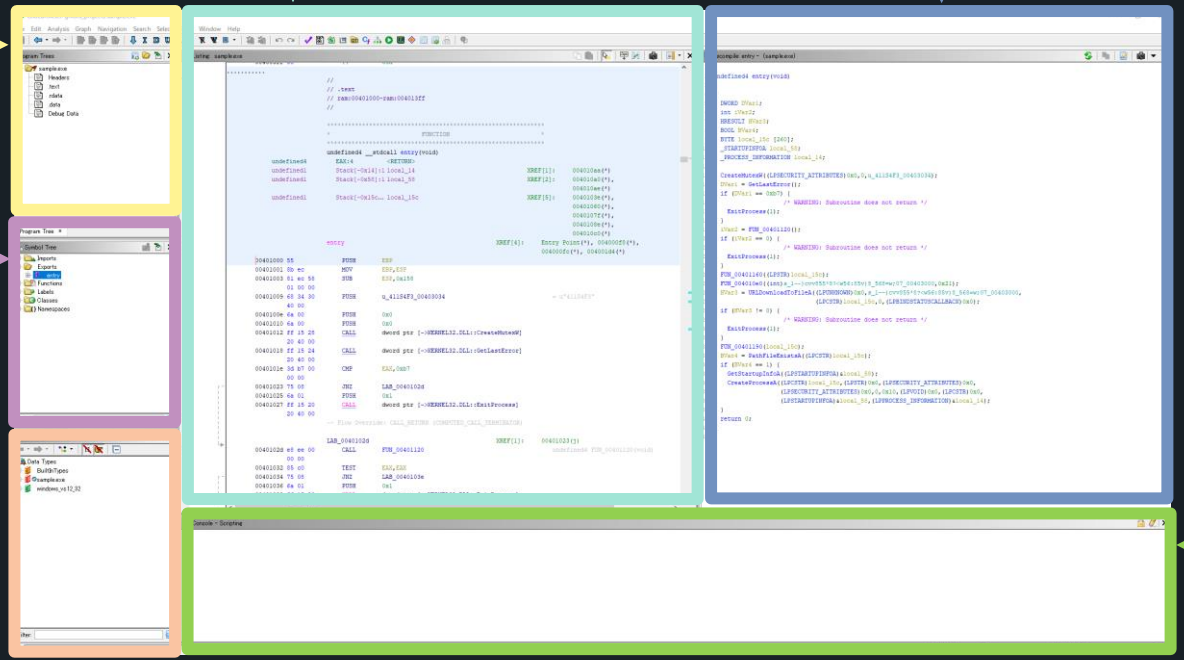
セクション名などを表示

Symbol Tree

インポート/エクスポート関数、関数名、クラス名などなどを列挙

Data Type Manager

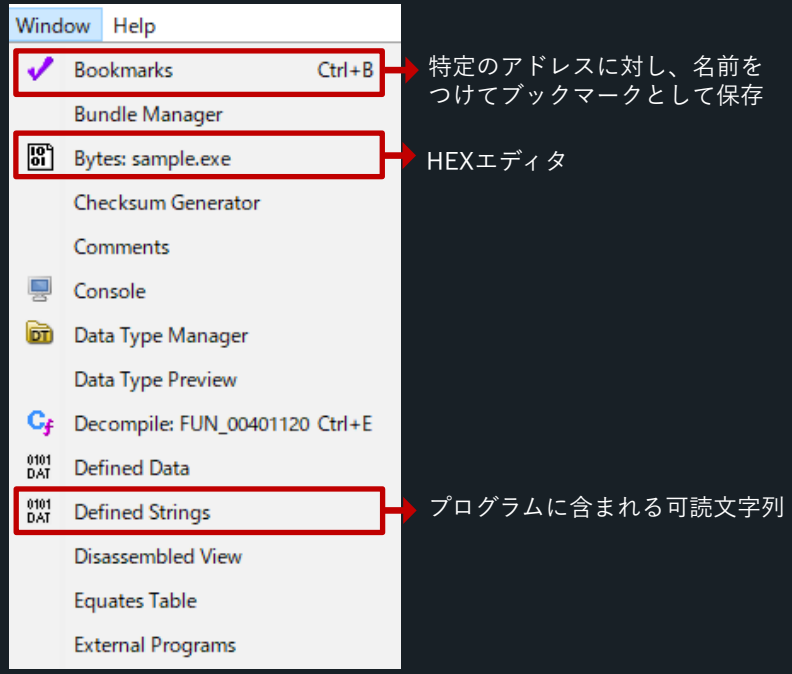
独自構造体などのデータ型を管理

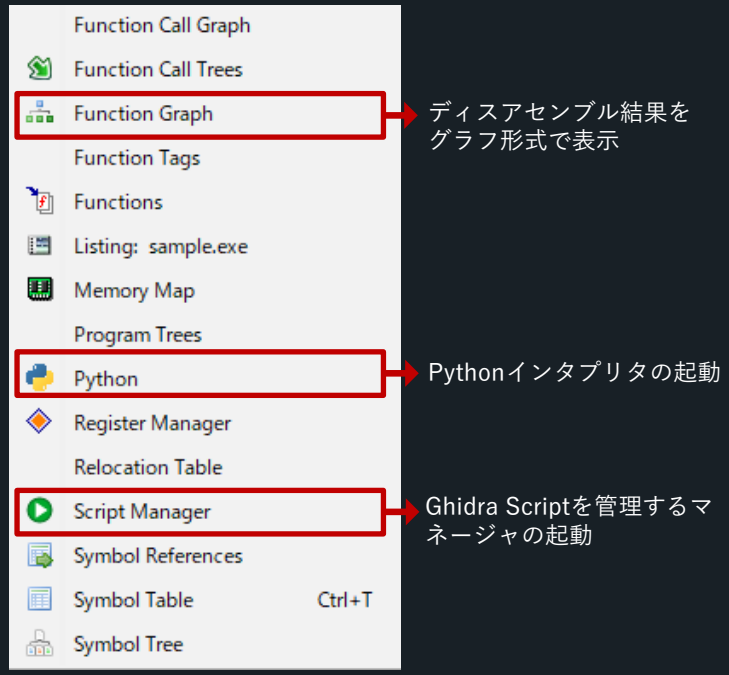


Console

スクリプトの標準出力結果を表示

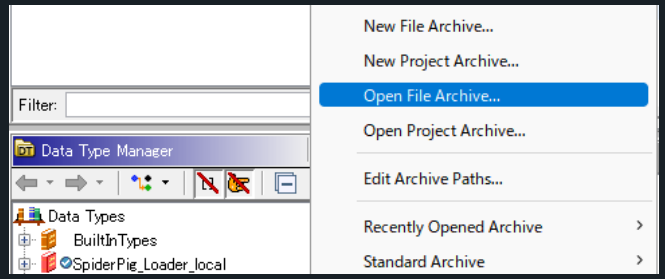
Other Window



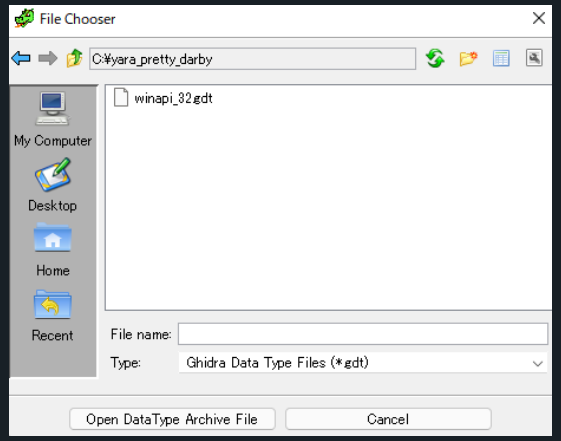


Load gdt

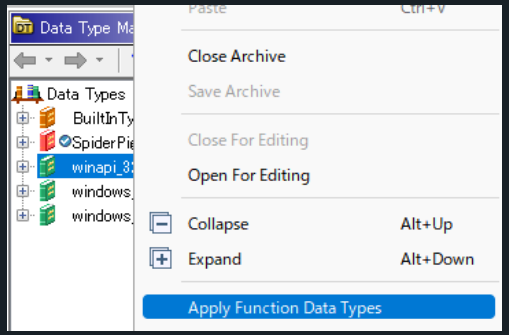
- 解析に必要なAPIの情報を追加する



❶ ▼からOpen File Achiveを選択



❷ winapi_32.gdtを選択



❸ winapi_32を右クリックして、
Apply Function Data Typesを選択

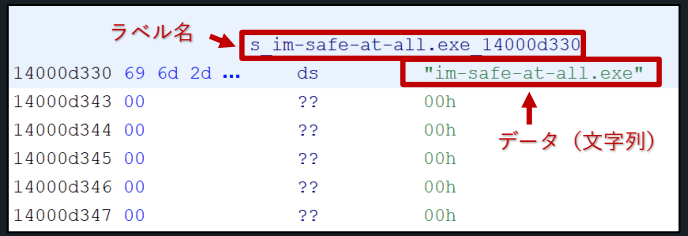
Code Browser: Listing View

ディスアセンブル結果や、データ領域のデータを表示・操作する画面

- Decompile Viewでざっと流れを確認し、細かい処理をListing View で眺める
- あらゆる処理をデコンパイルできるわけではなく、常に正しくデコンパイルできるわけでもないので、Listing Viewに頼るほかない場面も多い

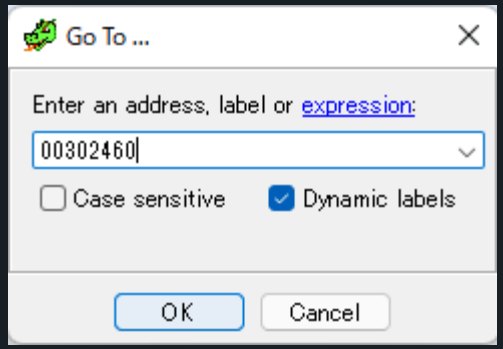
関数のディスアセンブル結果を表示

データを表示



Go to ...

- Gキーを押すとGo to ... ポップアップが表示される
  - 任意のアドレス、ラベルを入力してジャンプできる



Code Browser: Decompile View

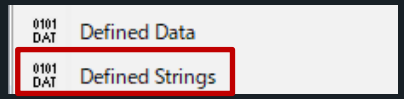
デコンパイル結果を表示する画面

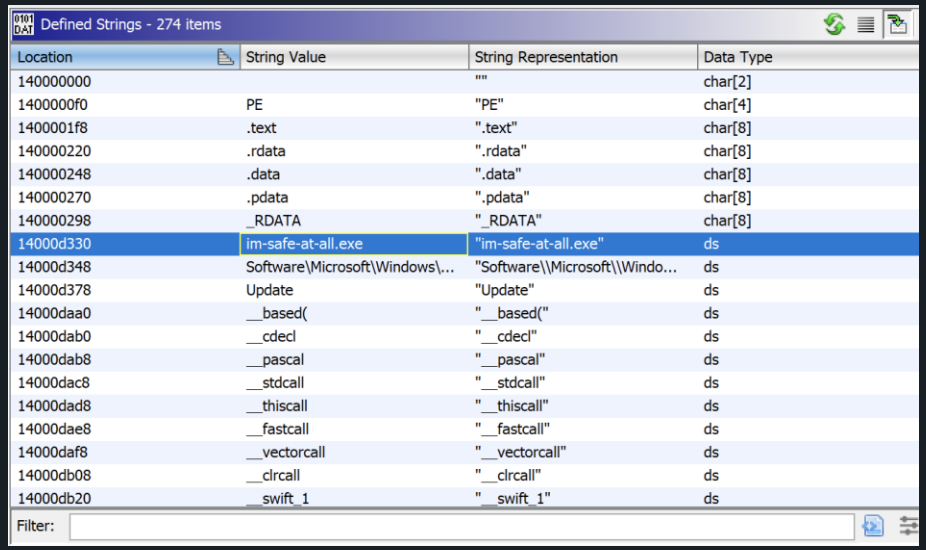
- Ghidraは、機械語をP-CODEという中間表現に変換後、擬似C言語にデコンパイルしている
- 常にデコンパイルできるわけではないし、常にデコンパイル結果が正しいわけではないことに注意
- データは表示できない

