Witch's Key Party
魔女のお茶会

CTFでRev問題を作成した時のポイントを振り返る

@kazukiigeta

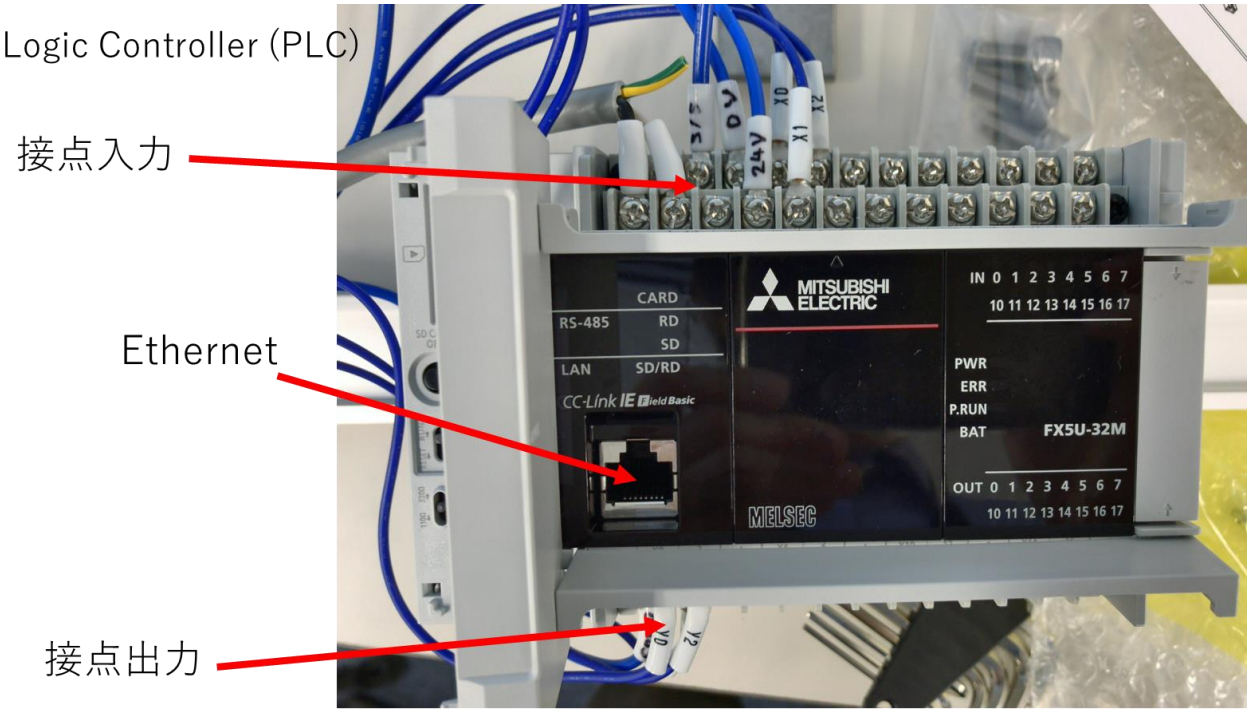
2021/05/23

@kazukiigeta

昨年まで**制御システムセキュリティ**
日本全国の制御システムをまわっていました。

PCAPとって**独自制御プロトコルをリバーシング**
独自制御プロトコル専用のセキュリティ対応スクリプト開発

Programmable Logic Controller (PLC)



Kazuki Igeta
kazukiigeta

Follow

2 followers · 3 following · ☆ 15

Japan
Sign in to view email
https://binarygenes.com

Highlights
Arctic Code Vault Contributor

@kazukiigeta

昨年まで**制御システムセキュリティ**
　日本全国の制御システムをまわっていました。

　PCAPとって**独自制御プロトコルをリバーシング**
　独自制御プロトコル専用のセキュリティ対応スクリプト開発

趣味として**マルウェア解析**をやっています。

趣味も業務もデータドリブンな活動を取り入れていきたいと思い、
今年から**メイン業務をデータサイエンス系**に変更しました。

セキュリティ成分が薄まってきています😟

こんな発表させてもらいますが、競技者としてCTFに
参加した経験は少なく、競技者レベルも低いです🫧

Kazuki Igeta
kazukiigeta

Follow ···

2 followers · 3 following · ☆ 15

Japan
Sign in to view email
https://binarygenes.com

Highlights
✳ Arctic Code Vault Contributor

関わったCTF（話せるもの）①

実際のビルシステムの通信解析をしてきた経験を活かして作問

http://mnctf.info/mnctf2019/

建屋制御Ⅱ

カテゴリ：ネットワーク スコア：80 Pts 正解者数：204人

建屋制御Ⅰの続きの問題です。
マクニキセロリの生育のカギとなる「工場の室温」を調べなさい。
※2019/07/02 11:06(UTC)時点の温度を小数点第5桁まで答えよ。
例）10.12345

