進化するGo言語製マルウェアとどう戦うか？：解析能力向上に向けての実践的テクニック

JSAC 2023

株式会社FFRIセキュリティ
桑原翼

自己紹介

桑原翼

- 株式会社FFRIセキュリティ
  - マルウェア解析
  - その他セキュリティサービスの提供
- GhidraやX64dbgのプラグイン開発
  - Go言語製バイナリを解析するためのプラグインなども開発
- SecHack365、セキュリティ・キャンプ全国大会2019修了生
- セキュリティ・キャンプ全国大会2021講師

目次

進化するGo言語製マルウェアとどう戦うか？：解析能力向上に向けての実践的テクニック

- **現状と問題点**
- **解析の基礎**
  - **Go言語製マルウェア解析の経験が少ない方に向けて**
- **発展的な解析**
  - **Go言語製マルウェア解析の経験がある方に向けて**

GO言語製マルウェア
現状と問題点

マルウェア開発におけるGo言語の利点

- Go言語について
  - 記述が容易
  - ライブラリが豊富
  - クロスコンパイル

- マルウェア開発者にとっての利点
  - 開発しやすい
    - マルウェアでもOSSライブラリが良く利用される
  - 複数プラットフォームを攻撃しやすい

Go言語製マルウェアの今後

- 複数プラットフォームへの攻撃
  - ElectroRAT *1
    - デジタルウォレットの鍵などを盗むRAT
    - Windows/Linux/macOS向けに活動
  - Chaos *2
    - 脆弱性の悪用などを行う多機能なマルウェア
    - ARM/Intel/MIPS/PowerPC向けが存在

- 今後について
  - 2022年にも多く登場している
    - Chaos、Nerbian、Agenda、etc
  - 今後もGo言語製マルウェアが継続しそう

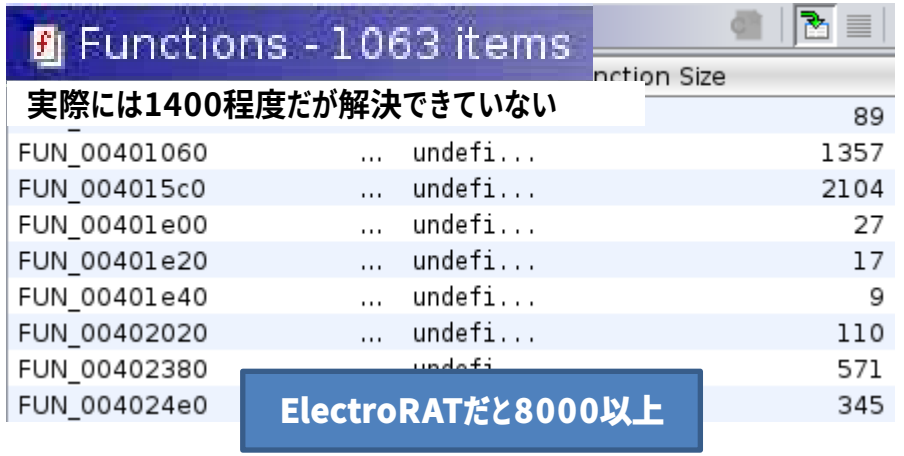
*1 https://www.intezer.com/blog/research/operation-electrorat-attacker-creates-fake-companies-to-drain-your-crypto-wallets/
*2 https://blog.lumen.com/chaos-is-a-go-based-swiss-army-knife-of-malware/

Go言語製マルウェア解析の問題点

- 既存のマルウェアはC/C++製のバイナリが多い

- Go言語製バイナリとC/C++製バイナリは大きく異なる
  - 関数量が膨大
    - Hello Worldでも関数が1000以上ある
    - Windows APIなどは動的に呼び出される
      - 関数の機能が分かりづらい
  - 独自のデータ構造
    - 文字列がヌル終端ではなく構造体で表現される
    - interface{}などの独自の型が利用される
  - 独自の呼び出し規約
    - 引数や戻り値に使用されるレジスタ等が異なる

Hello Worldの関数



Hello Worldの文字列

```
                        s_Hello_World_Join_ControlMeetei_M_0049a709
0049a709 48 65 6c ...   ds          "Hello World\nJoin_ControlM
```

Hello Worldの関数呼び出し

```
0048243b 48 8d 05 ...   LEA         RAX,[PTR_DAT_004b9178]
00482442 48 8d 0d ...   LEA         RCX,[DAT_0049a709]
00482449 bf 0c 00 ...   MOV         EDI,0xc
0048244e 31 f6          XOR         ESI,ESI
00482450 45 31 c0       XOR         R8D,R8D
00482453 4d 89 c1       MOV         R9,R8
00482456 e8 05 95 ...   CALL        FUN_0047b960
```

ツールによる対応と問題点

- 既存ツールの機能
  - 関数名の解決
    - これが分からないとかなり解析が困難
  - 文字列抽出、etc

- 問題点
  - 既存ツールを長い間使い回せない
    - Go言語はバージョンアップで構造が変化することがある
      - 新たなバージョンでは関数名などが取り出せない場合も
  - マルウェアChaChiなどの難読化が施されたマルウェアが存在する
    - Go言語用の難読化ツールが利用される

- 既存ツールをそのまま使用するだけでは解決が難しい
  - 適宜改造などが必要になる

紹介する内容

- Go言語製マルウェア　解析の基礎
  - Go言語特有の構造
  - 解析フロー
  - マルウェアを用いた事例

- Go言語製マルウェア　発展的な解析
  - 既存ツールが参照するGo言語製バイナリのデータ
  - Go言語のバージョンアップへの既存ツールの改造による対応
  - 難読化対策

GO言語製マルウェア解析の基礎

Go言語特有の構造

- 解析方法紹介の前にGo言語特有の構造について紹介

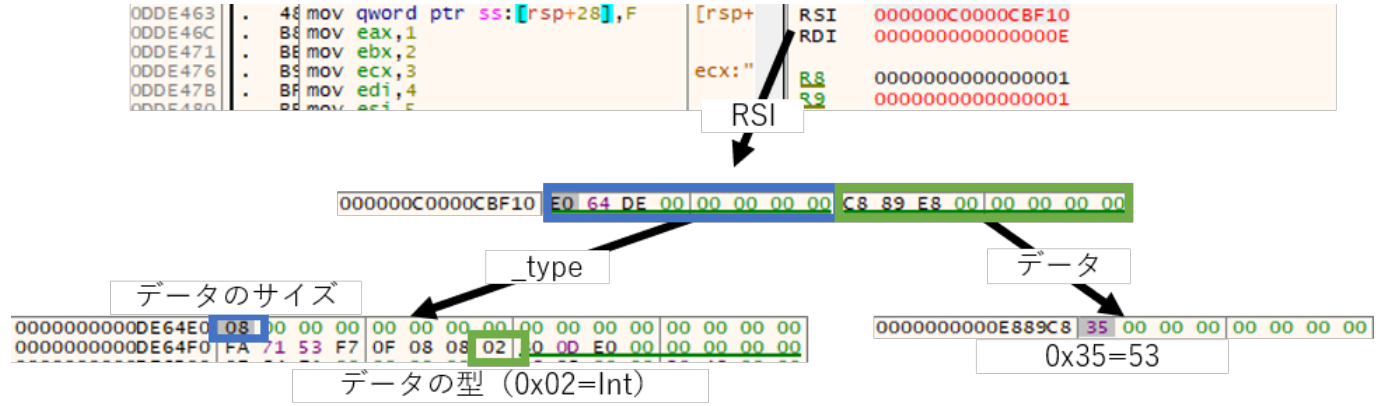
- データ構造関連
  - string
  - interface{ }
  - slice
  - map
- 関数関連
  - 呼び出し規約

string & interface{ }

- **string: 文字列**
  - **構造**
    - **__data**：文字列へのポインター
      - **ヌル終端ではない**
    - **__len**：文字列長
      - **サイズはポインターと同等**

```
struct string {
    char* __data;
    int32 or int64 __len;
}
```

- **interface{ }**：複数のデータ型を許容
  - **構造**
    - **tab**：データの型情報へのポインター
    - **data**：データへのポインター



slice

- slice：動的にサイズを変更可能な配列
- 構造
  - array：配列へのポインター
  - len：配列の長さ
    - サイズはポインターと同等
  - cap：メモリ確保されている配列の長さ
    - サイズはポインターと同等

```
struct slice {
    void* array;
    int32 or int64 len;
    int32 or int64 cap;
}
```

- 操作関数
  - func growslice(et *_type, old slice, cap int) slice [*1]
    - 第2引数に指定したsliceを第3引数で指定したサイズ以上のメモリを保持するsliceにコピーする
    - sliceへの値の追加はこのgrowsliceの戻り値に対して行われる
    - Go 1.20からは引数の構成が異なる

- 可変長引数もsliceと同様の構造で表現
  - func Command(name string, arg ...string) *Cmd [*2]
    - Command(“cmd”, “/C”, “bin”) == Command(“cmd”, [2]string{“/C”, “bin”})

*1 https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/slice.go#L178
*2 https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/os/exec/exec.go#L271

map

- **map**：連想配列

- **操作関数（キーが文字列の場合）**
  - **func mapaccess2_faststr(t *maptype, h *hmap, ky string) (unsafe.Pointer, bool)** [*1]
    - **第1引数にmapの型、第2引数にアクセス先map、第3引数にキーを指定する**
    - **キーに対応する値と成否が返る**
  - **func mapassign_faststr(t *maptype, h *hmap, s string) unsafe.Pointer** [*2]
    - **第1引数にmapの型、第2引数に操作対象map、第3引数にキーを指定する**
    - **戻り値に対して値を割り当てる**

*1 https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/map_faststr.go#L108
*2 https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/map_faststr.go#L203

呼び出し規約

- Windowsのamd64での既存との差
  - i386であればスタックが利用される

	Microsoft x64 呼び出し規約	Go言語(>=Go 1.17)	Go言語(<Go 1.17)
引数	rcx,rdx,r8,r9,スタック	rax,rbx,rcx,rdi,rsi,r8,r9,r10,r11,スタック	スタック
戻り値	rax	同上	同上

- 引数・戻り値の割り当て
  - Go 1.17以上 & amd64: 引数も戻り値もraxから順番に使用される
  - Go 1.16以下 ¦ i386: 戻り値は引数が使用したスタックの末尾から使用される
- Go言語のソースコードでの定義
  - 以下の「paramIntReg＜アーキテクチャ名＞」に定義されている
    - https://raw.githubusercontent.com/golang/go/go1.19.1/src/cmd/compile/internal/ssa/opGen.go