Defined Strings

バイナリ内の可読文字列を表示

- 「Window > Defined Strings」から開く
- クリックすると、文字列が定義されているアドレスが Listing Viewに表示される
- コンパイラによって自動で埋め込まれた本来のコードと無関係な文字列も多いので注意
- Ghidraが文字列として認識できていない場合は現れないので注意

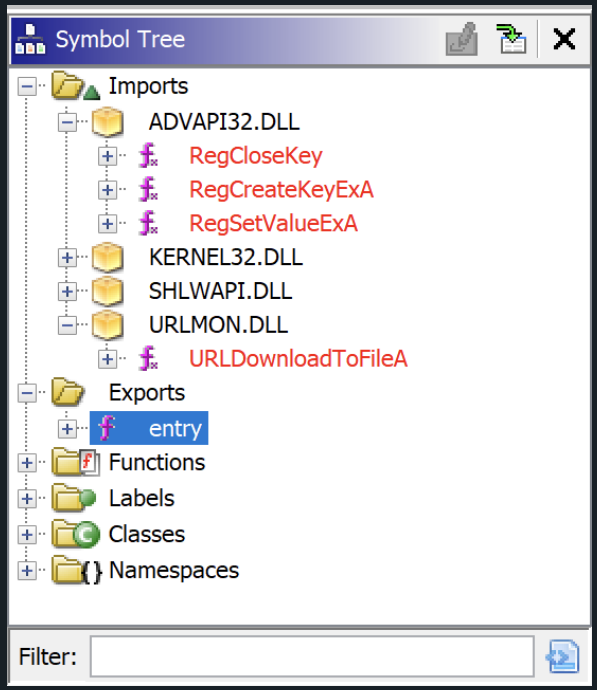




Symbol Tree

シンボル情報をカテゴライズして表示

- Imports
  - ImportしているAPI一覧
  - APIベースで解析を進める際に参照
- Exports
  - Exportされている関数一覧
  - entryはプログラムのエントリーポイント
  - DLLの場合は、Export関数がみえる



Make YARA rule

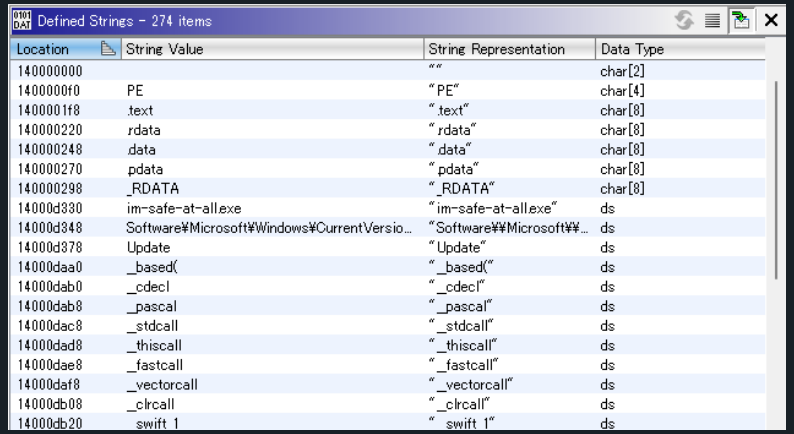
meta Section

- metaセクションを記述
  - author: 作成者の情報を記述
  - created_date: 作成日
  - modified_date: 編集日
  - hash*n*: ルールの作成に使ったファイルのハッシュ
  - description: 検出ルールの説明

```
rule Allsafe_downloader {
    meta:
        author = "Allsafe"
        created_date = "2022-01-28"
        modified_date = "2022-01-28"
        hash1 = "01276c52ee0a725fd83de4685993da11bdc202b6075d94d0d3d5814eab048725"
        description = "detect sample downloader"
```

Step1: strings rule

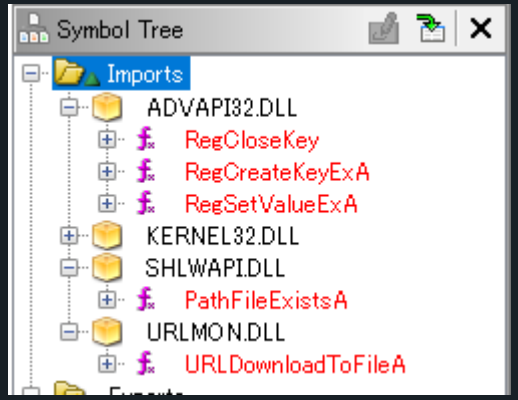
- Defined Stringsウィンドウから特徴的な文字列を選択してルールを作成




```
rule Allsafe_downloader {
    meta:
        author = "Allsafe"
        created_date = "2022-01-28"
        modified_date = "2022-01-28"
        hash1 = "01276c52ee0a725fd83de4685993da11bdc202b6075d94d0d3d5814eab048725"
        description = "detect sample downloader"
    strings:
        $s1 = "im-safe-at-all.exe" ascii
        $s2 = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" ascii
    condition:
        uint16(0) == 0x5a4d and // MZ
        uint32(uint32(0x3c)) == 0x00004550 and //PE
        all of them
}
```

Step2: behavior rule

- Symbol TreeウィンドウのImportsから特徴的な挙動を実現するAPIを選択してルールを作成
- 明示的な解決の場合は文字列やバイト列を利用してルールを作成する



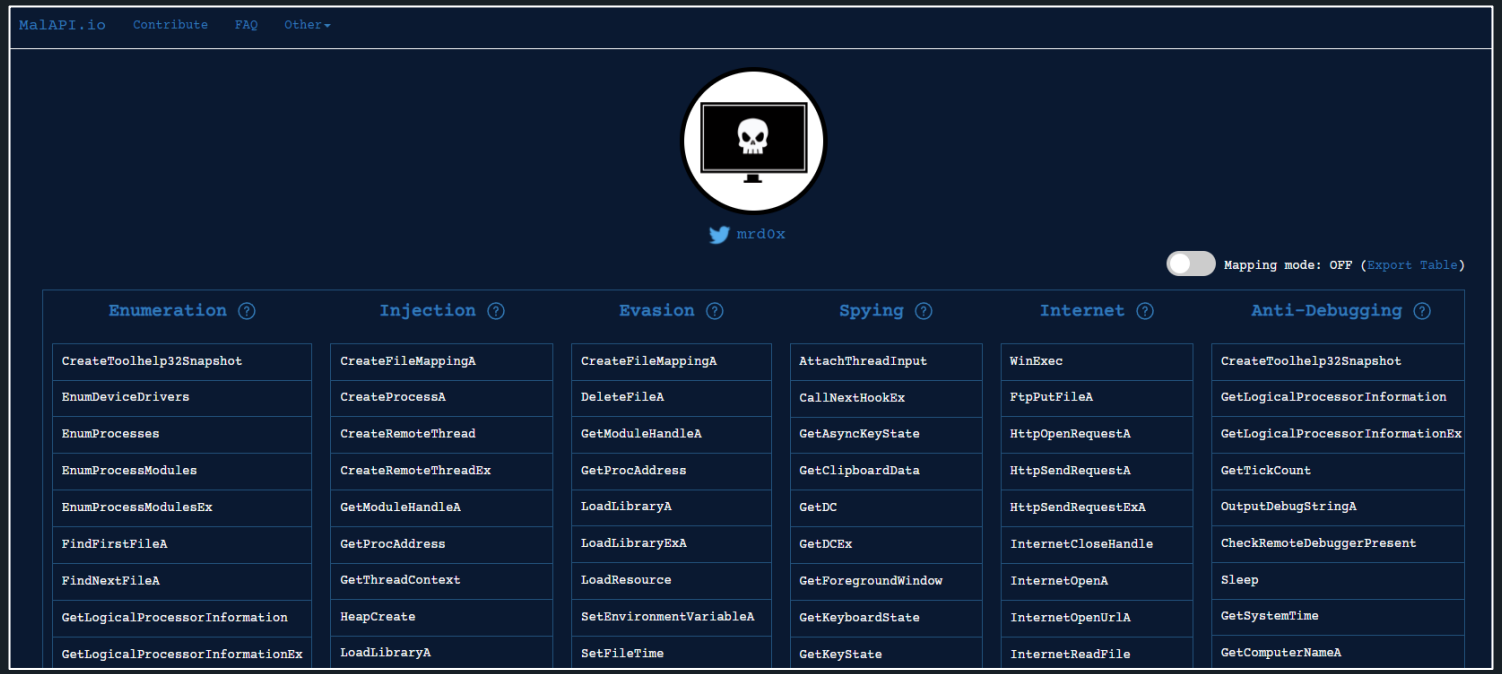

```
import "pe"

rule Allsafe_downloader {
    meta:
        author = "Allsafe"
        created_date = "2022-01-28"
        modified_date = "2022-01-28"
        hash1 = "01276c52ee0a725fd83de4685993da11bdc202b6075d94d0d3d5814eab048725"
        description = "detect sample downloader"
    strings:
        $s1 = "im-safe-at-all.exe" ascii
        $s2 = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" ascii
    condition:
        uint16(0) == 0x5a4d and // MZ
        uint32(uint32(0x3c)) == 0x00004550 and //PE
        pe.imports("urlmon.dll", "URLDownloadToFileA") and
        pe.imports("kernel32.dll", "CreateProcessA") and
        all of them
}
```

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
    - 実行時にAPI名を復元してGetProcAddressの引数に渡すことで難読化
  - APIハッシュを使ったAPIのアドレス取得手法
    - JSAC2021の資料で解説

API List

- https://malapi.io/ を参考にマルウェアが利用するAPIを調査する



Step3: effective rule

Stay Hard & Stay Flexible

変更しづらいコードをみつける

- 文字列は変更しやすい
- マルウェア固有のアルゴリズムやコードは変更しづらい

可変なコードは可変なままで

- アドレスなどコンパイラによって可変なものはワイルドカードにする
- 定数など攻撃者が変更する可能性がある値もワイルドカードにする

Effective rule Sample

- 一部の定数のみが異なる、2種類の文字列の復号ルーチン

```
lVar1 = 0;
do {
  *param_1 = *param_1 + 10;
  *param_1 = *param_1 ^ 0x1d;
  *param_1 = *param_1 - 3;
  param_1 = param_1 + 1;
  lVar1 = lVar1 + 1;
} while (lVar1 != param_2);
return;
}
```

```
80 06 0a        ADD     byte ptr [RSI],0xa
80 36 1d        XOR     byte ptr [RSI],0x1d
80 2e 03        SUB     byte ptr [RSI],0x3
48 ff c6        INC     RSI
48 ff c7        INC     RDI
48 3b fa        CMP     RDI,param_2
75 ec           JNZ     LAB_140001b17
5f              POP     RDI
c3              RET
```

```
lVar1 = 0;
do {
  *param_1 = *param_1 + 0xd;
  *param_1 = *param_1 ^ 0x1e;
  *param_1 = *param_1 - 2;
  param_1 = param_1 + 1;
  lVar1 = lVar1 + 1;
} while (lVar1 != param_2);
return;
}
```