建屋制御Ⅰ

カテゴリ：ネットワーク スコア：60 Pts 正解者数：277人

株式会社マクニキは培ってきたものづくりのノウハウを武器に新しい産業「農業」に進出した。おいしい野菜を作るべく、温度、輝度（照明）を厳格に管理するために新しい制御システムを導入し、試行錯誤の末、自称世界一おいしい「セロリ」を開発することに成功した。マクニキセロリはみずみずしい触感とフルーツのような甘さが話題となり、瞬く間に世界に広まった。
一方その頃、競合である「株式会社八区」は産業スパイを雇い、マクニキの生産工程を窃取することにした。
産業スパイは野菜工場に物理的に侵入し、1階の工場にてマクニキセロリの自動精算を行っていることを突き止めた。さらなる情報収集のため、バックドアデバイス（Cranberry Pi Zero）の設置および、制御システムの設定シートの盗撮に成功した。
Cranberry Pi Zeroから取得したPCAPを解析し、工場の空調コントローラの「location」名を特定せよ。
ヒント1）ビルシステムの制御プロトコルであるBACnetにおいて、deviceオブジェクトのlocationプロパティ内に制御システムの場所情報を保持している。

ここに答えを入力してください。

送信

送信

関わったCTF（話せるもの）②

仮想ボイラー制御システムを持つ
ネットワークを用意して、
制御システムの破壊、情報窃取を演習

Concurrently, participants of the Cyber Technology Challenge had to go through three different challenges that focused on Blockchain, Threat Intelligence and Operational Technology. The challenges were developed by Netpoleon and the Institution of Engineering and Technology (IET), who were Strategic Partners of the KPMG Cyber Challenge 2019.

https://www.uowmkdu.edu.my/news/kpmg-cyber-security-challenge-2019

関わったCTF（話せるもの）③

情報処理に関わる訓練

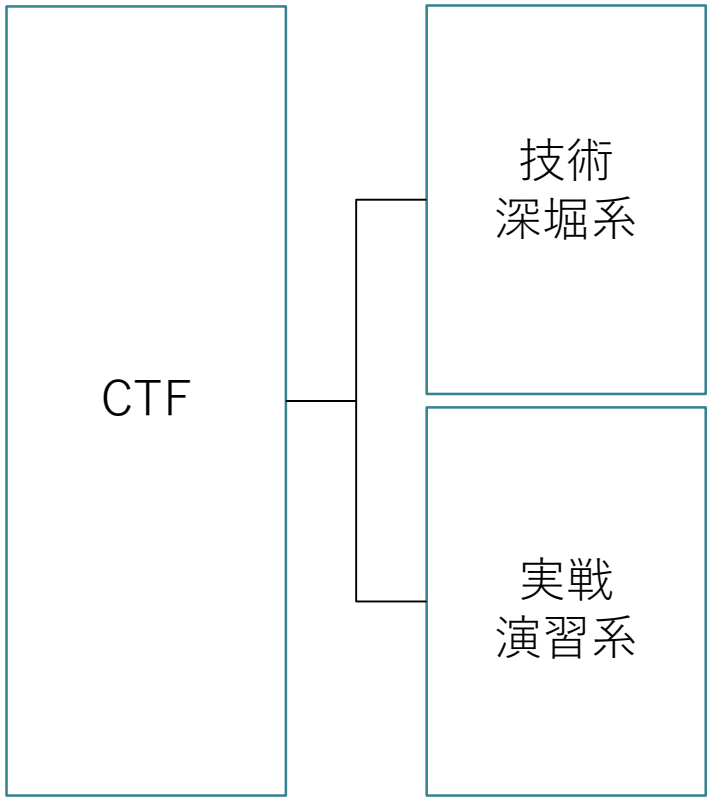
本日のテーマ

1. CTFのRev問題を作るときの**考え方**の1例を紹介
   ⇒皆さんがCTF作問をする場面が来た時の参考にしてもらう

2. Golangで書かれたマルウェアのRevの**技術面**から紹介
   ⇒Rev初心者がGolangバイナリを解析する際の参考にしてもらう

CTFを２つに大別してみる

※私の印象です



競技としてのCTF
DEFCON, SECCON etc.

無償の一般公開されているものは基本的にこちら

セキュリティトレーニングの一環
（インシデントレスポンス・フォレンジック学習）

有償サービスの中に組み込まれることがある

CTFを２つに大別してみる

※私の印象です

競技としてのCTF
DEFCON, SECCON etc.