ツールによる対応

- https://github.com/mooncat-greenpy/Ghidra_GolangAnalyzerExtension
  - 関数名の解決
  - 引数・戻り値で利用されるバイト数やレジスタの解決
  - データ型名やフィールド情報の抽出
  - 文字列の抽出
  - アセンブリに対応するソースコードファイル名
  - アセンブリに対応するソースコード行番号

ツールなし

```
puVar2 = (undefined8 *)FUN_00410500((ulonglong)&DAT_004a6bab);
puVar2[1] = 6;
if (_DAT_005924c0 == 0) {
  *puVar2 = &DAT_004a70e3;
}
```

ツールあり

```
runtime.mapassign_faststr(&datatype.Map.map[string]string,extraout_RAX_03,"go",2);
extraout_RAX_04[1] = (char *)0x6;
if (_DAT_005924c0 == 0) {
  *extraout_RAX_04 = "golang";
}
```

- 以降のGhidraの画面ではGhidraAnalyzerExtensionを利用した結果を表示

解析フロー

- main (main.main) 関数を確認
  - アセンブリとデコンパイル結果を俯瞰する程度
    - 呼び出される関数など
  - 存在するなら初期化に利用されるinit関数も確認
- 関数名・ファイル名・構造体列確認
  - 明らかに怪しい名前
  - 利用しているOSSライブラリ
- 上記情報から機能推定及び推定内容の確認
  - 利用有無や推定通りの機能であるか
- 詳細解析

補足
main関数が分からない場合はエントリーポイントから辿る
ここでは解説しないので以下のブログを参考にして欲しい
https://engineers.ffri.jp/entry/2022/04/11/141131

init関数についてはAppendixで説明

事例：ElectroRAT

- Windows、Linux、macOSで活動していたRAT

- 解析フロー
  - GolangAnalyzerExtensionをGhidraにインストール・実行
  - main.mainの確認
  - 関数名・ファイル名リストから保有機能を推測
    - 構造体の確認は割愛
  - 上記推測を簡易的に検証
  - 詳細解析

ElectroRAT：main.mainの確認

- 注目すべき関数呼び出し抜粋
  - net/http.(*Client).Get
  - io/ioutil.readAll
  - main.registerUser
  - main.setAutostart
  - main.StartKeyLogger
  - main.socketConnect

- 推測
  - 自動実行機能があり、主な機能はC2との通信をベースに行う

- main.initは割愛
  - mapの作成などが行われている

main.mainのデコンパイル結果（ツール適用済み、手動での修正なし）

ElectroRAT：関数名・ファイル名確認

- 予想されるマルウェアの機能
  - **情報窃取**
    - github.com/gorilla/websocket
    - main.uploadFile
    - main.uploadFolder
  - **自動起動設定**
    - github.com/ProtonMail/go-autostart
    - main.HideFile
    - main.copyAppToStartDir
  - 端末情報取得
  - キーロガー
  - スクリーンショット取得
  - etc

- 推定した機能の呼び出し元と推定が正しいかを簡易的に確認する
  - 今回は上記の太字のみ確認

ファイル名リスト
（Window → GolangAnalyzerExtension）

functions | filenames | datatypes

Filename
C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/screenshot.go
C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/socket.go
C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/startup.go
C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/uploadFile.go
C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/uploadFolder.go
C:/Users/exec/go/src/github.com/denisbrodbeck/machineid/id.go
C:/Users/exec/go/src/github.com/denisbrodbeck/machineid/id_windows.go
C:/Users/exec/go/src/github.com/gorilla/websocket/client.go

関数名リスト

functions | filenames | datatypes

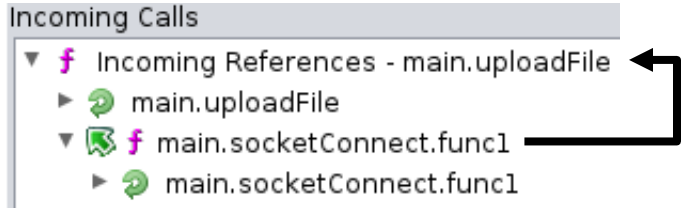
Location	Function Name	Args Size	Size
007ae860	main.getMachineID	16	176
007aebc0	main.getOsInfo	24	384
007aee50	main.getProcessList	32	752
007b03d0	main.getUserPath	16	208
007ae500	main.HideFile	32	224
007b3160	main.init	0	1328
007aed40	main.killProcessWindows	16	272
007ae910	main.main	0	688
007af140	main.registerUser	16	1056
007af560	main.RunBinary	64	384
007b0900	main.setAutostart	0	560
007afa90	main.socketConnect	16	2368

ElectroRAT：情報窃取の呼び出し元

- gorilla/websocket.(*Dialer).Dial
  - (*Dialer).Dialが最初に呼ばれる
  - 呼び出し元：main.socketConnect
    - main.mainから呼ばれる
- main.upload(File¦Folder)
  - 呼び出し元：両方ともmain.socketConnect.func1
- main.socketConnect.func1
  - Call Treesでは分からない
    - 並行処理を実行するGoroutineが使われているため
      - 実行される関数は引数を経由して渡される
  - 呼び出し元：main.socketConnect
    - アドレス参照などから調査

main.socketConnectのCall Trees

main.uploadFileのCall Trees



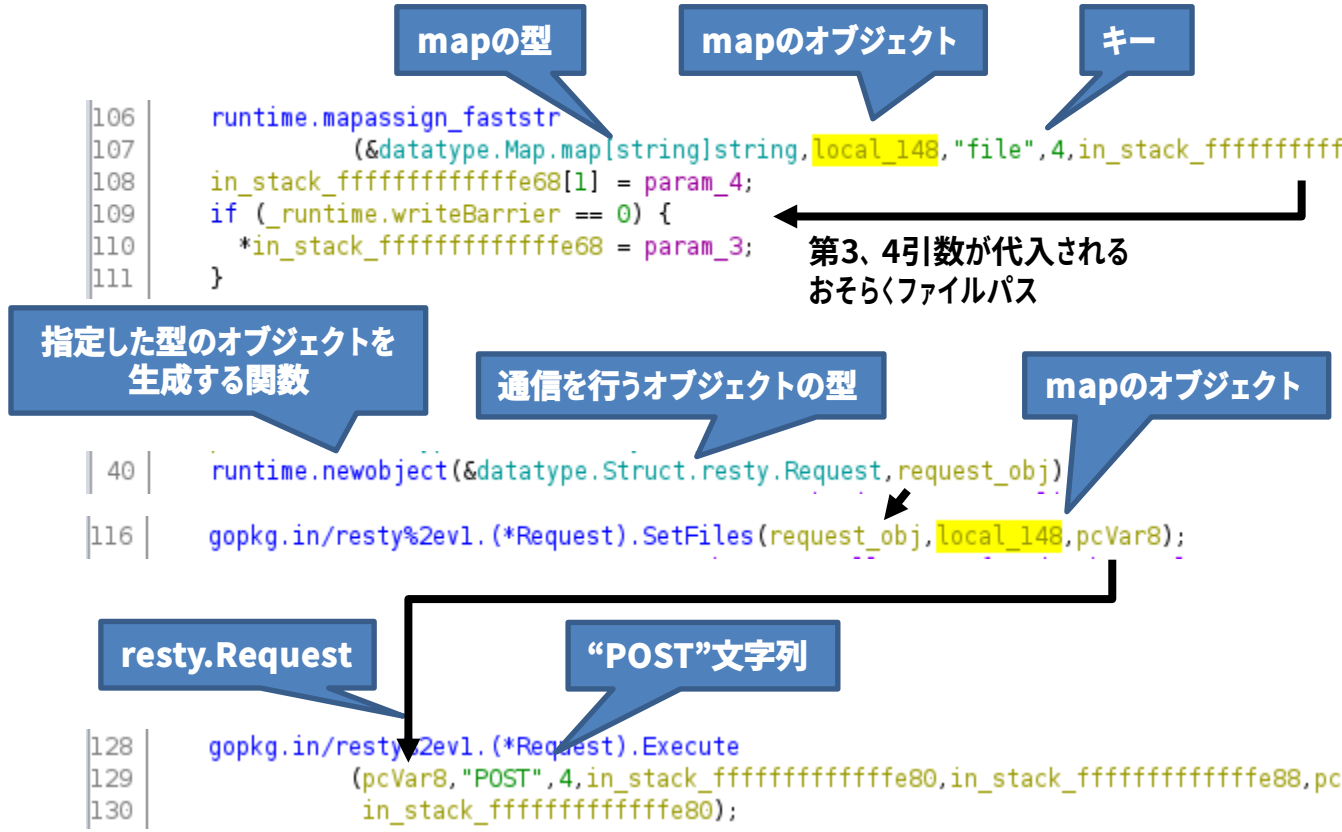
main.socketConnect関数

ElectroRAT：情報窃取処理

- mapに以下が追加される
  - キー：”file”
  - 値：main.uploadFileの引数
    - アップロードするファイルパス

- ファイルの送信による情報窃取
  - SetFilesでファイルパスを含むmapを指定
    - 送信ファイル設定
  - Executeで外部に送信
    - POSTコマンド

main.uploadFileのデコンパイル結果



ElectroRAT：自動起動設定

- main.copyAppToStartDirで作成されたファイルに自動起動設定
  - OSSが利用されている
    - https://github.com/ProtonMail/go-autostart

main.setAutostartのデコンパイル結果

IsEnabled の成否確認

```
53                  /* C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/startup.go:69 */
    github.com/ProtonMail/go-autostart.(*App).IsEnabled(&app,in_stack_rsp+8);
    if ((char)in_stack_rsp+8 == '\0') {
                    /* C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/startup.go:70 */
      github.com/ProtonMail/go-autostart.(*App).Enable(&app,in_stack_rsp+8,in_stack_rsp+10);
```

- go-autostart
  - cgoを利用している
    - Go言語からC言語を使用
  - COMを利用してスタートアップフォルダにショートカットファイルが作成される

go-autostartのソースコード

```
9   uint64_t CreateShortcut(char *shortcutA, char *path, char *args) {
10          IShellLink*   pISL;
11          IPersistFile* pIPF;
12          HRESULT       hr;
13
14          CoInitializeEx(NULL, COINIT_MULTITHREADED);
15
16          // Shortcut filename: convert ANSI to unicode
17          WORD shortcutW[MAX_PATH];
18          int nChar = MultiByteToWideChar(CP_ACP, 0, shortcutA, -1, shortcutW, MAX_PATH);
19
20          hr = CoCreateInstance(&CLSID_ShellLink, NULL, CLSCTX_INPROC_SERVER, &IID_IShellLink, (LPV
```