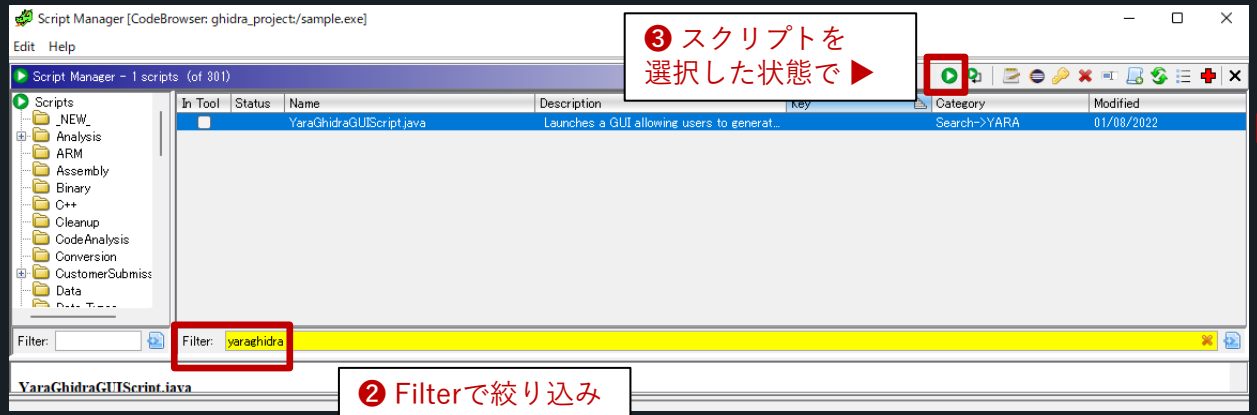
```
80 06 0d        ADD     byte ptr [RSI],0xd
80 36 1e        XOR     byte ptr [RSI],0x1e
80 2e 02        SUB     byte ptr [RSI],0x2
48 ff c6        INC     RSI
48 ff c7        INC     RDI
48 3b fa        CMP     RDI,RDX
75 ec           JNZ     LAB_140001b17
5f              POP     RDI
c3              RET
```

定数やレジスタなどが
変わっても検出可能な
パターン

```
strings:
    $custom_xor = {
        80 06 ??
        80 36 ??
        80 2e ??
        4? ff c6
        4? ff c7
        4? 3b fa
        75 ??
    }
```

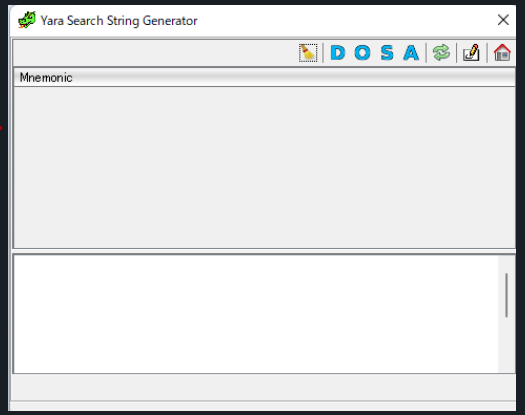
Tool: YaraGhidraGUIScript

❶ Ghidra Scriptを
管理・実行するためのScript Managerを開く



❸ スクリプトを
選択した状態で ▶

❷ Filterで絞り込み

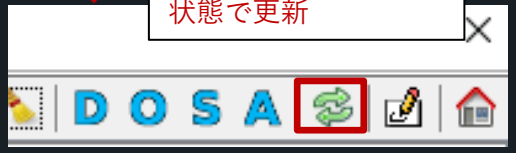


Tool: YaraGhidraGUIScript Usage 1

```
                    FUN_140001b10

140001b10 57
140001b11 48 8b f1
140001b14 48 33 ff          XOR         RDI,RDI

                    LAB_140001b17
140001b17 80 06 0a          ADD         byte ptr [RSI],0xa
140001b1a 80 36 1d          XOR         byte ptr [RSI],0x1d
140001b1d 80 2e 03          SUB         byte ptr [RSI],0x3
140001b20 48 ff c6          INC         RSI
140001b23 48 ff c7          INC         RDI
140001b26 48 3b fa          CMP         RDI,param_2
140001b29 75 ec             JNZ         LAB_140001b17
140001b2b 5f                POP         RDI
140001b2c c3                RET
```

❶ パターンを作成したい箇所を選択

❷ コードを選択した状態で更新



❸ Yaraルールを GUIでぽちぽち編集可能

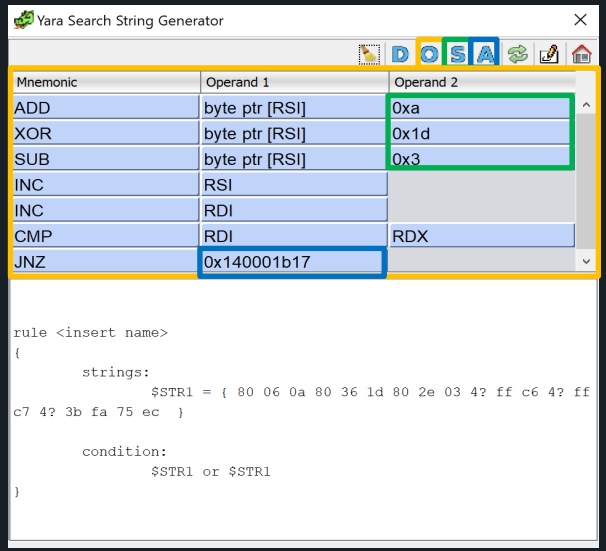
Yara Search String Generator

Mnemonic	Operand 1	Operand 2
ADD	byte ptr [RSI]	0xa
XOR	byte ptr [RSI]	0x1d
SUB	byte ptr [RSI]	0x3
INC	RSI	
INC	RDI	
CMP	RDI	RDX
JNZ	0x140001b17	

```
rule <insert name>
{
        strings:
                $STR1 = { 80 06 0a 80 36 1d 80 2e 03 4? ff c6 4? ff
c7 4? 3b fa 75 ec  }

        condition:
                $STR1 or $STR1
}
```

Tool: YaraGhidraGUIScript Usage 2

- 命令列、データをYARAルール用のHEX文字列に自動変換
- 手動でマスク（=ワイルドカードに置き換え）、もしくはバイト列の属性に応じて自動マスク




元に戻す（アンマスク）


データ（命令以外）をマスク


オペランド（引数）をマスク


スカラー（実数）をマスク

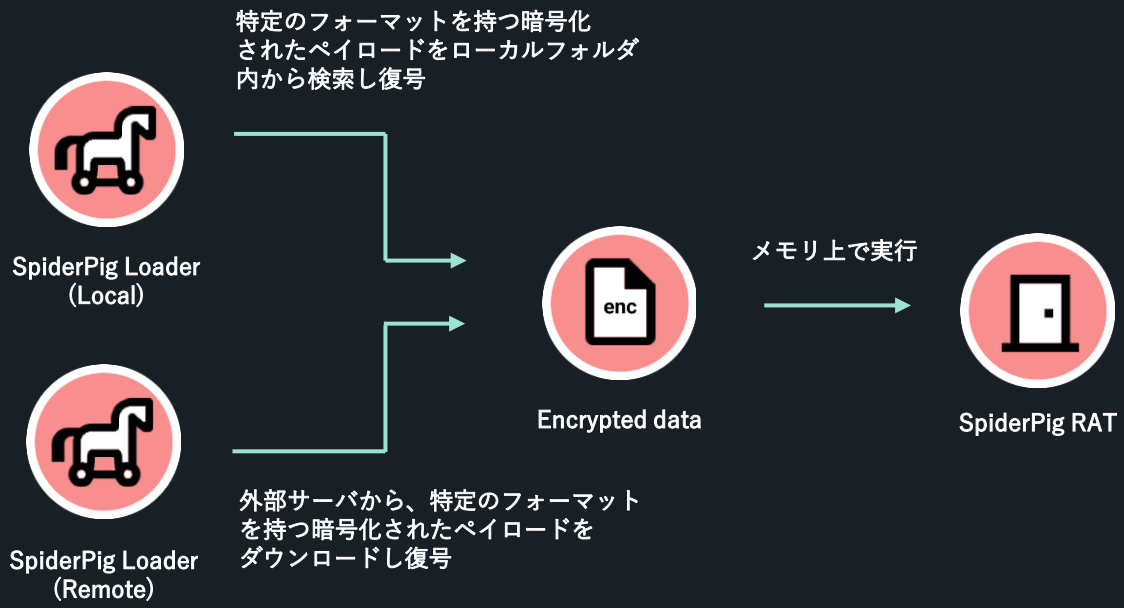

アドレスをマスク

3

Anatomy of SpiderPig Family

Overview of SpiderPig Family

- BlackTechが利用するPEローダとカスタムバックドア



SpiderPig Loader (a.k.a. selfmake Loader)

- オンメモリPEローダ、OSSのソースコードをカスタムして悪用している
  - https://github.com/abhisek/Pe-Loader-Sample/blob/master/src/PeLdr.cpp

SpiderPig Loaderのデバッグメッセージ

Decompile: FUN_00301cf0 - (SpiderPig_Loader_local)

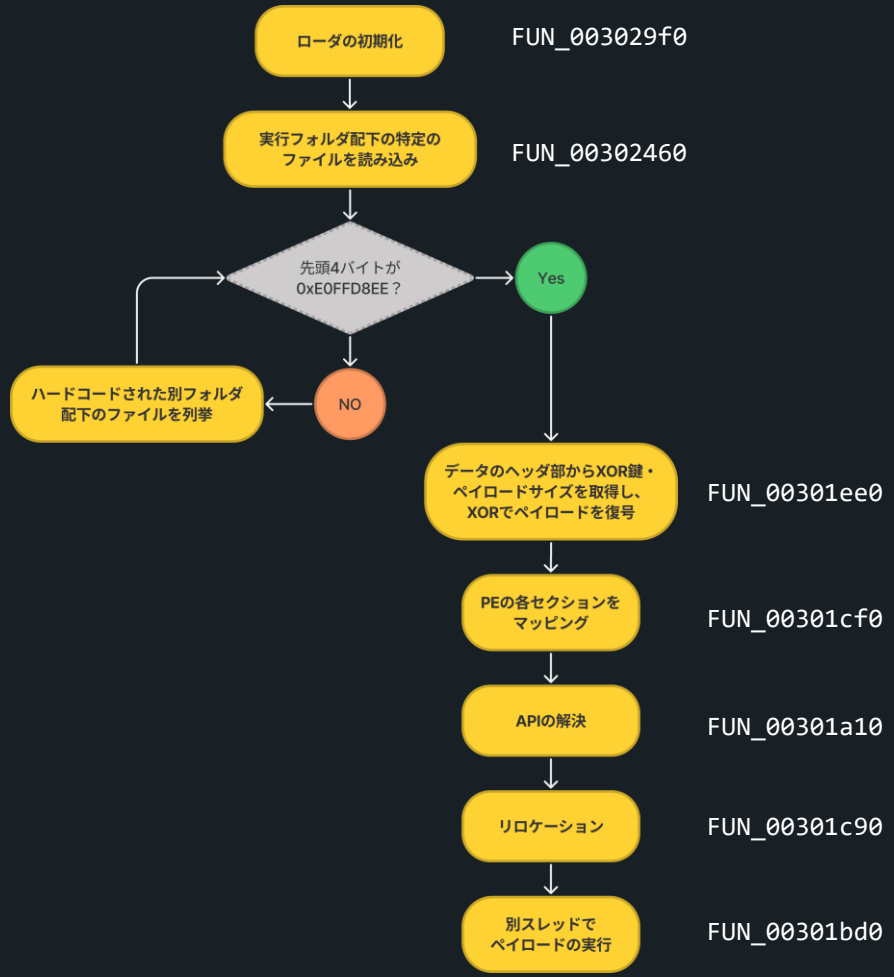
```
26  pcVar12 = "[+] Mapping Target PE File\n";
27  ppuVar2 = FUN_00311465();
28  _fprintf((FILE *)(ppuVar2 + 0x10),pcVar12);
29  uVar13 = *(undefined4 *)(unaff_ESI + 0x2c);
30  uVar10 = *(undefined4 *)(unaff_ESI + 0x28);
31  pcVar12 = "[+] Loader Base Orig: 0x%08x New: 0x%08x\n";
32  ppuVar2 = FUN_00311465();
33  _fprintf((FILE *)(ppuVar2 + 0x10),pcVar12,uVar10,uVar13);
34  uVar8 = *(uint *)(*(int *)(unaff_ESI + 8) + 0x34);
35  uVar1 = *(uint *)(unaff_ESI + 0x28);
36  uVar7 = *(int *)(*(int *)(unaff_ESI + 8) + 0x50) + uVar8;
37  pcVar12 = "[+] PE Base Orig: 0x%08x END: 0x%08x\n";
```