無償の一般公開されているものは基本的にこちら

セキュリティトレーニングの一環
（インシデントレスポンス・フォレンジック学習）

有償サービスの中に組み込まれることがある

今回は、演習系CTFとして
インシデントレスポンス/フォレンジックの流れで進行する問題の作成
を行った経験からお話しします

今回の作問の流れ（骨格）

※ 1番目と2番目は状況によって入れ替え

今回の作問の流れ（骨格）

※ 1番目と2番目は状況によって入れ替え

1. LinuxのWebサーバに
2. Webshellを仕込まれて
3. RATをインストールされ
4. 機密情報を窃取される

学んでほしいことが明確になる。

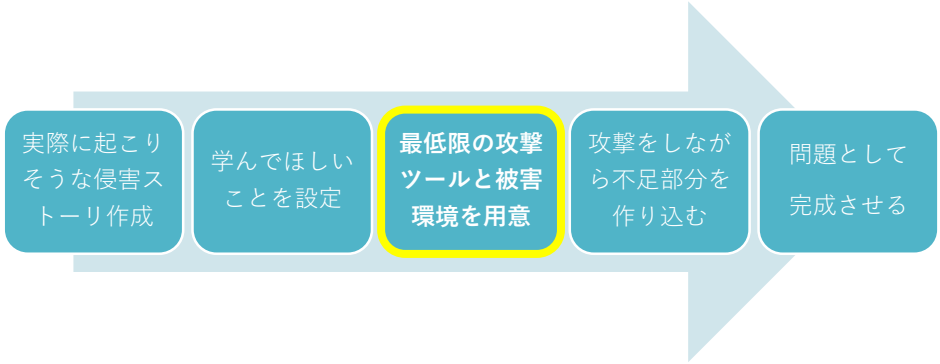
・窃取された情報の特定

・攻撃者の情報収集 TTPs, IOCs etc.

LinuxのWebサーバを構築

RATを開発

教育用RATの開発

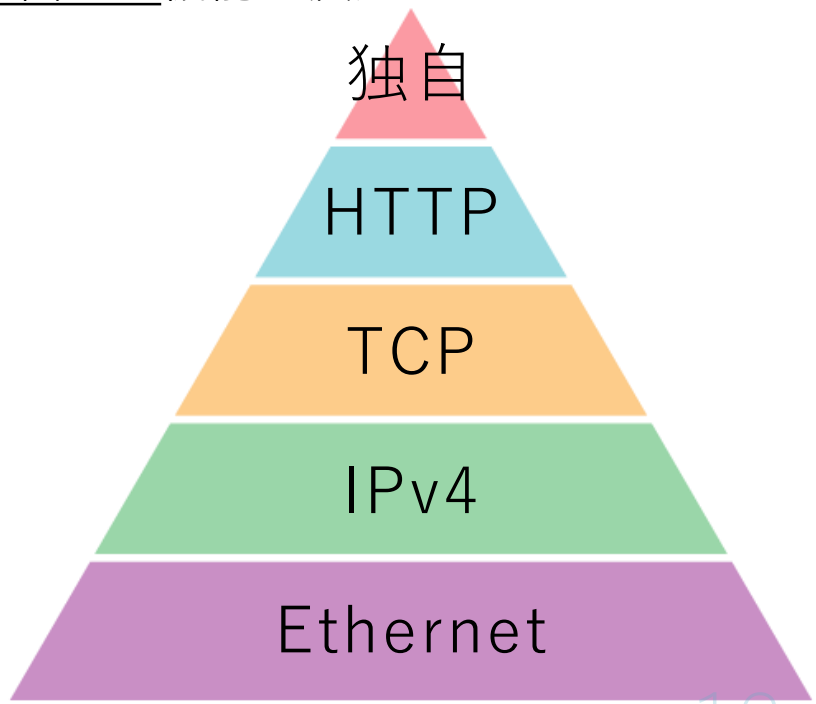


Golangで開発することに決定

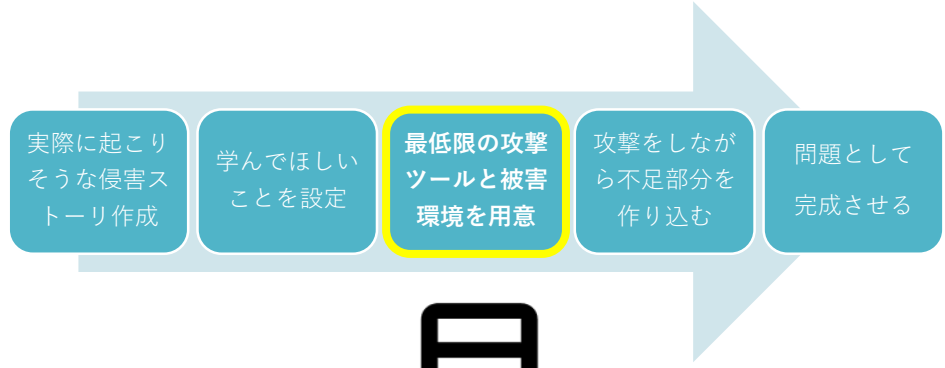
- 理由1: 今までGolangバイナリを解析したことが無かった
- 理由2:自分的に、testableで使いまわしやすいコードが書きやすい