ElectroRATで利用されるOSSライブラリ抜粋

- github.com/ProtonMail/go-autostart
  - 自動起動設定
- github.com/gorilla/websocket
  - ソケット通信
- github.com/matishsiao/goInfo
  - 端末の情報を取得
- github.com/mitchellh/go-ps
  - プロセスリストを取得
- etc

- どれもWindows、Linux、macOSに対応
- OSSライブラリの処理はソースコードを読めばよい

良く利用されるOSSライブラリ

- 複数のプラットフォームを対象としたライブラリ
  - github.com/denisbrodbeck/machineid
    - 端末を識別するIDを取得
  - github.com/shirou/gopsutil
    - プロセス・システム関連の情報を取得
  - github.com/gorilla/websocket
    - ソケット通信
  - github.com/kardianos/service
    - サービスの操作を提供
- 特定のプラットフォームのみ対象だがGo言語での実装に工数がかかる処理のライブラリ
  - github.com/go-ole/go-ole
    - Windows COMのラッパー
  - github.com/lxn/win
    - Windows APIのラッパー

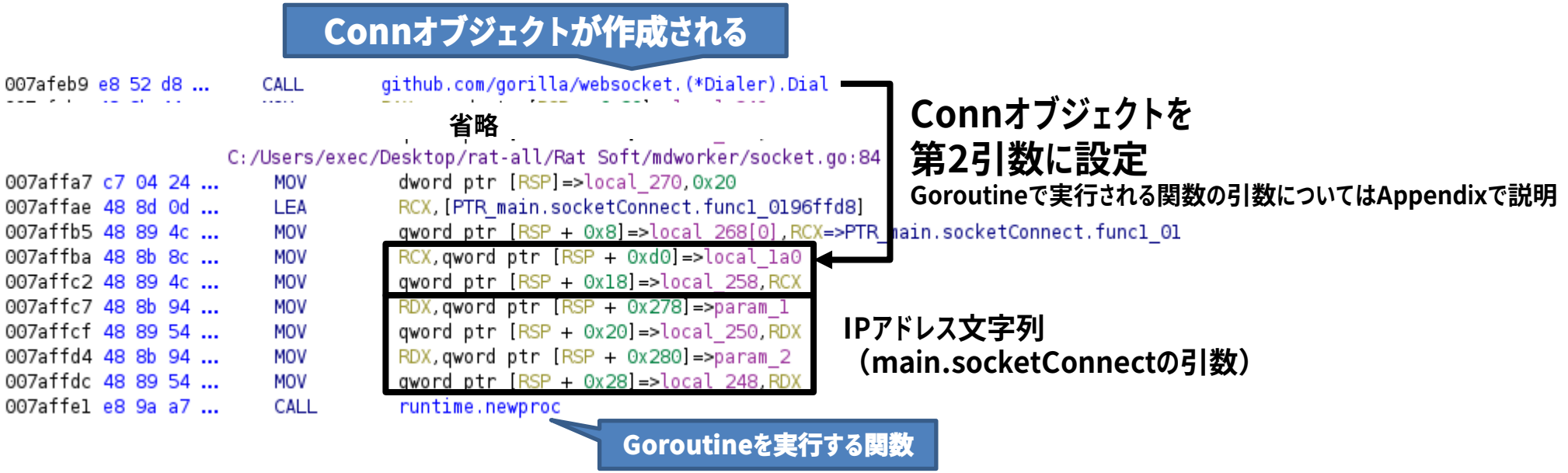
詳細

- これまでの調査でマルウェアの機能が大雑把に把握できた
- これらの情報を基により詳細な解析を行う
  - ここでは以下のみを扱う

- C2との通信
  - main.socketConnectでマルウェアのメインの処理を行っていると思われる
    - main.registerUserによるC2への感染通知やmain.setAutostartでの自動起動設定などの前準備の後に、main.socketConnectでソケット通信が行われるため

main.socketConnectの解析

- 主な処理
  - 通信の確立
    - 通信に必要なConnオブジェクトを作成する
  - main.socketConnect.func1の呼び出し
    - 第2引数にConnオブジェクトを渡す



main.socketConnect.func1

- 処理の流れ
  - C2から命令受信
  - 命令のパース
  - 命令の実行
  - C2に結果を送信

命令受信 & 命令のパース

- 受信したデータはmain.Commandに変換
  - Typeフィールドが命令として扱われる

main.Command構造体（Data Type Manager）

Structure Editor - main.Command (170cb5ea1a6b4af3c27

Offset	Length	Mnemonic	DataType	Name
0	16	string.conflict	string.conflict	Type
16	8	ulonglong	ulonglong	UID
24	104	struct { Folder string; FileName string; Fil	struct { Fold...	Data

main.socketConnect.func1関数

```
007b17dc 48 8b 84 ...   MOV     RAX,qword ptr [RSP + 0x200]=>param_2
                        MOV     qword ptr [RSP]=>local_1f0,RAX
                        CALL    github.com/gorilla/websocket.(*Conn).ReadMessage
                                省略
007b18d3 48 89 04 24    MOV     qword ptr [RSP],RAX
                        LEA     RAX,[datatype.Ptr.*main.Command]
                        MOV     qword ptr [RSP + 0x8],RAX
                        MOV     RCX,qword ptr [RSP + 0x110]
007b18eb 48 89 4c ...   MOV     qword ptr [RSP + 0x10],RCX
                        CALL    encoding/json.(*Decoder).Decode
C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/socket.go:99
                        MOV     RAX,qword ptr [RSP + 0x110]
007b18fd 48 8b 48 08    MOV     RCX,qword ptr [RAX + 0x8]
007b1901 48 89 8c ...   MOV     qword ptr [RSP + 0x88],RCX
007b1909 48 8b 10       MOV     RDX,qword ptr [RAX]
007b190c 48 89 94 ...   MOV     qword ptr [RSP + 0x100],RDX
                                省略
C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/socket.go:203
LAB_007b1d0b
007b1d0b 0f 85 70 ...   JNZ     LAB_007b2381
007b1d11 48 bb 52 ...   MOV     RBX,"mmoc nuR"
                        CMP     qword ptr [RDX],RBX
                        JNZ     LAB_007b2378
                        CMP     word ptr [RDX + 0x8],"na"
                        JNZ     LAB_007b236f
007b1d30 80 7a 0a 64    CMP     byte ptr [RDX + 0xa],'d'
007b1d34 0f 84 61 ...   JZ      LAB_007b229b
C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/socket.go:99
007b1d3a 48 83 f9 0b    CMP     RCX,0xb
```

引数のConnを利用 受信データを取得

Decoderオブジェクト 受信データを保持

&main.Command の型

&main.Command のオブジェクト

Type文字列

命令判定 Run command

命令の実行 & 結果の送信

- main.Command構造体からCommand文字列を取り出して実行
- 結果をJSON形式に変換して外部へ送信

main.Command構造体

Structure Editor - main.Command (170cb5ea1a6b4af3c27

Offset	Length	Mnemonic	DataType	Name
0	16	string.conflict	string.conflict	Type
16	8	ulonglong	ulonglong	UID
24	104	struct { Folder string; FileName string; Fil	struct { Fold...	Data

Dataフィールドの構造体

Structure Editor - struct { Folder string; FileName string; FilePath string; Name strir

Offset	Length	Mnemonic	DataType	Name
0	16	string.conflict1	string.conflict1	Folder
16	16	string.conflict1	string.conflict1	FileName
32	16	string.conflict1	string.conflict1	FilePath
48	16	string.conflict1	string.conflict1	Name
64	16	string.conflict1	string.conflict1	Command
80	16	string.conflict1	string.conflict1	FolderPath
96	8	longlong	longlong	Port

main.socketConnect.func1関数

```
C:/Users/exe                                        rker/socket.go:204
LAB_007b229b                          RAX = &main.Command
007b229b 48 8b 48 60     MOV          RCX,qword ptr [RAX + 0x60]
007b229f 48 8b 50 58     MOV          RDX,qword ptr [RAX + 0x58]
                         MOV          qword ptr [RSP]=>local_1f0,RDX
                         MOV          qword ptr [RSP + 0x8]=>local_1f0[8],RCX
007b22ac e8 2f b7 ...    CALL         main.ExecConsole
007b22b1 48 8b 44 ...    MOV          RAX,qword ptr [RSP + 0x18]=>local_1e0[8]
                         MOV          RCX,qword ptr [RSP + 0x10]=>local_1e0[0]
                                  省略
007b230f e8 6c 23 ...    CALL         encoding/json.Marshal
007b2314 48 8b 44 ...    MOV          RAX,qword ptr [RSP + 0x18]=>local_1e0[8]
007b2319 48 8b 4c ...    MOV          RCX,qword ptr [RSP + 0x10]=>local
007b231e 48 8b 54 ...    MOV          RDX,qword ptr [RSP + 0x20]=>local
                         C:/Users/exec/Desktop/rat-all/Rat Soft/mdworker/
007b2323 48 8b 9c ...    MOV          RBX,qword ptr [RSP + 0x200]=>param_2
007b232b 48 89 1c 24     MOV          qword ptr [RSP]=>local_1f0,RBX
007b232f 48 c7 44 ...    MOV          qword ptr [RSP + 0x8]=>local_1f0[8],0x1
007b2338 48 89 4c ...    MOV          qword ptr [RSP + 0x10]=>local_1e0[0],RCX
007b233d 48 89 44 ...    MOV          qword ptr [RSP + 0x18]=>local_1e0[8],RAX
007b2342 48 89 54 ...    MOV          qword ptr [RSP + 0x20]=>local_1d0,RDX
007b2347 e8 74 1c ...    CALL         github.com/gorilla/websocket.(*Conn).WriteMessage
```