PeLdr.cppのデバッグメッセージ

```
320 static
321 BOOL PeLdrMapImage(PE_LDR_PARAM *pe)
322 {
323         DWORD                                   i;
324         MEMORY_BASIC_INFORMATION        mi;
325         PIMAGE_SECTION_HEADER           pSectionHeader;
326         BOOL                                    ret = FALSE;
327
328         NTSTATUS        (NTAPI *NtUnmapViewOfSection)
329                                         (HANDLE, LPVOID) = NULL;
330         if(!pe)
331                 return ret;
332
333         DMSG("Mapping Target PE File");
334         DMSG("Loader Base Orig: 0x%08x New: 0x%08x",
335                 pe->dwLoaderBase, pe->dwLoaderRelocatedBase);
```

SpiderPig Loader: Local

- ローカルから暗号化されたペイロードを検索するタイプ
  - gzf: SpiderPig_Loader_local.gzf
  - SHA256: 8bdfc1ed5bfec964050a42a0f1ddd8709fcf14fab1ede151c5a7161be904cd96
- 大まかな実行フロー →



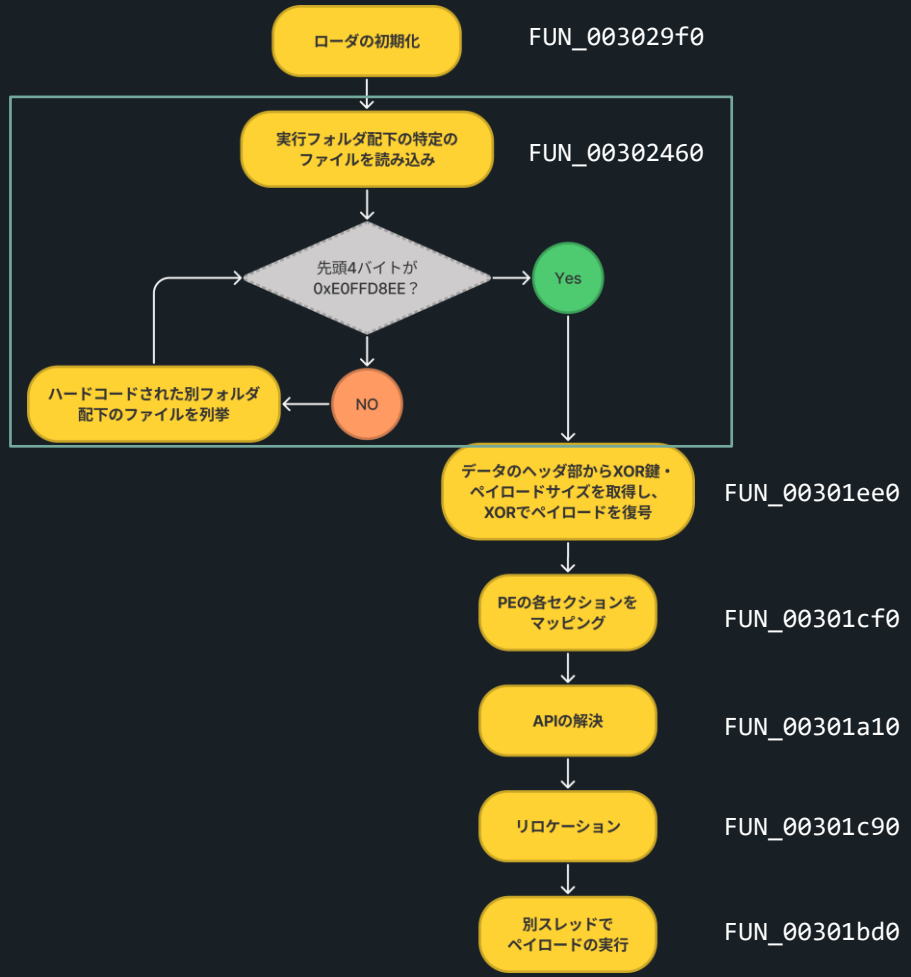
SpiderPig Loader: Local

実行ファイル名から拡張子を削除
FUN_00302410に渡す

```
96      _memset(local_11c,0,0x104);
97      DVar4 = GetModuleFileNameA((HMODULE)0x0,local_11c,0x104);
98      local_11c[DVar4 - 4] = '\0';
99      iVar5 = FUN_00302410(local_11c);
```

```
2   undefined4 FUN_00302410(char *param_1)
3
4   {
5     FILE *_File;
6     unsigned_int local_4;
7
8     _File = _fopen(param_1,"rb");
9     local_4 = 0;
10    if (_File != (FILE *)0x0) {
11      _fread(&local_4,1,4,_File);
12      _fclose(_File);
13      if (local_4 == 0xe0ffd8ee) {
14        return 1;
15      }
16    }
17    return 0;
18  }
```

先頭4バイトを読み込んで0xe0ffd8eeと比較



SpiderPig Loader: Local

```
if (param_1->is_load_from_buffer == 0) {
  in_EDX = extraout_EDX;
  if ((char *)param_1->target_filepath == (char *)0x0) goto LAB_00302192;
  _File = _fopen((char *)param_1->target_filepath,"rb");
  local_3c = _File;
  _fread(&signature,4,1,_File);
  i = extraout_EDX_00;
  if ((((signature == 0xe0ffd8ee) &&
       (sVar4 = _fread(&xor_key,4,1,_File), i = extraout_EDX_01, sVar4 != 0)) &&
      (sVar4 = _fread(&size_of_payload,4,1,_File), i = extraout_EDX_02, sVar4 != 0)) &&
     (sVar4 = _fread(_reserved,0x14,1,_File), i = extraout_EDX_03, sVar4 != 0)) {
    p_payload = (uint *)VirtualAlloc((LPVOID)0x0,size_of_payload + 1,0x3000,4);
    sVar4 = _fread(p_payload,1,size_of_payload,_File);
    i = extraout_EDX_04;
    if (sVar4 == size_of_payload) {
      i = 0;
      p_payload_ = p_payload;
      if (0 < (int)(size_of_payload + ((int)size_of_payload >> 0x1f & 3U)) >> 2) {
        do {
          *p_payload_ = *p_payload_ ^ xor_key;
          i = i + 1;
          p_payload_ = p_payload_ + 1;
        } while (i < (int)(size_of_payload + ((int)size_of_payload >> 0x1f & 3U)) >> 2);
      }
```

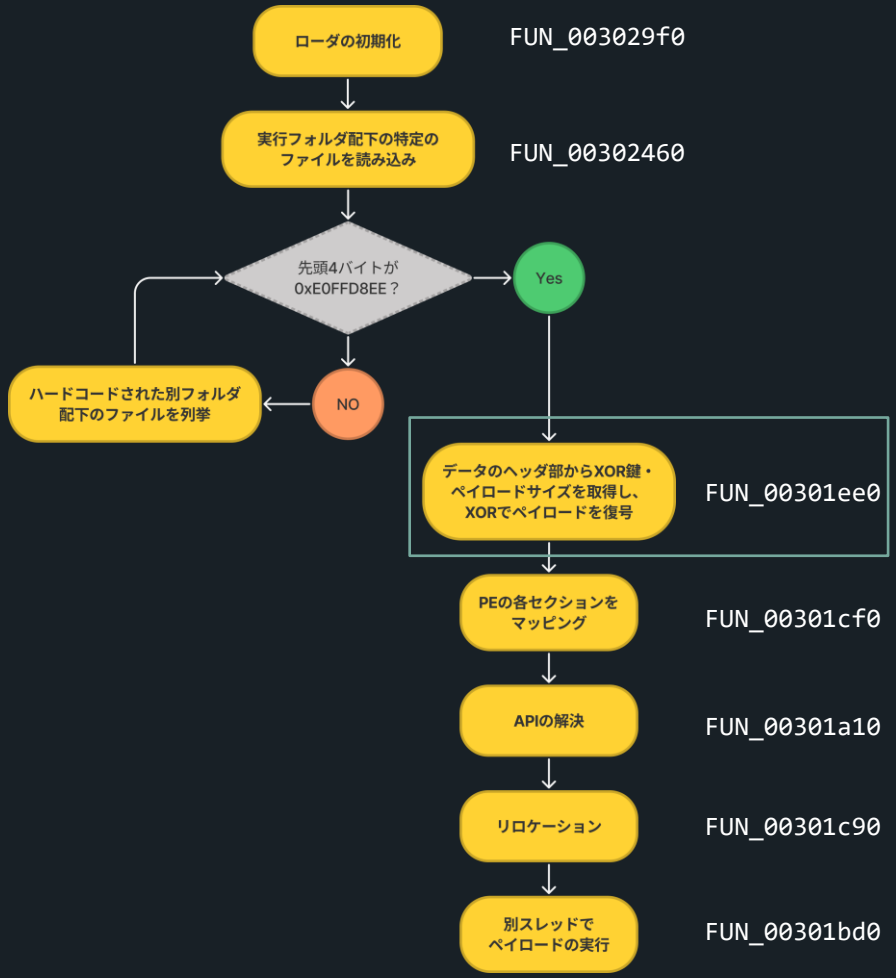
先頭4バイトのシグネチャをチェック

XOR鍵取得

ペイロードのサイズ取得

暗号化されたペイロード部分を読み込み

XORで復号



SpiderPig Loader: Remote

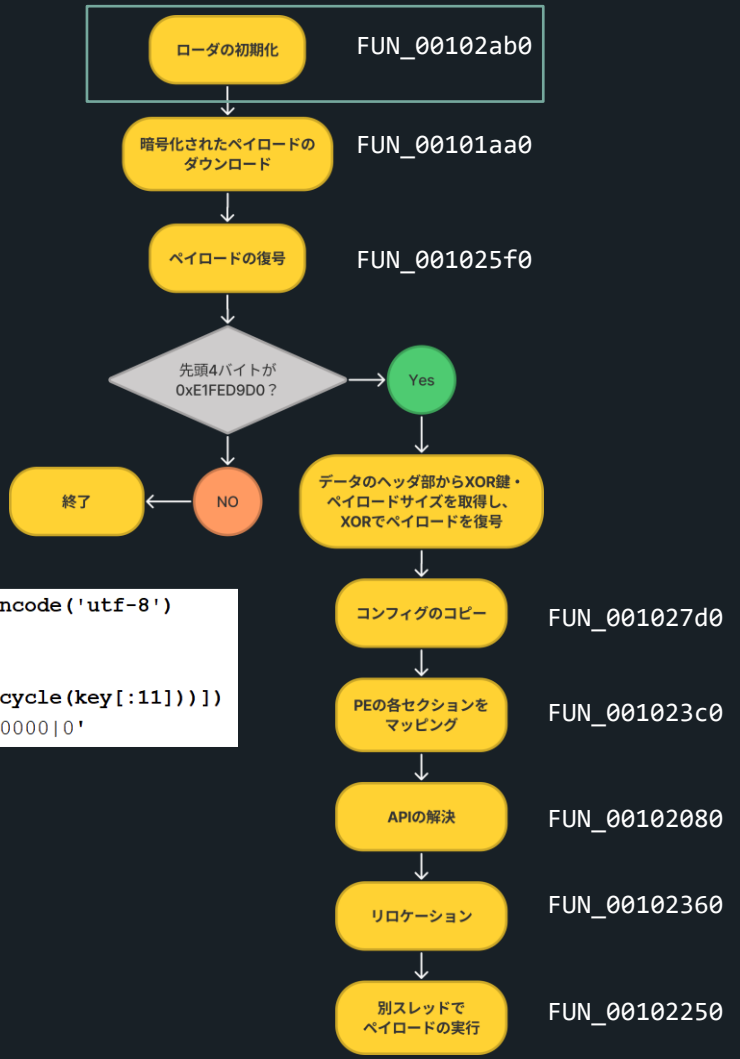
- 外部サーバから暗号化されたペイロードをダウンロードするタイプ
  - gzf: SpiderPig_Loader_remote.gzf
  - SHA256: 7da969010a55919aa66ed97a2d2d6d6a0be3d8dc6151eeb6cebc15e4f06d4553
- 大まかな実行フロー →