今まで自分が解析した日本を狙っている攻撃者グループが開発したRATを思い出して機能を決定

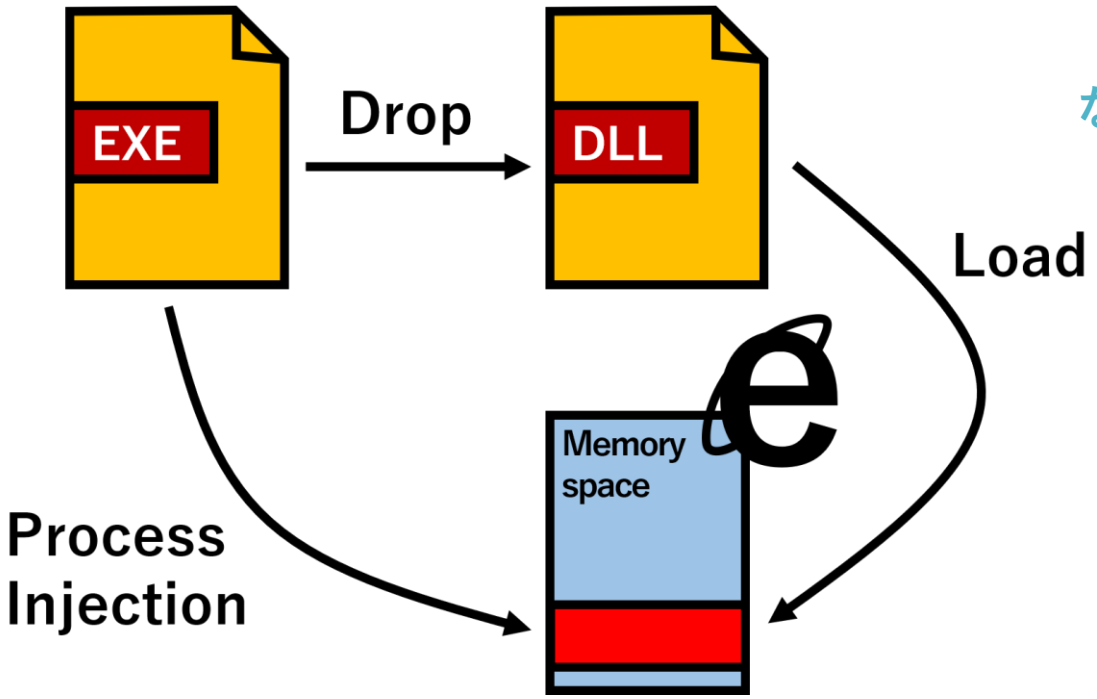
1. 独自通信プロトコルのパーサ（C2通信用）
   ⇒独自プロトコル over HTTP
2. コマンド実行機能
3. 通信処理
   ⇒HTTPクライアント
4. 暗号化/復号処理