オフセット：0x58

Commandフィールドを取得

Commandを実行し結果を返す

戻り値の実行結果

JSON形式

main.Commandの先頭からのオフセット：0x58

検体の比較による解析の効率化

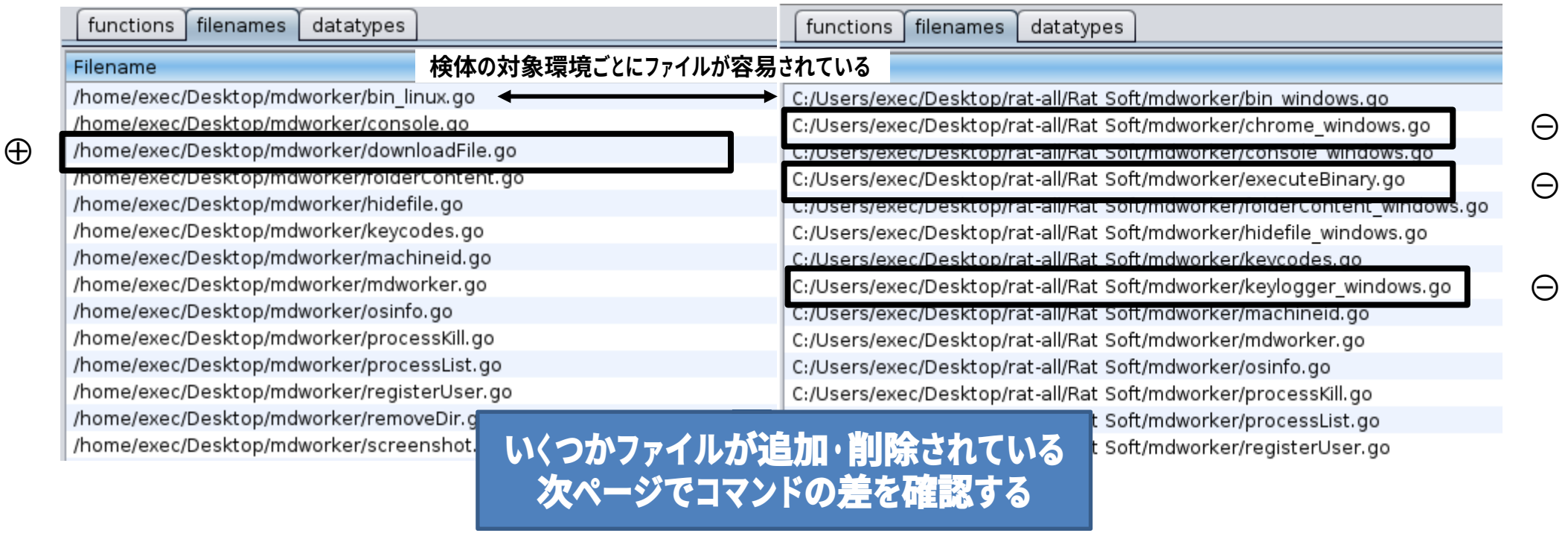
- 解析対象の検体と既存の検体や同一インシデント内で観測された検体との比較について紹介する

- 同一処理が存在
  - 解析を短縮可能
- 類似しているが異なる処理が存在
  - 攻撃の動向把握
  - 既存検体が防御できている場合に新規検体に感染してしまった原因調査

- Go言語のバイナリではソースコードのファイル名や行番号が取得できるため、詳細に差分を追える

新旧検体の比較

- 以下の検体を比較
  - 左：e9b83d5cdefd4486b32a927d7505cdeebb43e6977759ba069d9373e46ca7d0f2（新）
  - 右：170cb5ea1a6b4af3c27358ba267a1309ed5118481619fc874f717262cb91fb77（旧）



検体間のコマンドの変化

- 変化があった場所を主に解析すれば良い

コマンドが実装されているsocket.goファイルの行数を比較

コマンド	e9b83d…の行数（新）	170cb5…の行数（旧）
Get folder content	130~ 133	100~ 103
Keylogger	**135~ 149**	**105~ 118**
Screenshot	151~ 164	119~ 128
Camera photo		129~ 134
Processes list	166~ 169	135~ 137
Download file	171~ 179	138~ 145
Download folder	181~ 189	147~ 153
Add file	**191~ 205**	
Delete file	207~ 215	154~ 161
Kill process	217~ 225	162~ 169
Chrome passwords		**170~ 176**
Run service	227~ 235	177~ 202
Run command	237~ 245	203~ 210

keylogger_windows.goが削除されていたがコマンドは存在

追加されたdownloadFile.goに関連するコマンドが追加

削除されたchrome_windows.goに関連するコマンドが削除

GO言語製マルウェア
発展的な解析

Goバージョンアップへの対応

- 今まで利用できていたツールが突然上手く動作しなくなる
  - Go言語のバージョンアップによってツールが参照するデータが変更される
  - 例：ElectroRATは解析できるが、より新しいChaosは解析できないなど

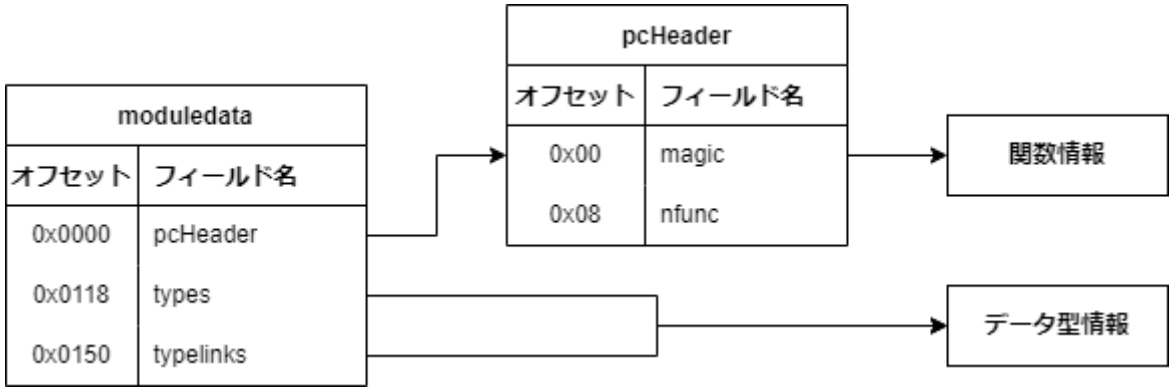
- 説明の流れ
  - Go言語製バイナリのメタデータ構造説明
    - 既存ツールが利用する情報
    - 過去のバージョンアップでの変更点
  - ツールの改造事例紹介

Go言語製バイナリのメタデータ紹介

- ツールが良く参照するデータ
  - 関数情報
    - 関数名などの情報を含む
    - スタックトレースの表示などで使用される
  - データ型情報
    - データ型名やフィールド情報を含む
    - オブジェクト生成やinterface{}、mapでの型指定に使用される

メタデータの構成概要

- 紹介するメタデータを管理する構造体
  - moduledata
    - pcHeaderとデータ型情報へのリンクを保持
  - pcHeader
    - 関数情報へのリンクを保持



64bit向けバイナリでの一部フィールドのみ表示している、以降の図も同様

メタデータの構成要素

- moduledata *
  - pcHeader
    - pcHeaderへのポインター
  - text
    - コードへのポインター
    - 基本的に.textセクションと同一
  - types
    - 型情報のベースアドレス
  - typelinks
    - typesから型情報へのオフセットの配列(slice)

```
415  type moduledata struct {
416          pcHeader     *pcHeader
417          funcnametab  []byte
418          cutab        []uint32
419          filetab      []byte
420          pctab        []byte
421          pclntable    []byte
422          ftab         []functab
423          findfunctab  uintptr
424          minpc, maxpc uintptr
425
426          text, etext           uintptr
427          noptrdata, enoptrdata uintptr
428          data, edata           uintptr
429          bss, ebss             uintptr
430          noptrbss, enoptrbss   uintptr
431          end, gcdata, gcbss    uintptr
432          types, etypes         uintptr
433          rodata                uintptr
434          gofunc                uintptr // go.func.*
435
436          textsectmap []textsect
437          typelinks   []int32 // offsets from types
438          itablinks   []*itab
439
440          ptab []ptabEntry
441
442          pluginpath string
443          pkghashes  []modulehash
```

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/symtab.go#L415-L457

メタデータの構成要素

- moduledata *
  - pcHeader
    - pcHeaderへのポインター
  - text
    - コードへのポインター
    - 基本的に.textセクションと同一
  - types
    - 型情報のベースアドレス
  - typelinks
    - typesから型情報へのオフセットの配列(slice)

```
415  type moduledata struct {
416          pcHeader     *pcHeader
417          funcnametab  []byte
418          cutab        []uint32
424          minpc, maxpc uintptr
425
426          text, etext           uintptr
431          end, gcdata, gcbss    uintptr
432          types, etypes         uintptr
433          rodata                uintptr
434          gofunc                uintptr // go.func.*
435
436          textsectmap []textsect
437          typelinks   []int32 // offsets from types
442          pluginpath string
443          pkghashes  []modulehash
```

フィールド追加によるオフセットの変化
Go 1.7 、Go 1.8 、Go 1.16 、Go 1.17 、Go 1.18 、Go 1.20

typesの追加
Go 1.7

ポインターからオフセットに変更
Go 1.7

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/symtab.go#L415-L457

メタデータの構成要素

- pcHeader *
  - magic、pad1、pad2
    - 固定値
  - ptrSize
    - ポインターのサイズ
  - nfunc
    - 関数の数
  - textStart
    - moduledata.textと同様
  - funcnameOffset
    - pcHeaderから関数名列へのオフセット
  - pclnOffset
    - pcHeaderから関数情報へのリンクを含む配列へのオフセット

```
395  type pcHeader struct {
396          magic          uint32  // 0xFFFFFFF0
397          pad1, pad2     uint8   // 0,0
398          minLC          uint8   // min instruction size
399          ptrSize        uint8   // size of a ptr in bytes
400          nfunc          int     // number of functions in the module
401          nfiles         uint    // number of entries in the file tab
402          textStart      uintptr // base for function entry PC offsets in this modul
403          funcnameOffset uintptr // offset to the funcnametab variable from pcHeader
404          cuOffset       uintptr // offset to the cutab variable from pcHeader
405          filetabOffset  uintptr // offset to the filetab variable from pcHeader
406          pctabOffset    uintptr // offset to the pctab variable from pcHeader
407          pclnOffset     uintptr // offset to the pclntab variable from pcHeader
408  }
```

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/symtab.go#L395-L408

メタデータの構成要素

- pcHeader *
  - magic、pad1、pad2
    - 固定値
  - ptrSize
    - ポインターのサイズ
  - nfunc
    - 関数の数
  - textStart
    - moduledata.textと同様
  - funcnameOffset
    - pcHeaderから関数名列へのオフセット
  - pclnOffset
    - pcHeaderから関数情報へのリンクを含む配列へのオフセット

Magicの変更
Go 1.16, Go 1.18, Go 1.20

```
395  type pcHeader struct {
396          magic          uint32  // 0xFFFFFFF0
397          pad1, pad2     uint8   // 0,0
398          minLC          uint8   // min instruction size
399          ptrSize        uint8   // size of a ptr in bytes
400          nfunc          int     // number of functions in the module
401          nfiles         uint    // number of entries in the file tab
402          textStart      uintptr // base for function entry PC offsets in this modul
403                                 ariable from pcHeader
404                                 e from pcHeader
405                                 ble from pcHeader
406                                 e from pcHeader
407          pclnOffset     uintptr // offset to the pclntab variable from pcHeader
408  }
```