SpiderPig Loader: Remote

通信先情報を含むコンフィグのデコード

```
if (0 < iVar6) {
  do {
    pbVar1 = (byte *)(s_efxjj964528428425_00159e20 + iVar18);
    iVar18 = iVar18 + 1;
    (&DAT_0015d0a0)[iVar17] = (&DAT_00159ea4)[iVar17] ^ *pbVar1;
    FUN_00101a10();
    if (10 < iVar18) {
      iVar18 = 0;
    }
    iVar17 = iVar17 + 1;
    uVar8 = extraout_EDX_01;
  } while (iVar17 < iVar6);
}
```

```
>>> key = getDataAt(toAddr(0x159e20)).getValue().encode('utf-8')
>>> size = getInt(toAddr(0x159ea0))
>>> enc = getBytes(toAddr(0x159ea4), size)
>>> ''.join([chr(e ^ ord(k)) for e, k in zip(enc, cycle(key[:11]))])
'45.117.102.243|443|104.168.213.95|443||||||1500|30000|0'
```



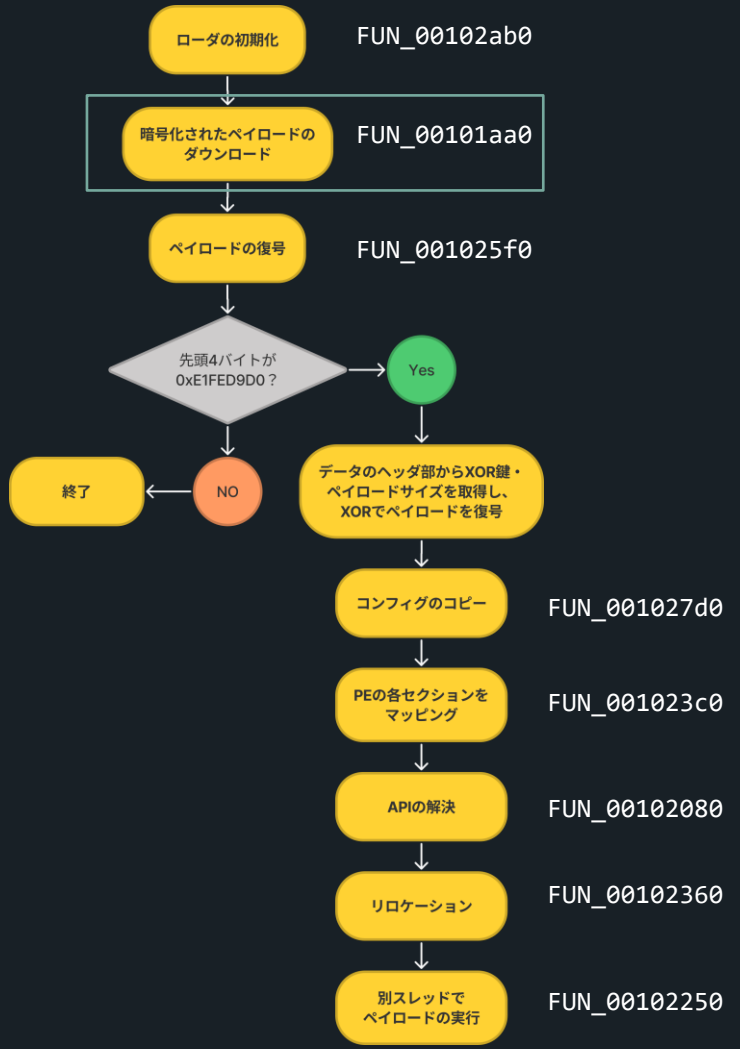
SpiderPig Loader: Remote

FUN_00101aa0に通信先と保存先を渡してダウンロード

```
cVar4 = FUN_00101aa0(local_478,(char *)&hostname_1,"/pfxg.bin",tmp_path);
uVar14 = extraout_ECX_13;
uVar15 = extraout_EDX_11;
if (cVar4 != '\0') break;
cVar4 = FUN_00101aa0(local_47c,(char *)&hostname_2,"/pfxg.bin",tmp_path);
uVar14 = extraout_ECX_14;
uVar15 = extraout_EDX_12;
if (cVar4 != '\0') break;
cVar4 = FUN_00101aa0((int)uVar20,(char *)&hostname_3,"/pfxg.bin",tmp_path);
```

一時ファイルに保存

```
_File = _fopen(param_4,"wb");
if (_File != (FILE *)0x0) {
  _fwrite(&stack0xffaffffc,_Size,1,_File);
  _fclose(_File);
```



SpiderPig Loader: Remote

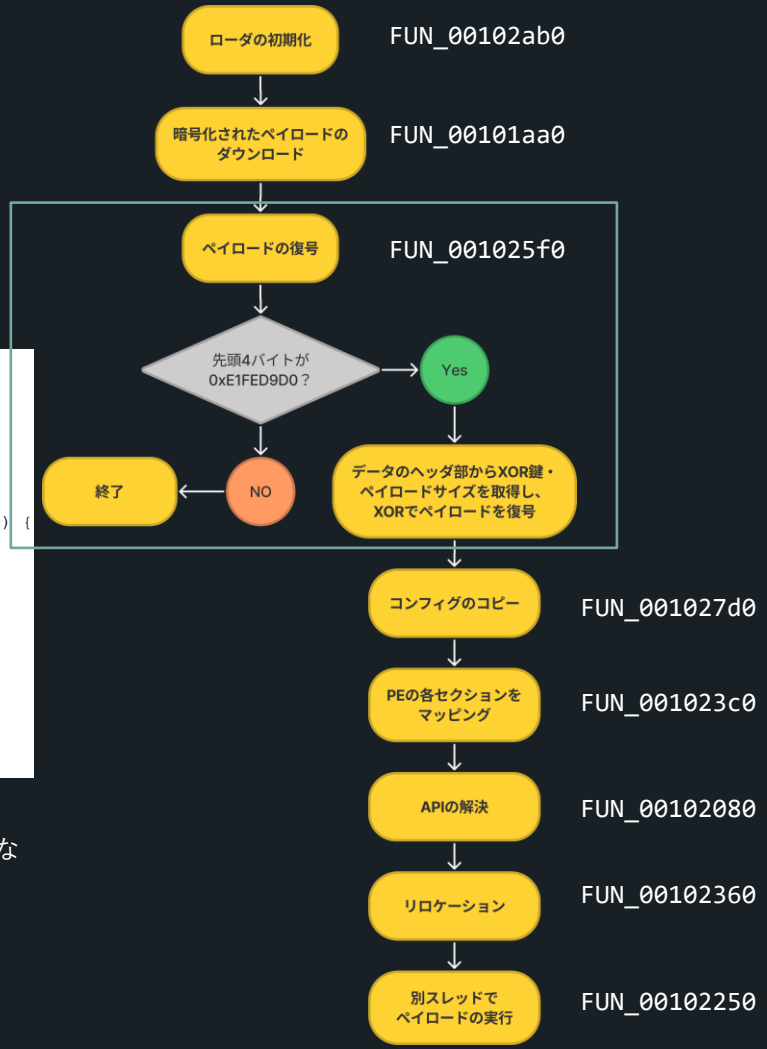
シグネチャのチェック後、ヘッダからXOR鍵やサイズを
取得しペイロードを復号

```
40  _fread(&local_28,4,1,_File);
41  if ((((local_28 == 0xe1fed9d0) && (sVar3 = _fread(&xor_key,4,1,_File), sVar3 != 0)) &&
42      (sVar3 = _fread(&payload_size,4,1,_File), sVar3 != 0)) &&
43     (((sVar3 = _fread(local_24,0x14,1,_File), sVar3 != 0 &&
44       (p_payload = (uint *)VirtualAlloc((LPVOID)0x0,(SIZE_T)(payload_size + 1),0x3000,4),
45       p_payload != (uint *)0x0)) &&
46      (pcVar7 = (char *)_fread(p_payload,1,(size_t)payload_size,_File), pcVar7 == payload_size)))) {
47    iVar6 = 0;
48    puVar5 = p_payload;
49    if (0 < (int)(payload_size + ((int)payload_size >> 0x1f & 3)) >> 2) {
50      do {
51        *puVar5 = *puVar5 ^ xor_key;
52        iVar6 = iVar6 + 1;
53        puVar5 = puVar5 + 1;
54      } while (iVar6 < (int)(payload_size + ((int)payload_size >> 0x1f & 3)) >> 2);
55    }
```

先頭4バイトのシグネチャをチェック
XOR鍵取得
ペイロードのサイズ取得
暗号化されたペイロード部分を読み込み
XORで復号

```
if (*psVar1 == 0x5a4d) {
  piVar4 = (int *)((int)psVar1 + *(int *)(psVar1 + 0x1e));
  unaff_EBX[2] = (char *)piVar4;
  if (*piVar4 == 0x4550) {
    unaff_EBX[3] = (char *)piVar4[0x30];
    local_30 = 1;
    _File = local_34;
    goto LAB_001027aa;
  }
```

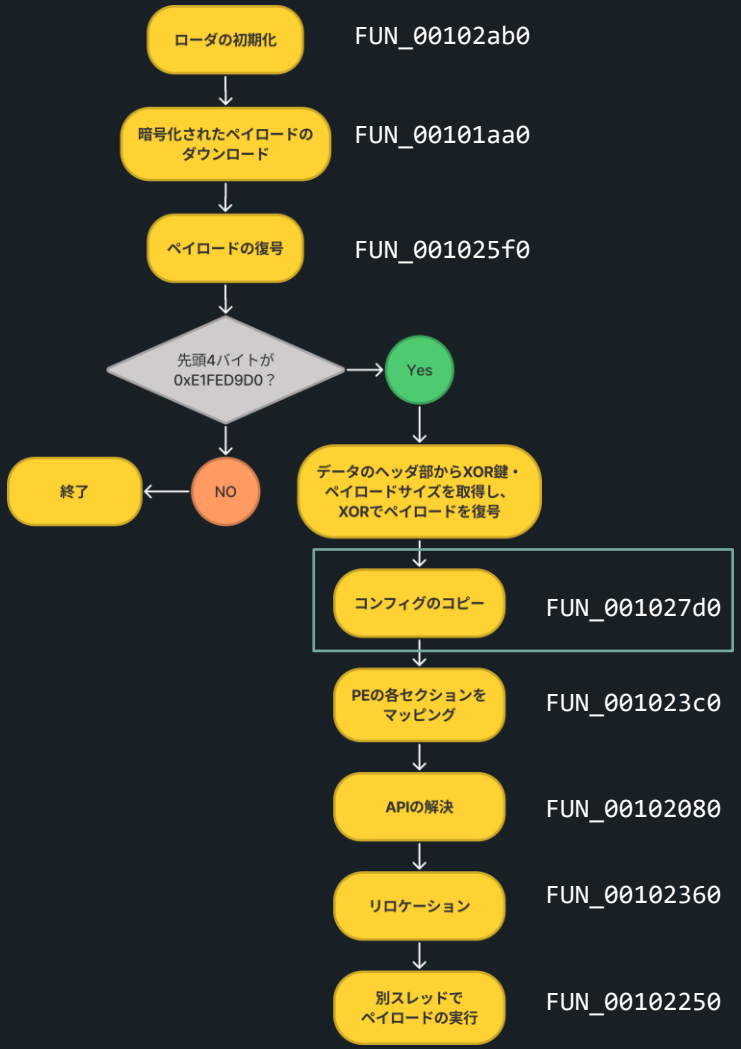
復号後のデータがvalidな
PEファイルかチェック



SpiderPig Loader: Remote

復号したペイロードに埋め込まれた“fixmeconfig”というシグネチャを検索し、そこに暗号化コンフィグをコピー

```
43        pcVar7 = pcVar7 + 1;
44        pcVar3 = "fixmeconfig";
45        do {                    ”fixmeconfig”を検索
46          pcVar2 = pcVar3;
47          pcVar3 = pcVar2 + 1;
48        } while (*pcVar2 != '\0');
49      } while (pcVar7 < pcVar1 + -(int)(pcVar2 + -0x14d3d4));
50    }
51    if (pcVar7 < pcVar1 + -0xb) {
52      puVar6 = &DAT_00159ea0;   ← 暗号化されたコンフィグをコピー
53      puVar8 = (undefined4 *)(pcVar7 + *(int *)(in_EAX + 0x14));
54      for (iVar4 = 0x100; iVar4 != 0; iVar4 = iVar4 + -1) {
55        *puVar8 = *puVar6;
56        puVar6 = puVar6 + 1;
57        puVar8 = puVar8 + 1;
58      }
59      return 1;
```