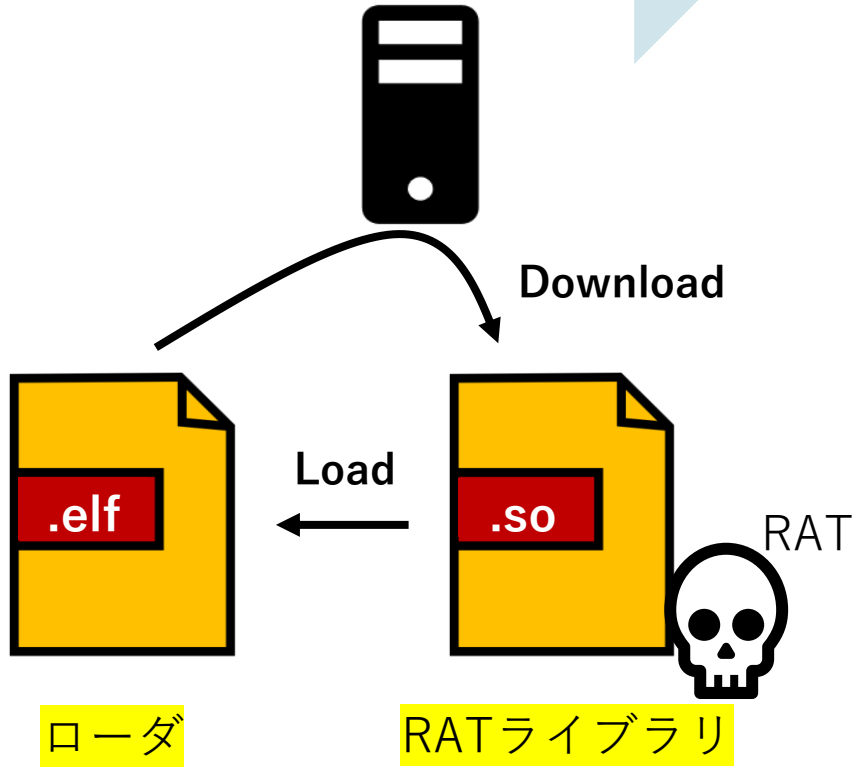
開発した教育用RATの動作



以前解析した本物のマルウェアの動作※



マルチステージを
採用しつつ
なるべくシンプルに



ローダにはRATの動作は含まれていないため
RATライブラリも併せて解析する必要がある

※https://binarygenes.com/posts/2020-03-12-analyzing-shellcode-with-statically-linked-dll

ローダの機能: RATライブラリをロード

実際に起こりそうな侵害ストーリ作成 | 学んでほしいことを設定 | **最低限の攻撃ツールと被害環境を用意** | **攻撃をしながら不足部分を作り込む** | 問題として完成させる

```go
p, err := plugin.Open(f.Name())
if err != nil {
    fmt.Printf("failed to open plugin: %s", err)
    os.Exit(-1)
}
rat, err := p.Lookup("SGRT")
if err != nil {
    fmt.Fprintln(os.Stderr, err)
    os.Exit(-1)
}
rat.(func())()
```

RAT共有ライブラリ（plugin）をOpen

“SGRT”という名称の関数ポインタを取得

“SGRT”関数をCall

最初は”RAT”という名称で作っていたが、**問題に使えそう**と思い”SGRT”に変更

RAT共有ライブラリのRevに移るための繋ぎとして、
ローダが呼び出している共有ライブラリのエクスポート関数名を答えさせることに。

GolangバイナリRevの初歩

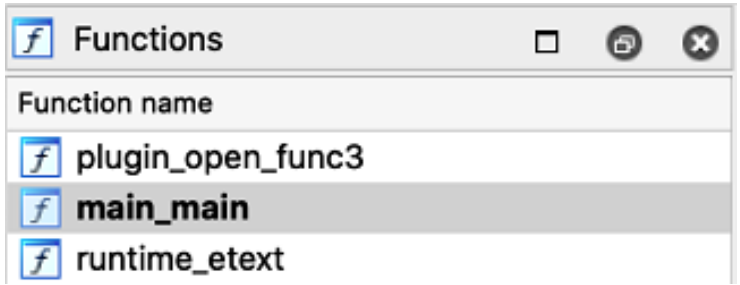
実際に起こりそうな侵害ストーリ作成 → 学んでほしいことを設定 → 最低限の攻撃ツールと被害環境を用意 → 攻撃をしながら不足部分を作り込む → **問題として完成させる**

以下のテーマで、GolangバイナリのRevの始まりを説明

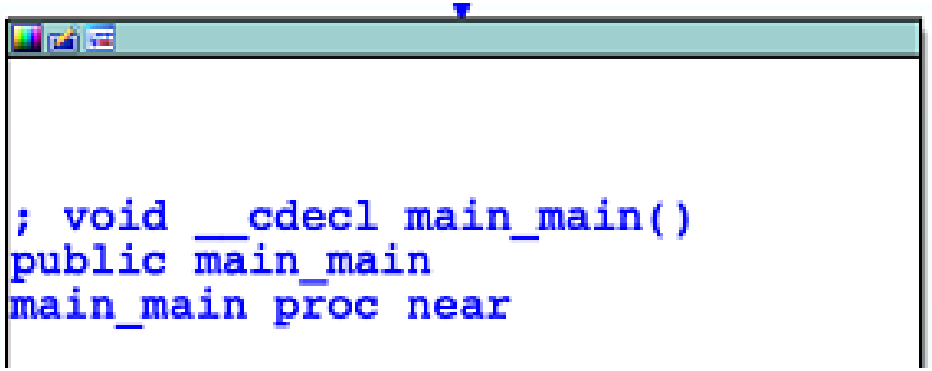
「ローダが呼び出している共有ライブラリのエクスポート関数名を答える」

Step1. main関数のSymbol名は**main.main**

逆アセンブラやデバッガでバイナリを見るときは基本的にココから（以降、IDA free v7.6 for Macで例示）



※ IDAではmain_mainと表示されるがmain.mainのこと。



Step2. 重要そうな関数の呼び出しを追いかける

GolangバイナリRevの初歩

実際に起こりそうな侵害ストーリ作成 → 学んでほしいことを設定 → 最低限の攻撃ツールと被害環境を用意 → 攻撃をしながら不足部分を作り込む → **問題として完成させる**