フィールド追加によるオフセットの変化
Go 1.16 、 Go 1.18

参照方法変更
Go 1.16

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/symtab.go#L395-L408

メタデータの位置特定

- セクションを使う
  - pcHeaderは.gopclntabセクションの先頭に配置されている
  - .gopclntabセクションはWindowsでは存在しない場合もある
    - 「-ldflags=“-w –s”」オプション使用時

- pcHeaderのmagicを使う
  - 先頭のmagicフィールドには「0xfffffff0」という値がある
    - 書き換えるとエラーが発生
      - pclnOffsetなどは書き換え可能
  - この値を探索する

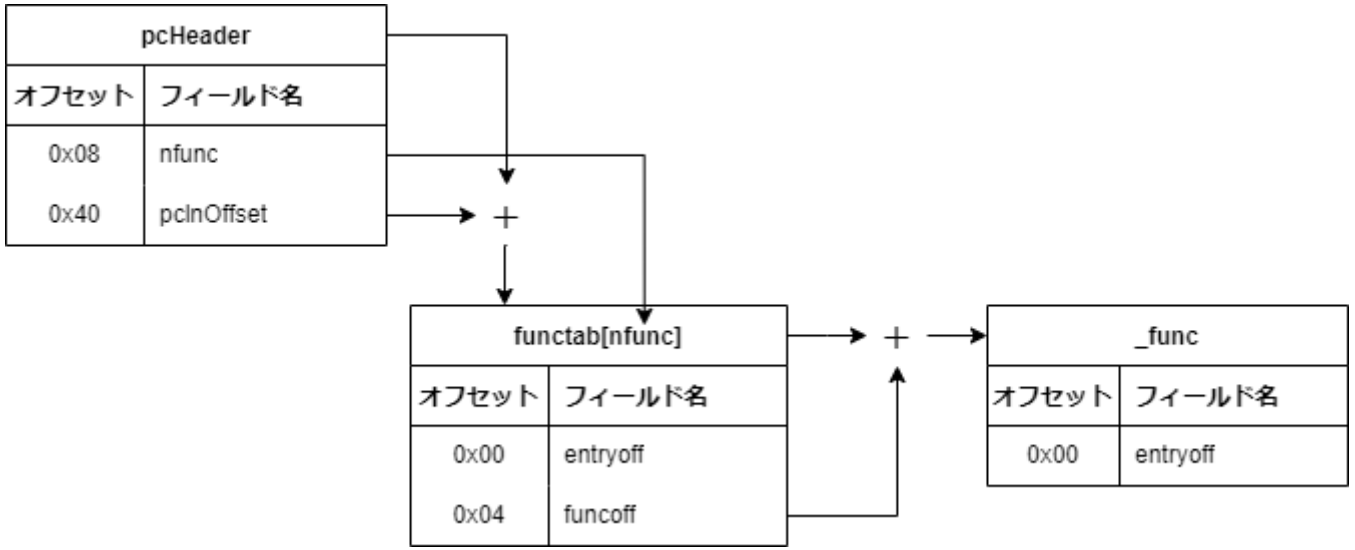
Go 1.19

- moduledataはpcHeaderへのポインターから検索する

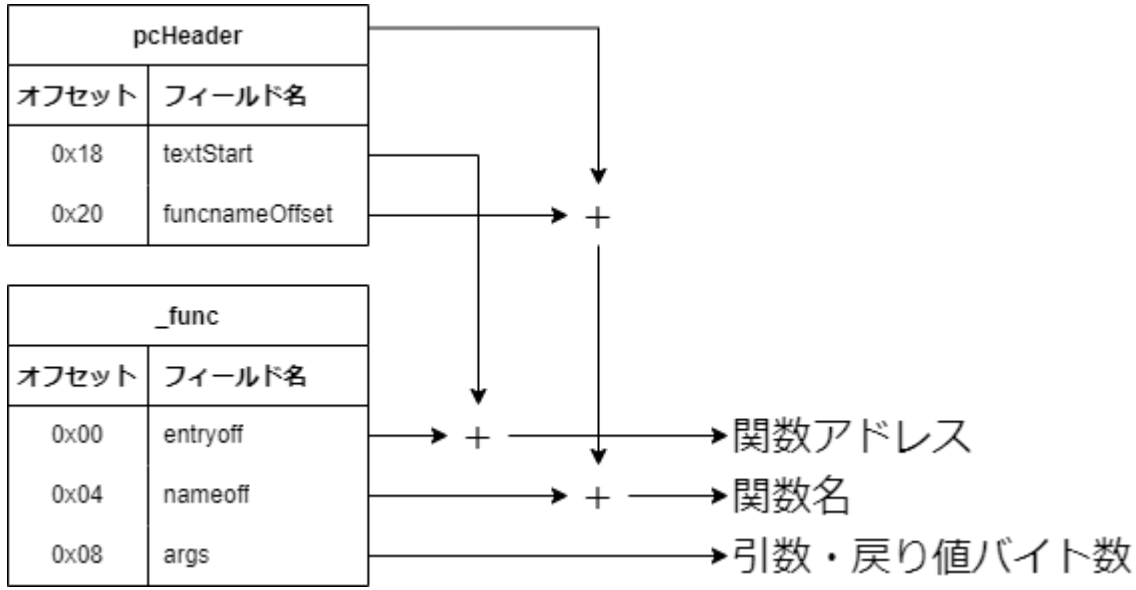
関数情報の構成

- pcHeaderから以下の情報を取り出す流れを示す
  - 関数アドレス
  - 関数名
  - 引数・戻り値バイト数

関数情報を保持する_funcを取得する流れ



_funcから関数情報を取得する流れ



関数情報の構成要素

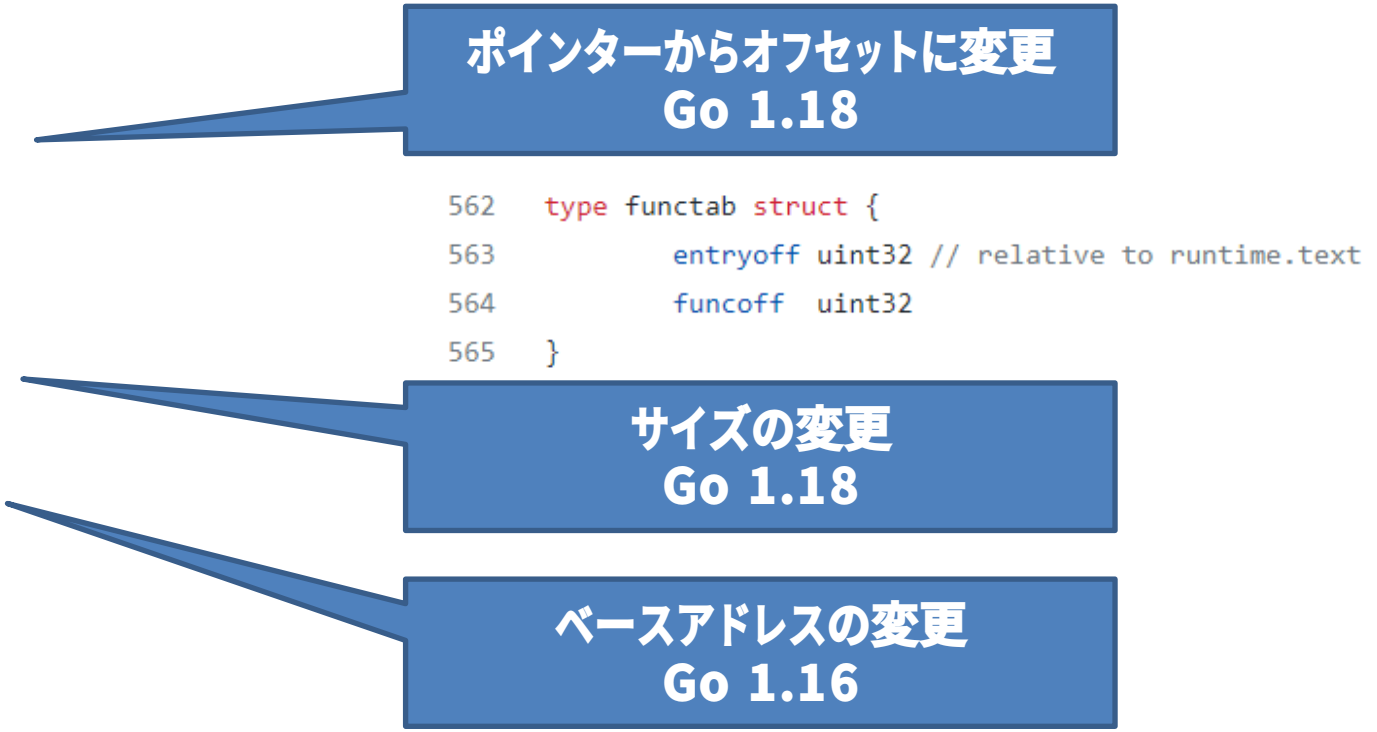
- functab *
  - entryoff
    - 関数の先頭アドレスへのオフセット
    - ベースはpcHeader.textStart
  - funcoff
    - 関数情報へのオフセット
    - ベースはfunctab配列の先頭

```
562   type functab struct {
563           entryoff uint32 // relative to runtime.text
564           funcoff  uint32
565   }
```

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/symtab.go#L562-L565

関数情報の構成要素

- functab *
  - entryoff
    - 関数の先頭アドレスへのオフセット
    - ベースはpcHeader.textStart
  - funcoff
    - 関数情報へのオフセット
    - ベースはfunctab配列の先頭



* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/symtab.go#L562-L565

関数情報の構成要素

- _func *
  - entryoff
    - 関数の先頭アドレスへのオフセット
    - ベースはpcHeader.textStart
  - nameoff
    - 関数名へのオフセット
    - pcHeaderのアドレス + pcHeader.funcnameOffset + _func.nameoff
  - args
    - 引数・戻り値で利用されるバイト数
  - pcsp、pcfile、pcln
    - アセンブリに対応するスタックサイズ、ファイル名、行番号を取り出すための情報
    - 情報の取り出し方については割愛するが後日弊社ブログにて解説予定

```
869  type _func struct {
870          entryoff uint32 // start pc, as offset from moduledata.text/pcHeader
871          nameoff  int32  // function name
872
873          args        int32  // in/out args size
874          deferreturn uint32 // offset of start of a deferreturn call instruct
875
876          pcsp      uint32
877          pcfile    uint32
878          pcln      uint32
879          npcdata   uint32
880          cuOffset  uint32 // runtime.cutab offset of this function's CU
881          funcID    funcID // set for certain special runtime functions
882          flag      funcFlag
883          _         [1]byte // pad
884          nfuncdata uint8   // must be last, must end on a uint32-aligned bou
885  }
```