Tips for Detection

- まずは検知ロジックに組み込みやすい文字列からYARAを作成してみる
- 次に、SpiderPig Loaderには存在していてOSSのPeLdrには存在しないコードや文字列部分を特定して、ロジックに組み込む
- 変更されにくそうな部分をロジックに組み込む

4

YARA Pretty Darby

我々が用意した約1万ファイルの中から13のSpiderPigファミリを見つけ出す「いい」YARAルールを書いてもらう、全員参加型ダービー戦！

Darby


Recent jobs
Job name
Author
Matches
Status/Progress
Actions
Trojan_SPIDERPIG
2022-01-16T06:01:39.000Z
(no author)
9
9 / 9 (100%)
Trojan_SPIDERPIG
2022-01-16T06:01:36.000Z
(no author)
9
9 / 9 (100%)
Trojan_SPIDERPIG
2022-01-16T06:01:31.000Z
(no author)
9
9 / 9 (100%)
yara_sample
2022-01-16T06:01:30.000Z
Allsafe
682
698 / 1589 (44%)
Trojan_SPIDERPIG
2022-01-16T06:01:27.000Z
(no author)
9
9 / 9 (100%)
yara_sample
2022-01-16T06:01:27.000Z
Allsafe
682
698 / 1589 (44%)
yara_sample
2022-01-16T06:01:21.000Z
Allsafe
682
698 / 1589 (44%)
yara_sample
2022-01-16T06:01:17.000Z
Allsafe
928
948 / 1589 (60%)
yara_sample
2022-01-16T06:01:14.000Z
Allsafe
928
948 / 1589 (60%)
10%
yara_sample
2022-01-16T06:01:05.000Z
Allsafe
1195
1228 / 1589 (77%)
« ‹ 1 2 3 4 5 › »

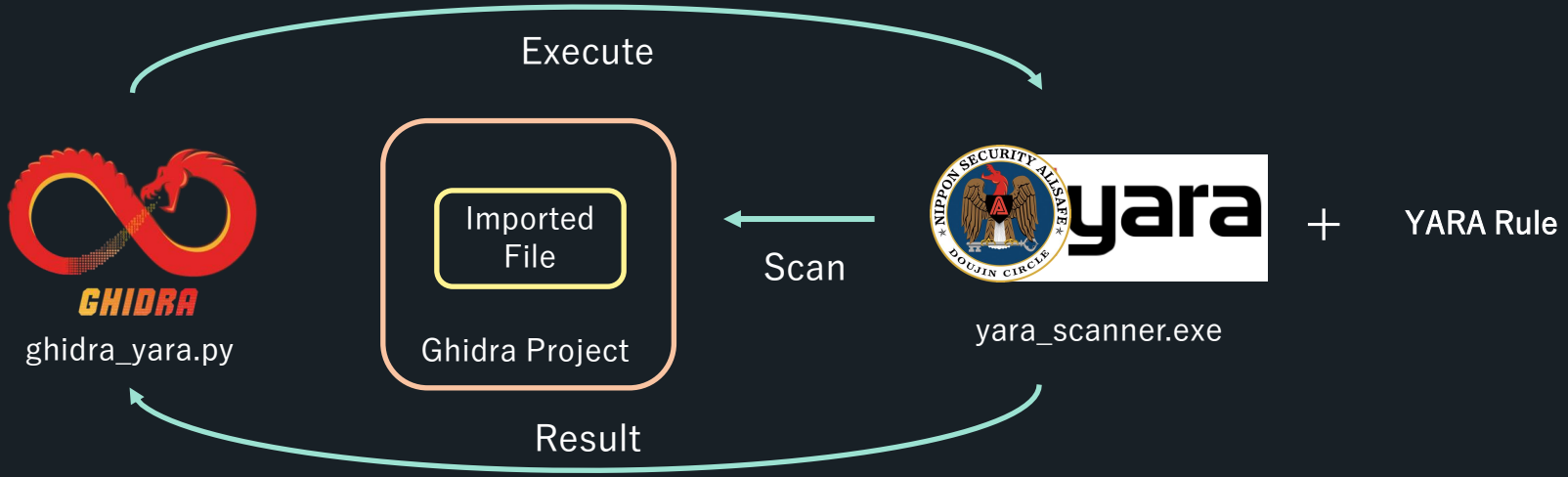
Flow

1. GhidraをつかってYARAルールを作成
2. ローカルテスト
    - 正しくルールが動作するか検証する
3. Webサーバのデータセットでテスト
    - 指定されたハッシュがすべて含まれるか (False Negative)
    - 指定されたハッシュ以外が含まれないか (False Positive)

Test YARA rule (Local Testing)

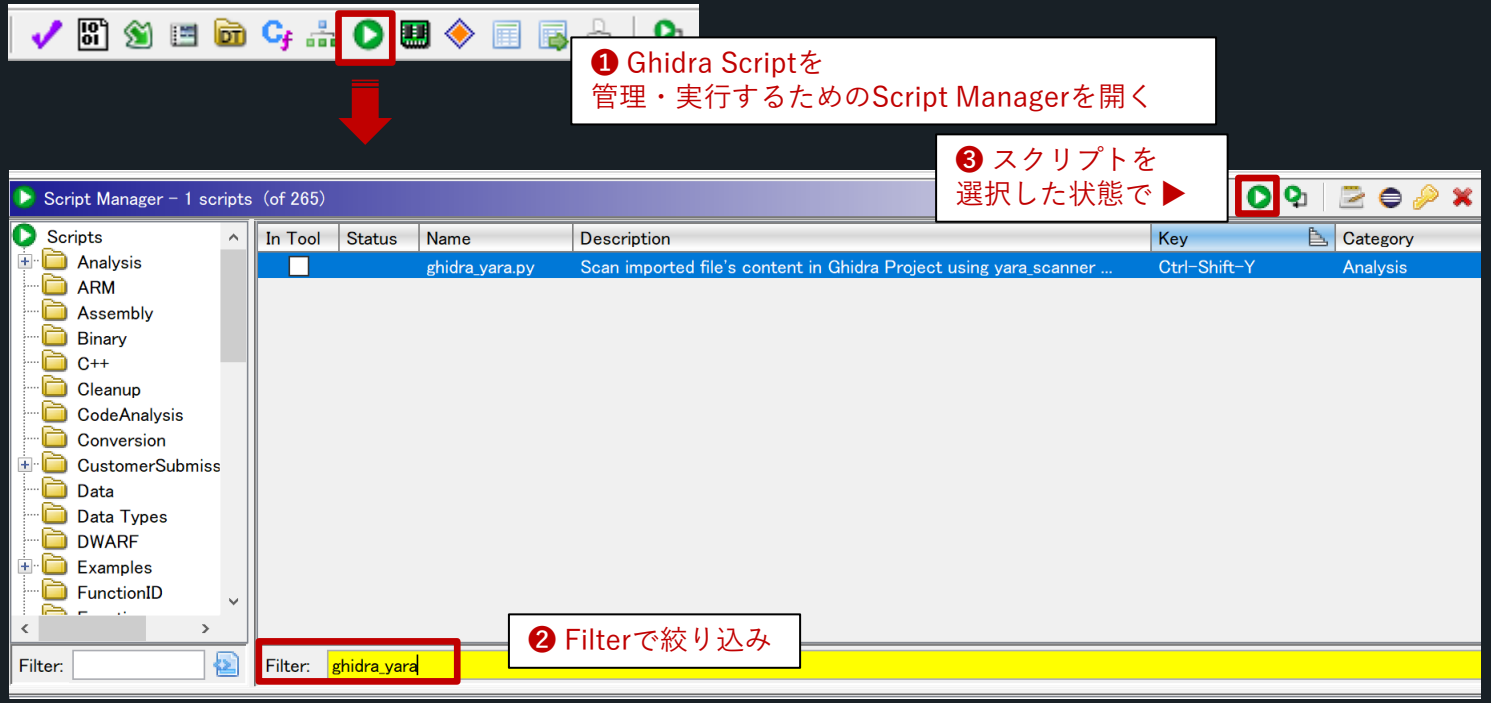
What is ghidra_yara.py ?

- Ghidra にインポート済みのファイルに対してスキャン可能なYARA (yara_scanner.exe)を利用するGhidraScript
  - 通常のYARA ではGhidra Project 内に含まれている解析対象のファイル本体をスキャンできない
  - 解析対象のファイル (マルウェア) を直接取り扱うのは危険が伴う



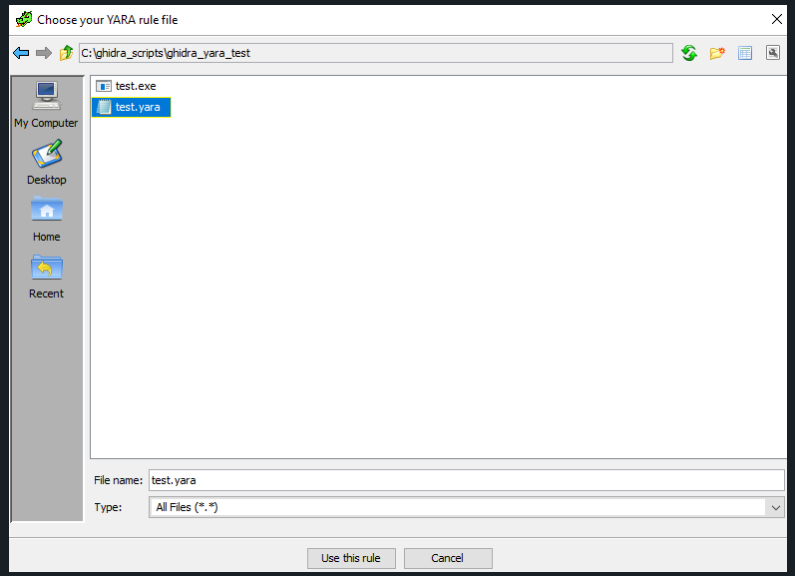
Step1 : How to use ghidra_yara.py

- Script Managerからghidra_yara.pyを選択して実行



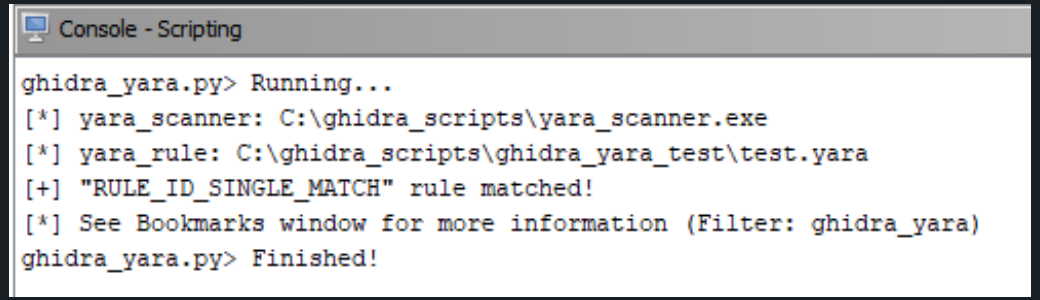
Step2 : How to use ghidra_yara.py

- YARAルールのファイル選択ダイアログが表示されたら、スキャンしたいYARAルールのファイルパスを選択
  - C:¥ghidra_scripts¥rule.yaraを設置すると省略可能