以下のテーマで、GolangバイナリのRevの始まりを説明

「ローダが呼び出している共有ライブラリのエクスポート関数名を答える」

Step2. 重要そうな関数の呼び出し周りを調べる

- main.mainをざっと眺めると、**plugin.Open**関数が見つかる。
  ⇒共有ライブラリをロードしていると分かる。
- pluginパッケージの残りの関数がどこに出てくるかという目線で再度ざっと眺める。
  ⇒何も見つからないが、call raxが共有ライブラリ上の関数のポインタ呼び出しであると推測できる。

```
mov     rax, [rcx+38h]
mov     rcx, [rcx+40h]
mov     qword ptr [rsp+100h+var_100], rax
mov     qword ptr [rsp+100h+var_100+8], rcx
call    plugin_open
mov     rax, [rsp+100h+var_E0]
mov     rcx, qword ptr [rsp+100h+var_F0+8]
mov     rdx, qword ptr [rsp+100h+var_F0]
```

```
mov     rdx, [rsp+100h+var_70]
mov     rax, [rdx]
call    rax
```

```go
p, err := plugin.Open(f.Name())
if err != nil {
    fmt.Printf("failed to open plugin: %s", err)
    os.Exit(-1)
}
rat, err := p.Lookup("SGRT")
if err != nil {
    fmt.Fprintln(os.Stderr, err)
    os.Exit(-1)
}
rat.(func())()
```

- エクスポート関数名を指定する**Lookup**()はどこにある？

GolangバイナリRevの初歩

実際に起こりそうな侵害ストーリ作成 → 学んでほしいことを設定 → 最低限の攻撃ツールと被害環境を用意 → 攻撃をしながら不足部分を作り込む → **問題として完成させる**

以下のテーマで、GolangバイナリのRevの始まりを説明

「ローダが呼び出している共有ライブラリのエクスポート関数名を答える」

Step2. 重要そうな関数の呼び出し周りを調べる

- エクスポート関数名を指定する**Lookup**()はどこにある？

Pluginパッケージのソースコードを読んでみると、
Golangの**Channel型**をつかって共有ライブラリから
実行コードを読みだしているっぽい？？

```
20  // Plugin is a loaded Go plugin.
21  type Plugin struct {
22          pluginpath string
23          err        string        // set if plugin failed to load
24          loaded     chan struct{} // closed when loaded
25          syms       map[string]interface{}
26  }
```

⇒**Lookup**()は最適化されて関数としては
callされていないがmain関数に存在すると考えられる

```
43  func open(name string) (*Plugin, error) {
44          cPath := make([]byte, C.PATH_MAX+1)
45          cRelName := make([]byte, len(name)+1)
46          copy(cRelName, name)
47          if C.realpath(
48                  (*C.char)(unsafe.Pointer(&cRelName[0])),
49                  (*C.char)(unsafe.Pointer(&cPath[0]))) == nil {
50                  return nil, errors.New(`plugin.Open("` + name + `"): realpath failed`)
51          }
52
53          filepath := C.GoString((*C.char)(unsafe.Pointer(&cPath[0])))
54
55          pluginsMu.Lock()
56          if p := plugins[filepath]; p != nil {
57                  pluginsMu.Unlock()
58                  if p.err != "" {
59                          return nil, errors.New(`plugin.Open("` + name + `"): ` + p.err + ` (p
60                  }
61                  <-p.loaded
62                  return p, nil
```

https://github.com/golang/go/blob/master/src/plugin/plugin.go

https://github.com/golang/go/blob/cca23a73733ff166722c69359f0bb45e12ccaa2b/src/plugin/plugin_dlopen.go#L43

GolangバイナリRevの初歩

実際に起こりそうな侵害ストーリ作成 → 学んでほしいことを設定 → 最低限の攻撃ツールと被害環境を用意 → 攻撃をしながら不足部分を作り込む → **問題として完成させる**

以下のテーマで、GolangバイナリのRevの始まりを説明

「ローダが呼び出している共有ライブラリのエクスポート関数名を答える」

Step2. 重要そうな関数の呼び出し周りを調べる

- エクスポート関数名を指定する**Lookup**()はどこにある？

（途中の手順を若干省略しますが）実は、お目当ての”SGRT”という文字列は
IDA上で”SGRTThai”として**別の文字列と連結した状態で表示**されています。