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/runtime2.go#L869-L885

関数情報の構成要素

- _func *
  - entryoff
    - 関数の先頭アドレスへのオフセット
    - ベースはpcHeader.textStart
  - nameoff
    - 関数名へのオフセット
    - pcHeaderのアドレス + pcHeader.funcnameOffset + _func.nameoff
  - args
    - 引数・戻り値で利用されるバイト数
  - pcsp、pcfile、pcln
    - アセンブリに対応するスタックサイズ、ファイル名、行番号を取り出すための情報
    - 情報の取り出し方については割愛するが後日弊社ブログにて解説予定

ポインターからオフセットに変更
Go 1.18

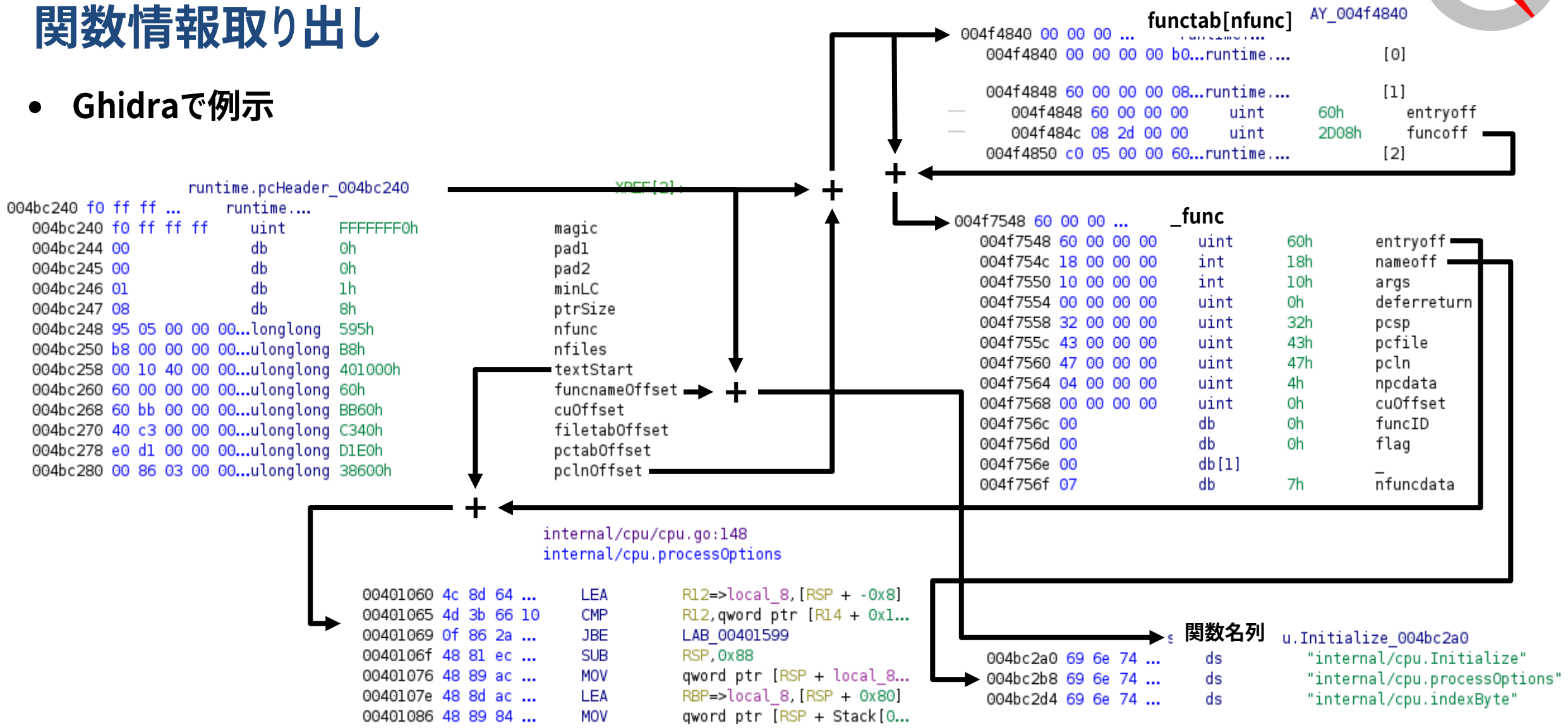
nameoffのベースアドレスの変更
Go 1.16

```
869  type _func struct {
870          entryoff uint32 // start pc, as offset from moduledata.text/pcHeader
871          nameoff  int32  // function name
872
873          args        int32  // in/out args size
874          deferreturn uint32 // offset of start of a deferreturn call instruct
875
876          pcsp      uint32
877          pcfile    uint32
             pcln      uint32
                                 et of this function's CU
                                 ecial runtime functions
883          _         [1]byte // pad
884          nfuncdata uint8   // must be last, must end on a uint32-aligned bou
885  }
```

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/runtime2.go#L869-L885

関数情報取り出し

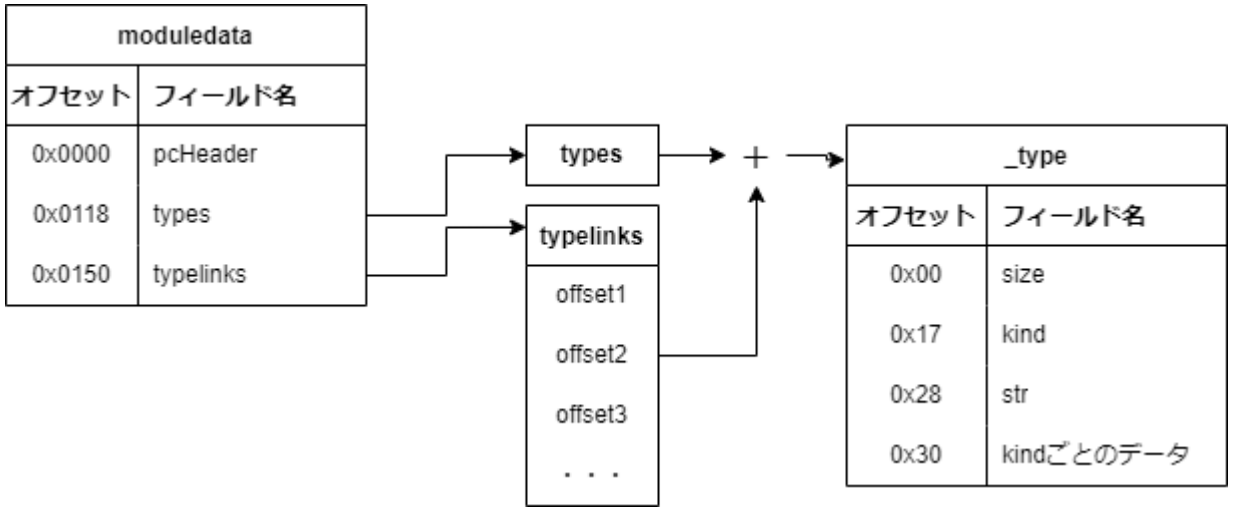
- Ghidraで例示



データ型情報の構成

- moduledataから各データ型情報を含む_type構造体を取得するための流れを下図に示す

- _type
  - データ型のサイズや種類などの情報を保持する
  - それぞれの種類ごとに固有の情報は_typeの末尾に追加される
  - オブジェクト生成やmapへのアクセス関数で使用される



データ型情報の構成要素

- _type *
  - size
    - データ型のバイト数
  - kind
    - データ型の種類
    - 値と種類の関係ー部抜粋
      - 1: Bool、2: Int、17: Array、21: Map、22: Pointer、25、Struct
  - str
    - データ型名へのオフセット
    - ベースはmoduledata.types

```
35  type _type struct {
36          size       uintptr
37          ptrdata    uintptr // size of memory prefix holding all pointers
38          hash       uint32
39          tflag      tflag
40          align      uint8
41          fieldAlign uint8
42          kind       uint8
43          // function for comparing objects of this type
44          // (ptr to object A, ptr to object B) -> ==?
45          equal func(unsafe.Pointer, unsafe.Pointer) bool
46          // gcdata stores the GC type data for the garbage collector.
47          // If the KindGCProg bit is set in kind, gcdata is a GC program.
48          // Otherwise it is a ptrmask bitmap. See mbitmap.go for details.
49          gcdata     *byte
50          str        nameOff
51          ptrToThis typeOff
52  }
```

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/type.go#L35-L52

データ型情報の構成要素

- _type *
  - size
    - データ型のバイト数
  - kind
    - データ型の種類
    - 値と種類の関係―部抜粋
      - 1: Bool、2: Int、17: Array、21: Map、22: Pointer、25、Struct
  - str
    - データ型名へのオフセット
    - ベースはmoduledata.types

```
35  type _type struct {
36          size       uintptr
37          ptrdata    uintptr // size of memory prefix holding all pointers
38          hash       uint32
39          tflag      tflag
40          align      uint8
41          fieldAlign uint8
42          kind       uint8
43          // function for comparing objects of this type
44          // (ptr to object A, ptr to object B) -> ==?
45          equal func(unsafe.Pointer, unsafe.Pointer) bool
46                                                    he garbage collector.
47                                                    , gcdata is a GC program.
48                                                    e mbitmap.go for details.
49          gcdata     *byte
50          str        nameOff
51          ptrToThis typeOff
```