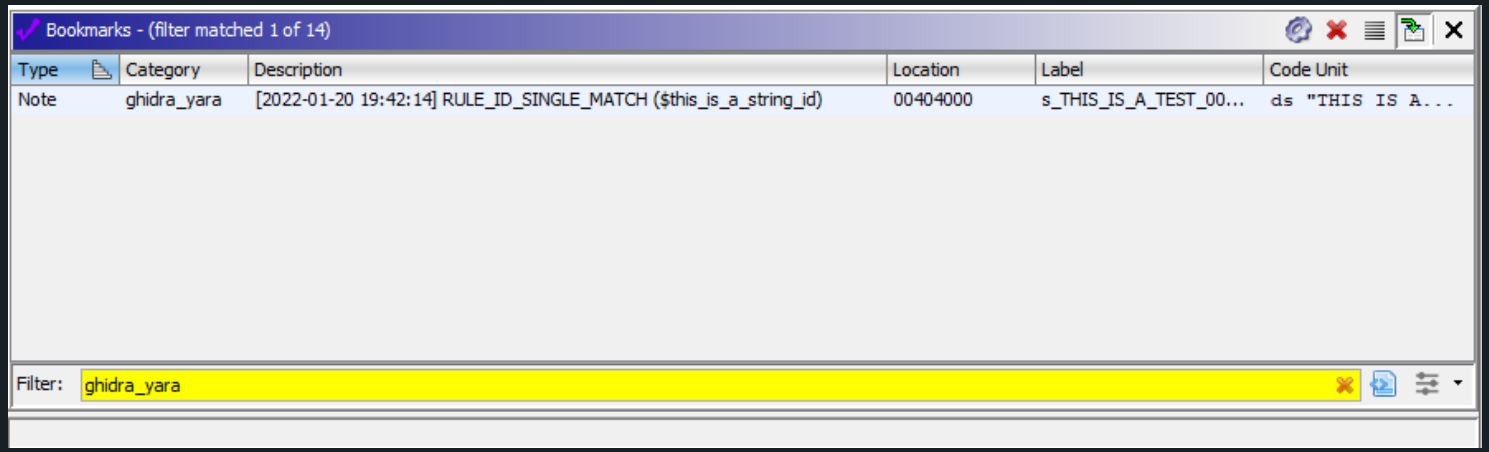
Step3 : How to use ghidra_yara.py

- Consoleウィンドウにスキャン結果が表示される
  - マッチした場合は、そのルール名が表示される



Step4 : How to use ghidra_yara.py

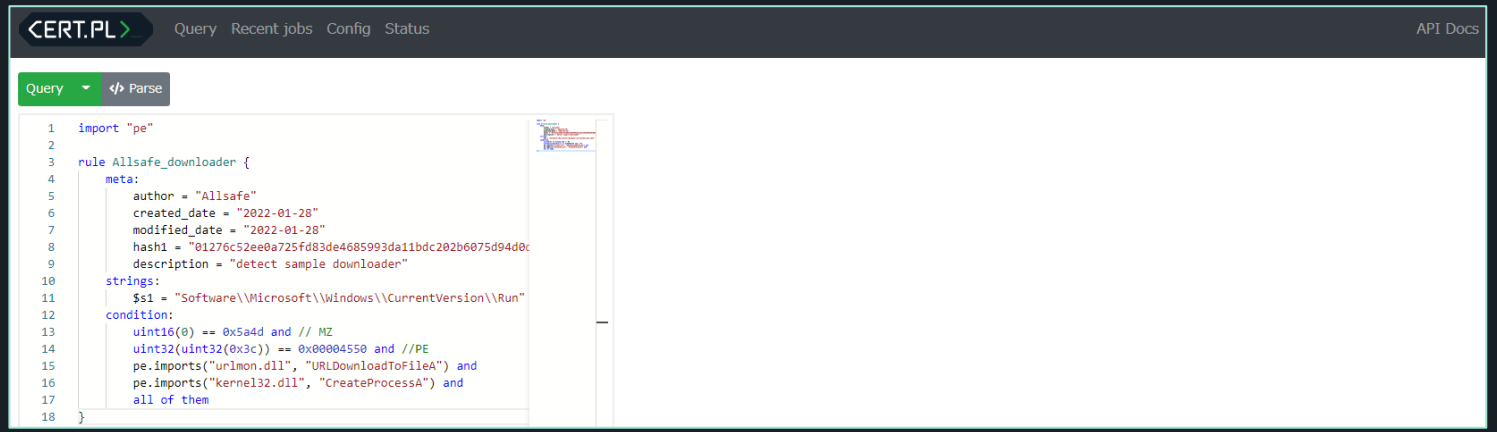
- Bookmarksウィンドウで検知結果の詳細を確認
  - [Window] -> [Bookmarks]から起動
  - "ghidra_yara"でフィルタして確認
  - `Location`をダブルクリックすることで、マッチしたアドレスに移動可能



Evaluation YARA rule (Test Dataset)

mquery

- CERT PLが開発したOSSのYARA検索用のWebサービス
  - https://github.com/CERT-Polska/mquery
  - 大量のデータを高速に検索することができる
- 演習ではAllsafeが用意した環境を利用する
  - アクセス方法はslackで共有