なぜ”SGRT”単体の文字列として保持されていないのかというと…

```
mov     qword ptr [rsp+100h+var_100], rdx
mov     qword ptr [rsp+100h+var_100+8], rcx
lea     rcx, aSgrtthai  ; "SGRTThai"
mov     qword ptr [rsp+100h+var_F0], rcx
mov     qword ptr [rsp+100h+var_F0+8], 4
call    runtime_mapaccess1_faststr
mov     rax, [rsp+100h+var_E0]
```

Call時のスタックの様子

rsp+100h+var_100	t *maptype
rsp+100h+var_100+8	h *hmap
rsp+100h+var_F0	key string 文字列へのポインタ
rsp+100h+var_F0+8	key string 文字列の長さ

```
func mapaccess1_faststr(t *maptype, h *hmap, ky string) unsafe.Pointer
```

GolangバイナリRevの初歩

実際に起こりそうな侵害ストーリ作成 → 学んでほしいことを設定 → 最低限の攻撃ツールと被害環境を用意 → 攻撃をしながら不足部分を作り込む → **問題として完成させる**

以下のテーマで、GolangバイナリのRevの始まりを説明

「ローダが呼び出している共有ライブラリのエクスポート関数名を答える」

Step2. 重要そうな関数の呼び出し周りを調べる

- エクスポート関数名を指定する**Lookup**()はどこにある？

（途中の手順を若干省略しますが）実は、お目当ての”SGRT”という文字列は
IDA上で”SGRTThai”として**別の文字列と連結した状態で表示**されています。

なぜ”SGRT”単体の文字列として保持されていないのかというと…

type stringStruct

```
type stringStruct struct {
    str unsafe.Pointer
    len int
}
```

理由は、文字列をstringStructという構造体で
文字列の長さと一緒に保存してcall時に渡しているから

GolangバイナリRevの初歩

実際に起こりそうな侵害ストーリ作成 → 学んでほしいことを設定 → 最低限の攻撃ツールと被害環境を用意 → 攻撃をしながら不足部分を作り込む → **問題として完成させる**

以下のテーマで、GolangバイナリのRevの始まりを説明

「ローダが呼び出している共有ライブラリのエクスポート関数名を答える」

Step2. 重要そうな関数の呼び出し周りを調べる

➢ エクスポート関数名を指定する**Lookup**()はどこにある？

（途中の手順を若干省略しますが）実は、お目当ての”SGRT”という文字列はIDA上で”SGRTThai”として**別の文字列と連結した状態で表示**されています。

なぜ”SGRT”単体の文字列として保持されていないのかというと…

戻り値がスタック上の引数の後ろに置かれるのも特徴

```
mov     qword ptr [rsp+100h+var_100], rdx
mov     qword ptr [rsp+100h+var_100+8], rcx
lea     rcx, aSgrtthai  ; "SGRTThai"
mov     qword ptr [rsp+100h+var_F0], rcx
mov     qword ptr [rsp+100h+var_F0+8], 4
call    runtime_mapaccess1_faststr
mov     rax, [rsp+100h+var_E0]
```

Call時のスタックの様子

rsp	t *maptype
rsp+0x8	h *hmap
rsp+0x10	key string 文字列へのポインタ
rsp+0x18	key string 文字列の長さ
rsp+0x20	unsafe.Pointer 戻り値

```
func mapaccess1_faststr(t *maptype, h *hmap, ky string) unsafe.Pointer
```

Golangバイナリのメタデータ

実際に起こりそうな侵害ストーリ作成 → 学んでほしいことを設定 → 最低限の攻撃ツールと被害環境を用意 → 攻撃をしながら不足部分を作り込む → **問題として完成させる**

1コマンドで
攻撃者の開発環境の
ディレクトリ構造などが分かる

```
$ go version -m get-alert-list
get-alert-list: go1.13.8
        path    github.com/kazukiigeta/go-myjvn/examples/get-alert-list
        mod     github.com/kazukiigeta/go-myjvn (devel)
        dep     github.com/cenkalti/backoff     v2.2.1+incompatible
h1:tNowT99t7UNflLxfYYSlKYsBpXdEet03Pg2g16Swow4=
        dep     github.com/google/go-querystring        v1.0.0
h1:Xkwi/a1rcvNg1PPYe5vI8GbeBY/jrVuDX5ASuANWTrk=
```