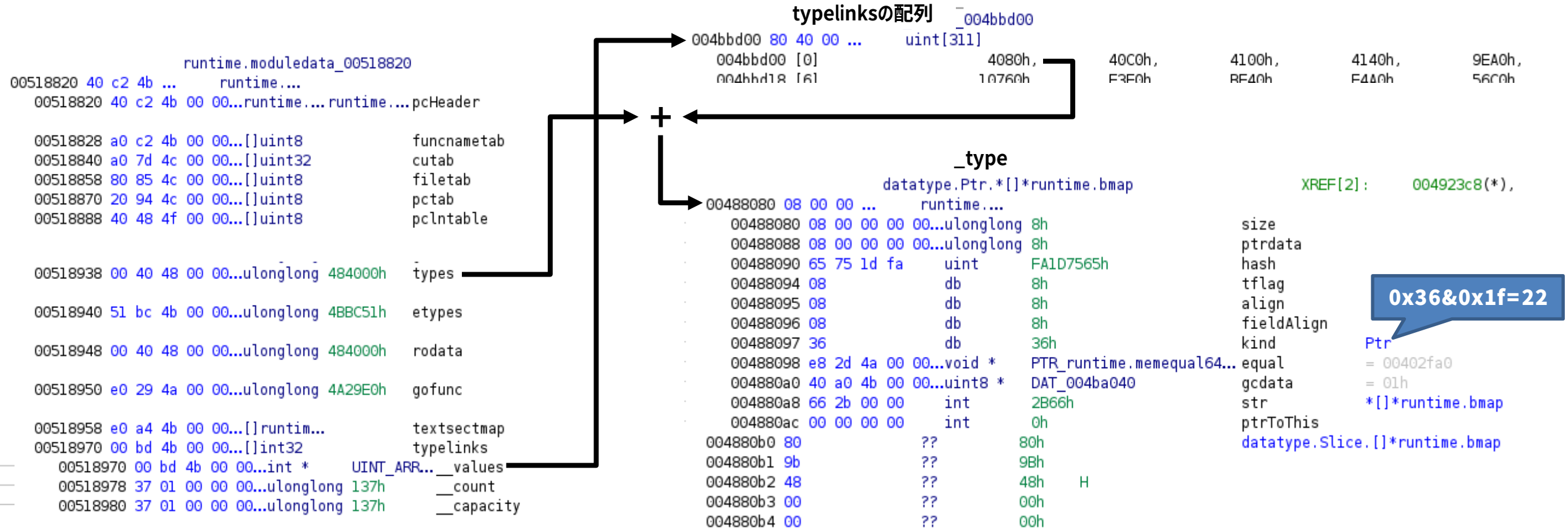
ポインターからオフセットに変更
Go 1.7

文字列の取得方法変更
Go 1.17

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/runtime/type.go#L35-L52

データ型情報取り出し

- Ghidraで例示



Goバージョンアップ対応例

- 対象検体
  - Chaos
    - ハッシュ値：
      ebe0f9855eb8f6bd980ed60c26e3a877dc1ace5d664e248bb0558996fe0bd06f
      - Goバージョン：Go 1.18.1
      - OS：Linux
      - Arch：x86

- ツール
  - https://github.com/f0rki/r2-go-helpers
    - radare2のスクリプト
    - 機能：関数名を解決

現状の対応範囲の確認

- Go 1.15でビルドしたバイナリへの適用結果
  - 関数列挙（afl）

```
0x0049b2c0    3 357          go.main.manyRet;
0x0049b440    3 383          go.main.goroutineFunc;
0x0049b5c0    9 1520         go.main.main;
0x0049bbc0    9 180          go.type..eq.[9]interface_{};
0x0049bc80    9 180          go.type..eq.[4]interface_{};
0x0049bd40    9 180          go.type..eq.[3]interface_{};
```

  - go.main.mainのアセンブリ表示（pdf）

```
0x0049b5c0      64488b0c25f8.  mov rcx, qword fs:[0xfffffffffffffff8]
0x0049b5c9      488d8424f8fe.  lea rax, qword [rsp - 0x108]
0x0049b5d1      483b4110       cmp rax, qword [rcx + 0x10]
0x0049b5d5      0f86cb050000   jbe 0x49bba6
0x0049b5db      4881ec880100.  sub rsp, 0x188
0x0049b5e2      4889ac248001.  mov qword [var_8h], rbp
0x0049b5ea      488dac248001.  lea rbp, qword [var_8h]
0x0049b5f2      488d05e33a02.  lea rax, qword [0x004bf0dc] ; "helloin
bde= c"
0x0049b5f9      48890424       mov qword [rsp], rax
0x0049b5fd      48c744240805.  mov qword [var_180h], 5
0x0049b606      e8b5f8ffff     call go.main.testFunc;
0x0049b60b      488b442418     mov rax, qword [var_170h]
0x0049b610      4803442410     add rax, qword [var_178h]
0x0049b615      4803442420     add rax, qword [var_168h]
```

一部修正

vaddrに修正

```
base_addr = gopclntab['paddr']
size_addr = base_addr + 8
```

引数に4を追加

```
name_str_offset = get_pointer_at(bas
name_addr = base_addr + name_str_off
```

現状の対応範囲の確認

- Chaosへの適用結果
  - スクリプトを実行

```
[0x080ac1b0]> #!pipe python3 gohelper.py
INFO : We're gonna 'aa' first, this might take a while
[x] Analyze all flags starting with sym. and entry0 (aa)
INFO : renaming functions based on .gopclntab section
WARNING : not using function name '
' for 0x247
WARNING : not using function name '
' for 0x40
WARNING : not using function name '
' for 0xf15e0
WARNING : not using function name '
' for 0x0
```

関数のアドレス
明らかに違う値が表示されている

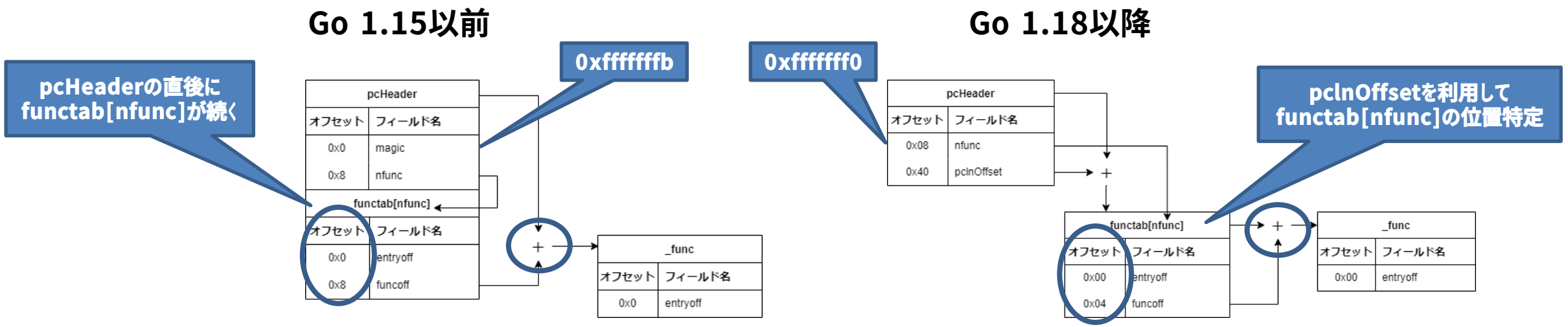
  - 関数列挙（afl）

```
[0x080ac1b0]> afl
0x080ac1b0    15 8613 -> 403  go.D32`U'>5(_G{$Dp7j?@WX[\_ab!
[0x080ac1b0]>
```

失敗している

バージョンアップによる影響

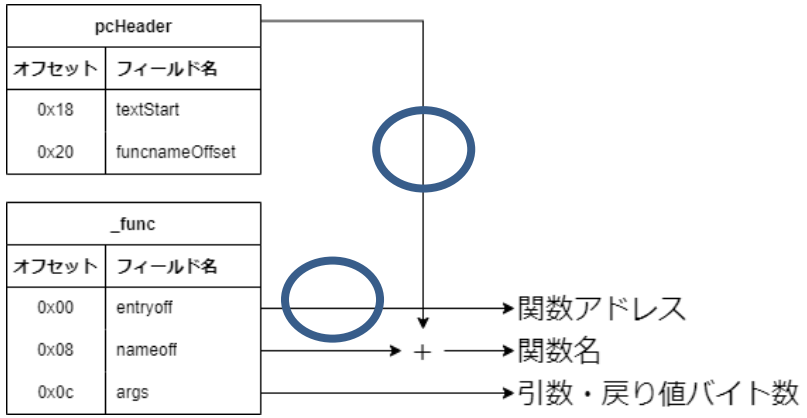
- pcHeader → _func
  - magicの値変更
  - 関数情報へのリンクを含む配列の参照方法変更
  - functab.funcoffのベースアドレス変更
  - functabのフィールドのサイズがポインターサイズから0x4に変更



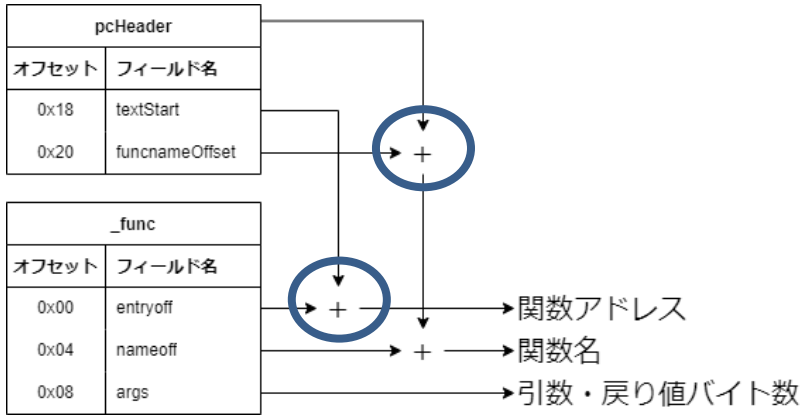
バージョンアップによる影響

- _func → 関数情報
  - _func.entryoffがポインターからオフセットに変更
  - _func.nameoffのベースアドレス変更

Go 1.15以前



Go 1.18以降



Chaosに向けたツール改造

- 抽出できない原因
  - Go 1.15～Go 1.18間の変化に対応できていない

- 修正すべき箇所 *

```
start_addr = size_addr + PTR_SIZE
end_addr = base_addr + (size * PTR_SIZE * 2)

for addr in range(start_addr, end_addr, (2 * PTR_SIZE)):
    log.debug("analyzing at 0x{:x}".format(addr))
    func_addr = get_pointer_at(addr)
    entry_offset = get_pointer_at(addr + PTR_SIZE)

    log.debug("func_addr 0x{:x}, entry offset 0x{:x}"
              .format(func_addr, entry_offset))

    name_str_offset = get_pointer_at(base_addr + entry_offset + PTR_SIZE)
    name_addr = base_addr + name_str_offset
```

functab配列位置の取得方法が異なる

functabのサイズが異なる

アドレスではなくオフセットになる

ベースアドレスが異なる

* https://github.com/f0rki/r2-go-helpers/blob/d1167e4b96ba0e3c33ee8c5e578bb3cde930324e/gohelper.py#L165-L177

改造したツールをChaosに適用

- Chaosへの適用結果
  - 関数列挙（afl）

```
0x082d8340    6 76          go.main.readPidsFromDir.func1;
0x082d8b80   64 2177        go.main.main;
0x082d9410    6 70          go.main.main.func2;
0x082d9460    6 62          go.main.main.func1;
0x082d94a0    3 66          go.main.chaos_time;
```