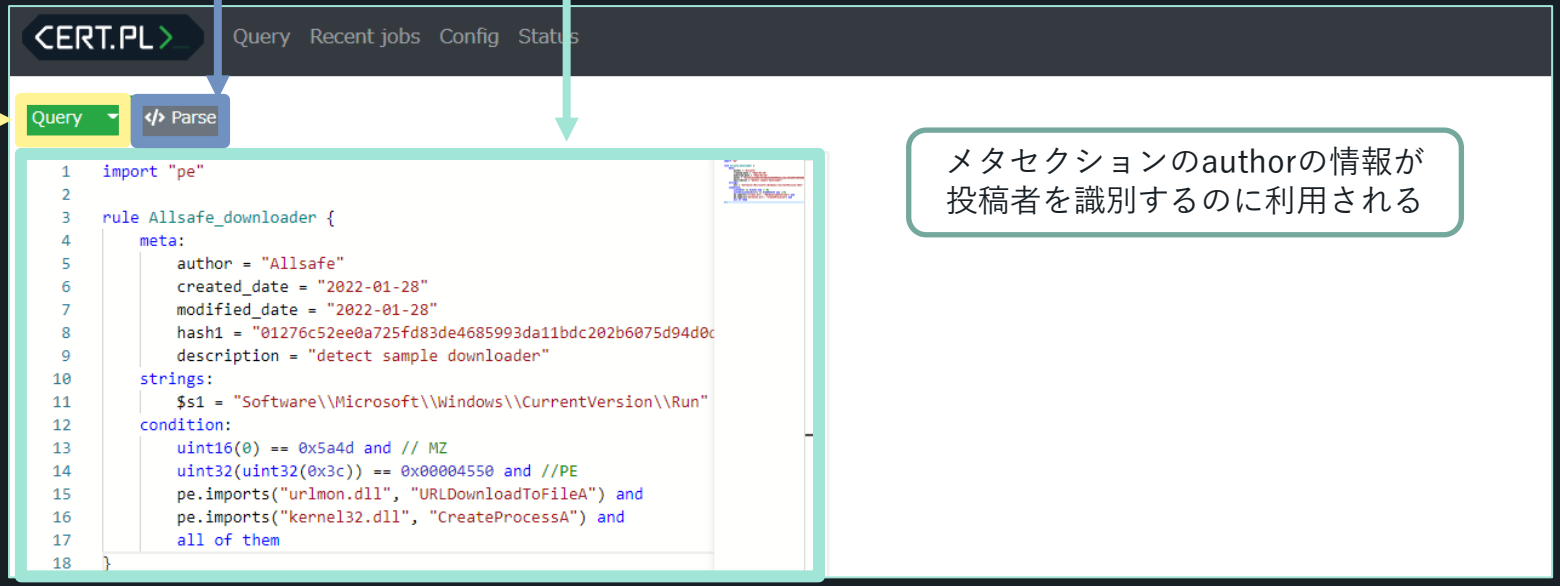


mquery Usage

構文の検証

YARAルールを入力

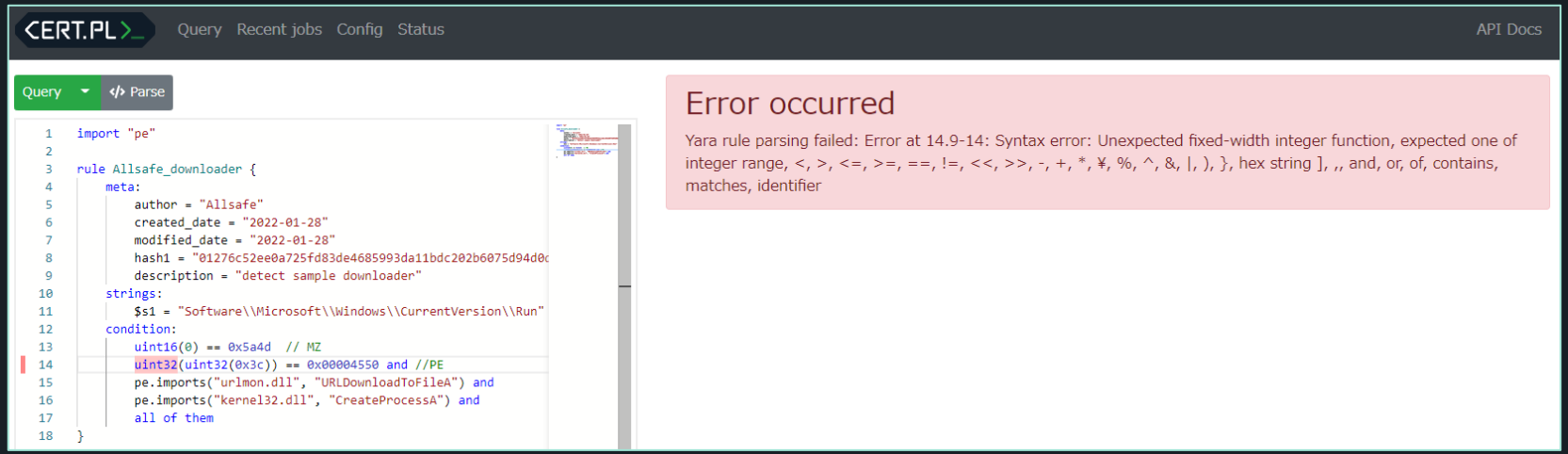
検索を実行



メタセクションのauthorの情報が
投稿者を識別するのに利用される

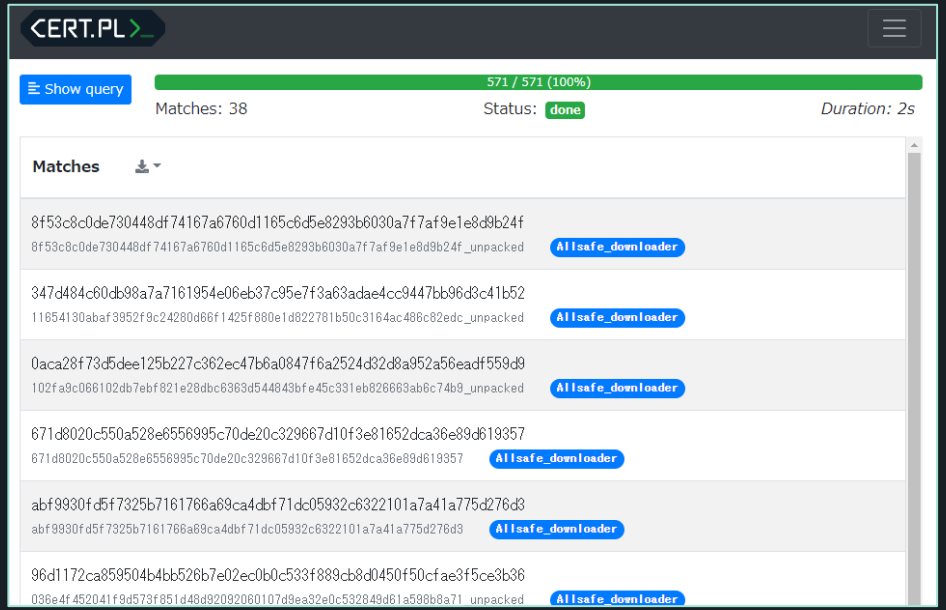
Parse

- 構文エラーをチェック
  - 基本的にはローカルでチェックするため利用しない



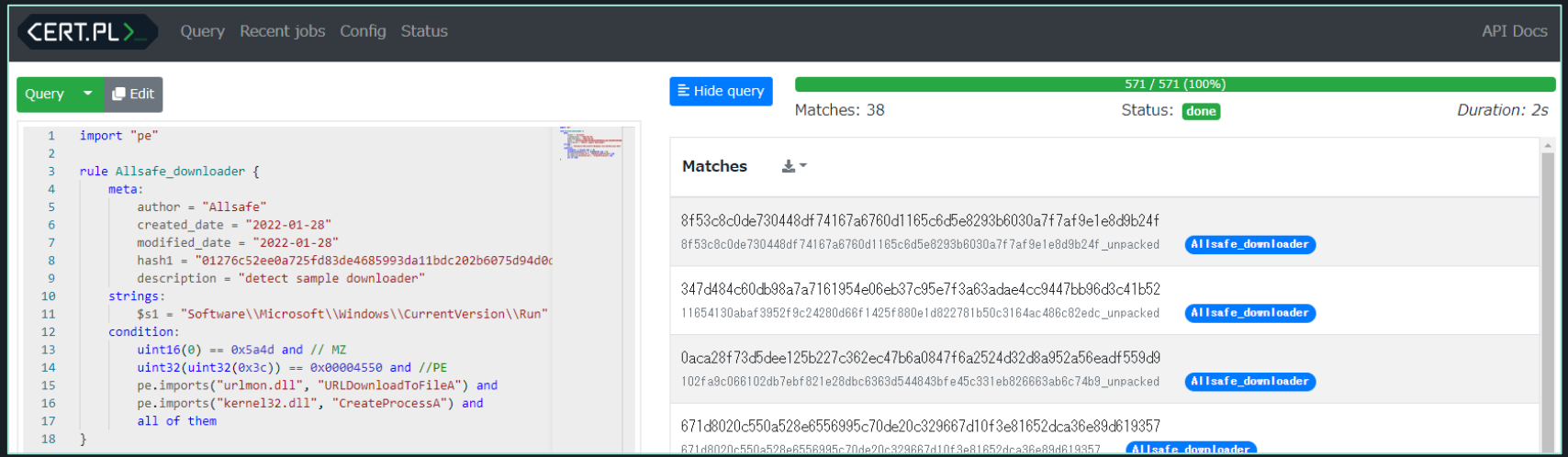
Query

- 検索結果を表示
- 正解のハッシュしか含まれないかを確認する
- ルールをトライアンドエラーで修正する



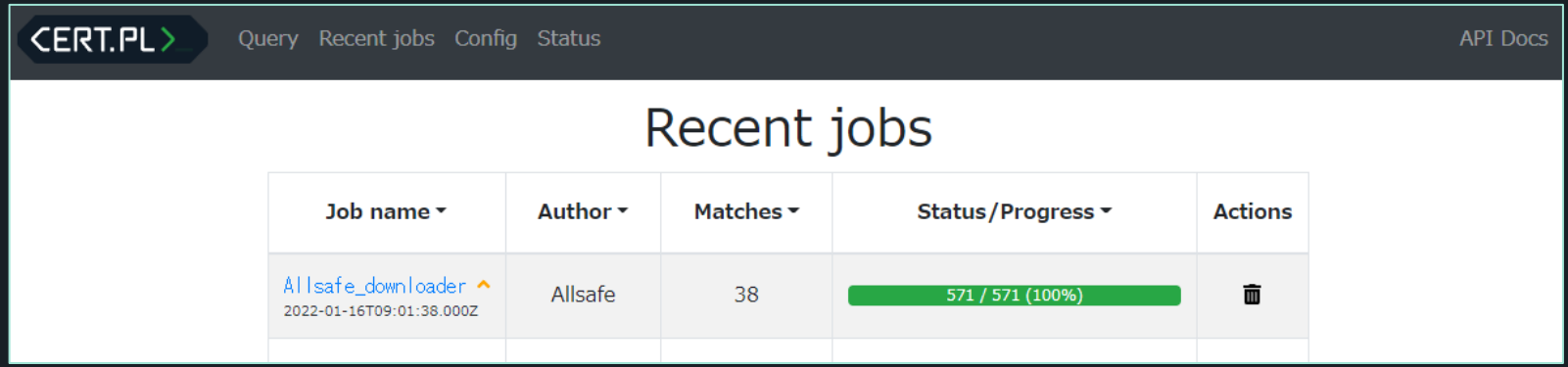
Query (Show query)

- 左上のShow queryから実行したクエリを確認できる



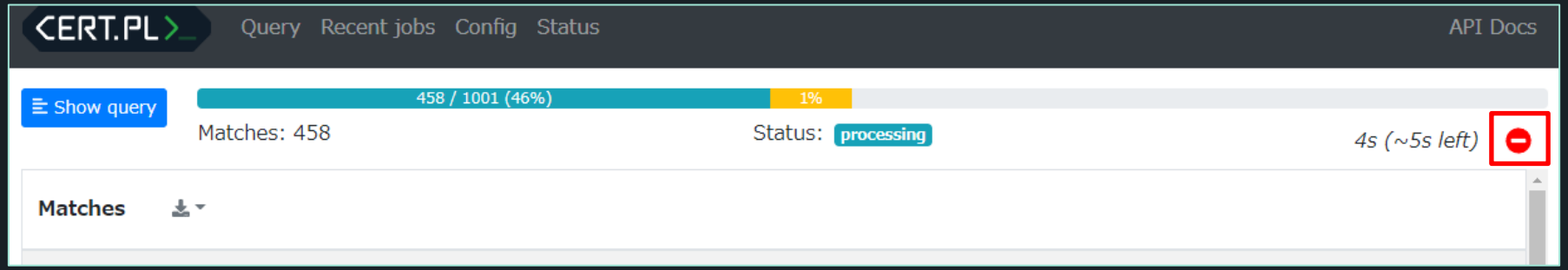
Recent jobs

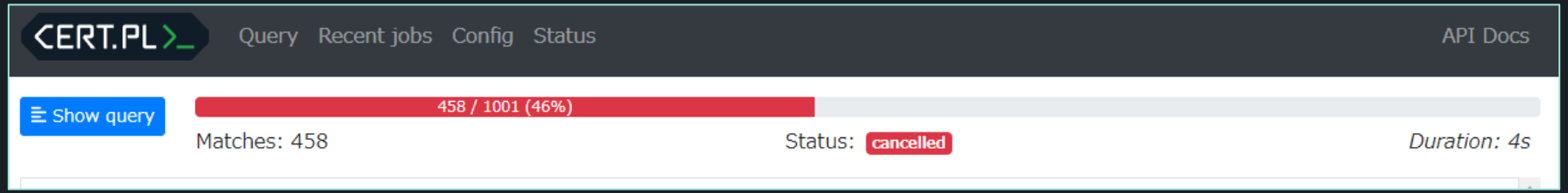
- 過去のqueryを確認できる
  - ユーザ毎に分離はされていないのでAuthorを目印に自分の投稿を探す



Cancel

- 想定以上に検知されてしまうルールは右上の停止ボタンからキャンセルする
  - サーバの負荷軽減にご協力お願いします





YARA Pretty Darby Play Book

DO

- ✓ フォーマットに従って書く
  - Meta情報を埋める
  - ルール名を統一する
  - 公式の構文に従う
- ✓ なぜ・なにを検知しようとしたのかわかるコメントを書く
- ✓ 検出の目的を加味し、誤検知の許容範囲を意識してルールをつくる
- ✓ 可能であればバイト列を含めたルールを作成する

DO NOT

- ✓ 4文字未満の短すぎる文字列を避ける
- ✓ 特徴的でない文字列を避ける
  - PEに必ず含まれる文字列
  - 正規プログラムでも利用されるユーザエージェントなどの文字列
- ✓ 正規表現を利用する場合はパフォーマンスに気をつける
  - 文字数を制限しない正規表現を使わない
- ✓ 同一ファイル内でマッチしすぎるルールを避ける

Good Rule vs Bad Rule

Good

```
rule APT_Allsafe_Win64_NotAllSafe
{
    meta:
        author = "Allsafe"
        create_date = "2022-01-28"
        modified_date = "2022-01-28"
        description = "detect allsafe trojan x64"
        hash = "fc79aa4a887e4a01c64a7c75cc48b16cdebf8f69"

    strings:
        // file name of encoded config
        $fname_cfg = "\\NotAll.safe" nocase
        // initialize decode key of config
        $stack_string_key = {
            // MOV BYTE PTR [RBP + ??], 'A'
            c6 45?? 41
            // MOV BYTE PTR [RBP + ??], 'l'
            c6 45?? 6c
            // MOV BYTE PTR [RBP + ??], 'l'
            c6 45?? 6c
            // MOV BYTE PTR [RBP + ??], 's'
            c6 45?? 73
            // MOV BYTE PTR [RBP + ??], 'a'
            c6 45?? 61
            // MOV BYTE PTR [RBP + ??], 'f'
            c6 45?? 66
            // MOV BYTE PTR [RBP + ??], 'e'
            c6 45?? 65
        }

    condition:
        uint16(0) == 0x5a4d and
        all of them
}
```

Bad

```
rule mal{
    strings:
    $a="This program cannot be run in DOS mode."
    $b =/.*\.exe$/
    condition:any of them
}
```

Dataset

- ターゲット
  - SpiderPig Loader: 9
  - SpiderPig Rat: 4 (+ 1)
- PEマルウェア
  - 9772

Sample

- SpiderPig Loader Remote
  - 7da969010a55919aa66ed97a2d2d6d6a0be3d8dc6151eeb6cebc15e4f06d4553
- SpiderPig Loader Local
  - 8bdfc1ed5bfec964050a42a0f1ddd8709fcf14fab1ede151c5a7161be904cd96
- SpiderPig RAT
  - d196969b35966462fa03ef857e375e9d6172b34053b115df04cefa3d673b9d85

Ground Truth

- SpiderPig Loader Remote
  - 7da969010a55919aa66ed97a2d2d6d6a0be3d8dc6151eeb6cebc15e4f06d4553
  - 1e25116f33f7248e4549cb15fb20bd5d9f87cc7424e6592e565d66095ec2b647
  - 2321690bb6cab49c9eb828c4b65182ceb05653479fe900b9e6dbd93a0b9a672f
  - 90406d0fc975f342f0e20b49e7946e891392eb06bfc8cc5f3b9b8c86b7c1b17a
  - 3891fb7b3d1e5fc2d028ed3d0debe868189971b20eb8edb295e2b8d2d0c1a02a
  - 5a57c9d19c7fb42832085f88d92f9f57d64b1bca8f2a19b0533a4caee1a792cc
  - 2657ca121a3df198635fcc53efb573eb069ff2535dcf3ba899f68430caa2ffce
- SpiderPig Loader Local
  - 8bdfc1ed5bfec964050a42a0f1ddd8709fcf14fab1ede151c5a7161be904cd96
  - 92c75df382218e7743359aa83b403e443550e766c8474a59c9dcbd4903a4bf02
- SpiderPig RAT
  - 8c3df0e4d7ff0578d143785342a8033fb6e76ce9f61c2ea14c402f45a76ab118
  - c2b23689ca1c57f7b7b0c2fd95bfef326d6a22c15089d35d31119b104978038b
  - d196969b35966462fa03ef857e375e9d6172b34053b115df04cefa3d673b9d85
  - 733b4d5174669caab2bbcc9bfe51606a13346b70af59fccea4f479d1fde7b5d7
  - Dced553a6f835162f0515a41a330404466f3ca44bc43a2f8b5675ca28609c905 (x86) //実装が異なるため検知できなくてもよい

出走！

Solutions

- 別途配布するPDFを参照

Summary

- 単に検出できるかどうかだけではなく、検出の目的や効率を考慮した「いい」YARAルールを書くように心がけましょう
- 「いい」YARAルールは、反復検証の結果として出来上がるので、仮説→検証を繰り返しましょう
- 書いたYARAルールをコミュニティに共有してみましょう

THANKS!