※この例は、CTFで使ったマルウェアとは関係のないGolangバイナリの情報です

Golangバイナリのメタデータ

実際に起こりそうな侵害ストーリ作成 → 学んでほしいことを設定 → 最低限の攻撃ツールと被害環境を用意 → 攻撃をしながら不足部分を作り込む → **問題として完成させる**

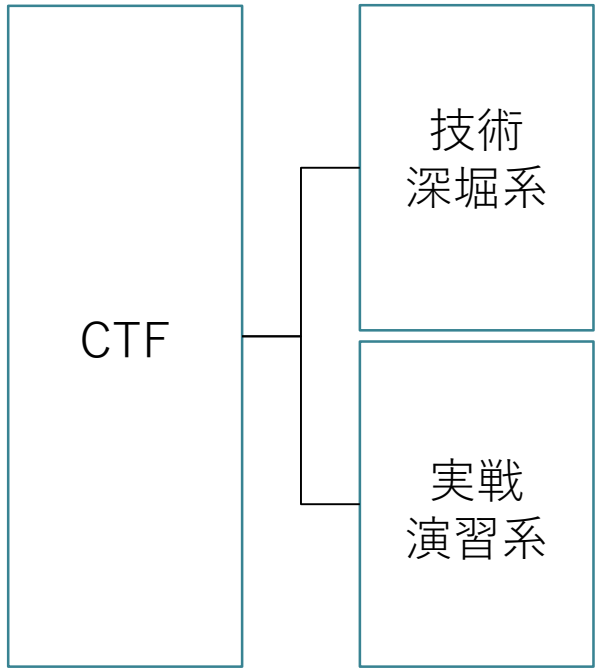
stringsコマンドでも見つけられる情報なので、
問題としての難易度は低いですが
攻撃者の情報を得るという観点では
適していると思います。

1コマンドで
攻撃者の開発環境の
ディレクトリ構造などが分かる

```
$ go version -m get-alert-list
get-alert-list: go1.13.8
        path    github.com/kazukiigeta/go-myjvn/examples/get-alert-list
        mod     github.com/kazukiigeta/go-myjvn (devel)
        dep     github.com/cenkalti/backoff     v2.2.1+incompatible
h1:tNowT99t7UNflLxfYYSlKYsBpXdEet03Pg2g16Swow4=
        dep     github.com/google/go-querystring        v1.0.0
h1:Xkwi/a1rcvNg1PPYe5vI8GbeBY/jrVuDX5ASuANWTrk=
```

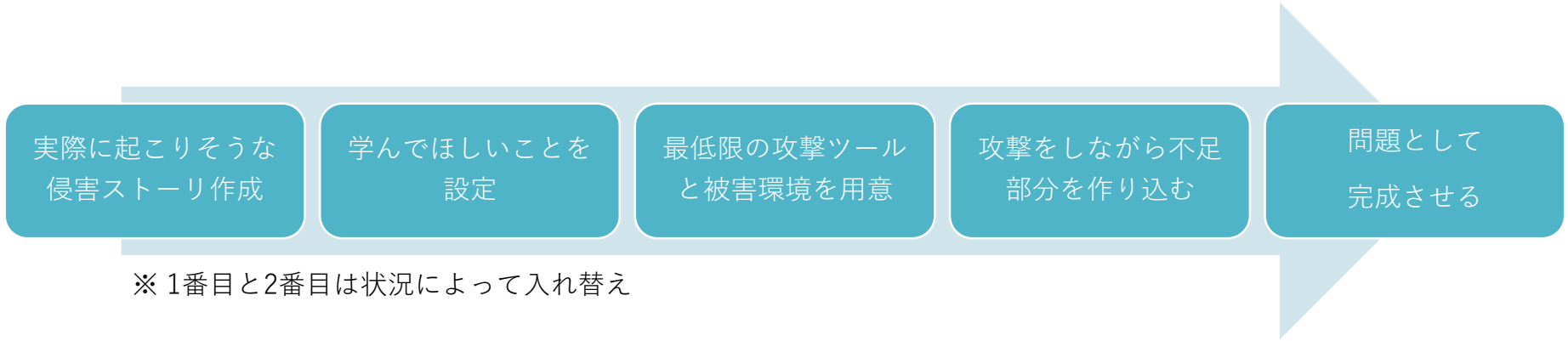
※この例は、CTFで使ったマルウェアとは関係のないGolangバイナリの情報です

最後に



実戦演習系のCTFで作問することがあれば

以下の流れで作ってみるのはいかがでしょうか？



※ 1番目と2番目は状況によって入れ替え

メイン業務をデータサイエンス系に切替えたので、

セキュリティ成分が薄まっています。

セキュリティ関連の遊びがあれば Twitter(**@kazukiigeta**)などで誘ってください！

データサイエンス系も歓迎です！

Kazuki Igeta