  - go.main.mainのアセンブリ表示（pdf）

```
0x082d8b80      658b0d000000.   mov ecx, dword gs:[0]
0x082d8b87      8b89fcffffff    mov ecx, dword [ecx - 4]
0x082d8b8d      8d442480        lea eax, dword [esp - 0x80]
0x082d8b91      3b4108          cmp eax, dword [ecx + 8]
0x082d8b94      0f865d080000    jbe 0x82d93f7
0x082d8b9a      81ec00010000    sub esp, 0x100
0x082d8ba0      b800000000      mov eax, 0
0x082d8ba5      898424f80000.   mov dword [var_8h], eax
0x082d8bac      898424fc0000.   mov dword [var_4h], eax
0x082d8bb3      c644242300      mov byte [var_ddh], 0
0x082d8bb8      e8e31d0000      call go.main.Getmyname;
0x082d8bbd      e8ce1c0000      call go.main.Getmypwd;
0x082d8bc2      8b0424          mov eax, dword [esp]
0x082d8bc5      8b4c2404        mov ecx, dword [var_fch]
```

gobfuscateによる難読化への対応

- 説明の流れ
  - gobfuscateによる難読化手法の紹介
  - マルウェアの文字列難読化解除
  - 難読化されたマルウェアでの機能推定

- 対象検体
  - ChaChi
    - ハッシュ値：
      8a9205709c6a1e5923c66b63addc1f833461df2c7e26d9176993f14de2a39d5b

gobfuscateによる難読化

- gobfuscateとは
  - 機能：文字列や関数の難読化等
  - ソースコードの修正等により難読化を実施
  - https://github.com/unixpickle/gobfuscate

難読化された関数

functions | filenames | datatypes

Location	Function Name
007c5580	main.naopcgecfflmbgciiiho.func4
007c5670	main.naopcgecfflmbgciiiho.func5
007c5760	main.naopcgecfflmbgciiiho.func6
007c5830	main.(*Bjbealpgdnpanhaihjba).mlkbhekbfckondcckjam.func1
007c58a0	main.(*Bjbealpgdnpanhaihjba).mlkbhekbfckondcckjam.func2
007c5990	main.(*Bjbealpgdnpanhaihjba).pmeanjdmanemokfpkhbm.fun
007c5a70	main.(*Bjbealpgdnpanhaihjba).eiadljbonenkfjpnbgea.func1

- ここでは文字列難読化を扱う

文字列難読化の様子

```
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:86
LAB_007be85a
     CALL        main.main.func1
     MOV         RAX,qword ptr [RSP + 0x8]
     MOV         qword ptr [RSP + 0x58],RAX
     MOV         RCX,qword ptr [RSP]
     MOV         qword ptr [RSP + 0xa8],RCX
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:94
     CALL        main.main.func2
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:78
     NOP
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:94
     MOV         RAX,qword ptr [RSP + 0x8]
     MOV         RCX,qword ptr [RSP]
/usr/local/go/src/flag/flag.go:644
     MOV         RDX,qword ptr [DAT_00c07008]
     MOV         qword ptr [RSP],RDX
     MOV         RDX,qword ptr [RSP + 0xa8]
     MOV         qword ptr [RSP + 0x8],RDX
     MOV         RDX,qword ptr [RSP + 0x58]
     MOV         qword ptr [RSP + 0x10],RDX
     MOV         byte ptr [RSP + 0x18],0x0
     MOV         qword ptr [RSP + 0x20],RCX
     MOV         qword ptr [RSP + 0x28],RAX
     CALL        flag.(*FlagSet).Bool
```

文字列のはず

文字列のはず

func Bool(name string, value bool, usage string) *bool *

* https://github.com/golang/go/blob/4a4127bccc826ebb6079af3252bc6bfeaec187c4/src/flag/flag.go#L734

gobfuscateの文字列難読化

- gobfuscateによる文字列の難読化手法
  - 無名関数で文字列を生成する
  - 文字列はXORによって暗号化されている

「golang」をgobfuscateで難読化した際に生成される関数

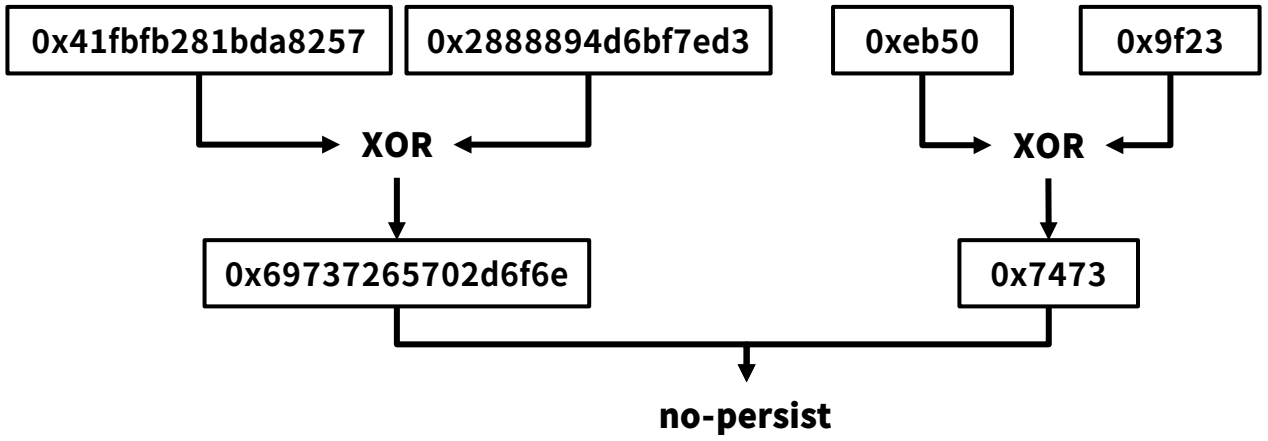
```
(func() string {
mask := []byte("\x21\x0f\xc7\xbb\x81\x86")
maskedStr := []byte("\x46\x60\xab\xda\xef\xe1")
res := make([]byte, 6)
        for i, m := range mask {
            res[i] = m ^ maskedStr[i]
        }
        return string(res)
        }())
```

無名関数が利用されている

バイト列を文字列に変換

ChaChiの文字列難読化

- 関数：**main.main.func1**



バイト列取り出し

```
                    /tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:79
007c4ab8 48 b8 a9 ...   MOV     RAX,-0x41fbfb281bda8257
007c4ac2 48 89 44 ...   MOV     qword ptr [RSP + local_1c],RAX
007c4ac7 66 c7 44 ...   MOV     word ptr [RSP + local_14],0xeb50
                    /tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:80
007c4ace 48 b8 c7 ...   MOV     RAX,-0x2888894d6bf7ed39
007c4ad8 48 89 44 ...   MOV     qword ptr [RSP + local_26],RAX
007c4add 66 c7 44 ...   MOV     word ptr [RSP + local_1e],0x9f23
                    /tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:81
007c4ae4 48 c7 44 ...   MOV     qword ptr [RSP + local_12],0x0
007c4aed 48 c7 44 ...   MOV     qword ptr [RSP + local_12+0x2],0x0
007c4af6 31 c0          XOR     EAX,EAX
                    /tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:82
007c4af8 eb 13          JMP     LAB_007c4afa
                    /tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:83
007c4afa 0f b6 4c ...   MOVZX   ECX,byte ptr [RSP + RAX*0x1 + 0x3c]
007c4aff 0f b6 54 ...   MOVZX   EDX,byte ptr [RSP + RAX*0x1 + 0x32]
007c4b04 31 d1          XOR     ECX,EDX
007c4b06 88 4c 04 46    MOV     byte ptr [RSP + RAX*0x1 + 0x46],CL
                    /tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:82
007c4b0a 48 ff c0       INC     RAX
                    LAB_007c4...
007c4b0d 48 83 f8 0a    CMP     RAX,0xa
007c4b11 7c e7          JL      LAB_007c4afa
                    /tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:85
007c4b13 48 c7 04 ...   MOV     qword ptr [RSP]=>local_58,0x0
007c4b1b 48 8d 44 ...   LEA     RAX=>local_12,[RSP + 0x46]
007c4b20 48 89 4
007c4b25 48 c7 4
007c4b2e 48 c7 4
007c4b37 e8 a4 92 ...   CALL    runtime.slicebytetostring
007c4b3c 48 8b 44 ...   MOV     RAX,qword ptr [RSP + local_38]
007c4b41 48 8b 4c ...   MOV     RCX,qword ptr [RSP + local_30]
007c4b46 48 89 44 ...   MOV     qword ptr [RSP + param_1],RAX
007c4b4b 48 89 4c ...   MOV     qword ptr [RSP + param_2],RCX
```

バイト列をXOR

ループの回数

バイト列を文字列へ変換し戻り値に設置

補足：無名関数

- Go言語の無名関数
  - 例：f := func(arg string) string { return “arg: “ + arg }

- アセンブリでの表現
  - 関数名は「<実装元の関数名>.func%d」という形式
    - この形式の関数名がすべて開発者が実装した無名関数というわけではない

ChaChiの文字列難読化解除

- コード案
  - 無名関数かつruntime.slicebytetostringのみ呼び出される関数を対象
    - 関数自身やruntime.morestack_noctxtなどの例外あり
  - 関数の2行目と3行目でスタックに格納されるデータを取得
  - 取得したデータをXORし、文字列に変換
  - 取得した文字列で関数名をリネーム

```
(func() string {
mask := []byte("\x21\x0f\xc7\xbb\x81\x86")
maskedStr := []byte("\x46\x60\xab\xda\xef\xe1")
res := make([]byte, 6)
        for i, m := range mask {
            res[i] = m ^ maskedStr[i]
        }
        return string(res)
        }())
```

main.main.func1関数

```
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:78
main.main.func1
007c4a90 65 48 8b ...    MOV        RCX,qword ptr GS:[0x28]
007c4a99 48 8b 89 ...    MOV        RCX,qword ptr [RCX]
007c4aa0 48 3b 61 10     CMP        RSP,qword ptr [RCX + 0x10]
007c4aa4 0f 86 b0 ...    JBE        LAB 007c4b5a
007c4aaa 48 83 ec 58     SUB
007c4aae 48 89 6c ...    MOV                    ,RBP
007c4ab3 48 8d 6c ...    LEA
```

2行目・3行目

```
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:79
007c4ac2 48 89 44 ...    MOV        RAX,-0x41fbfb281bda8257
007c4ac7 66 c7 44 ...    MOV        qword ptr [RSP + local_1c],RAX
                         MOV        word ptr [RSP + local_14],0xeb50
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:80
007c4ace 48 b8 c7 ...    MOV        RAX,-0x2888894d6bf7ed39
007c4ad8 48 89 44 ...    MOV        qword ptr [RSP + local_26],RAX
007c4add 66 c7 44 ...    MOV        word ptr [RSP + local_1e],0x9f23
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:81
```

省略

```
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go:83
007c4aff 0f b6 54 ...    MOVZX      EDX,byte ptr [RSP + RAX*0x1 + 0x32]
007c4b04 31 d1           XOR        ECX,EDX
007c4b06 88 4c 04 46     MOV        byte ptr [RSP + RAX*0x1 + 0x46],CL
```

ChaChiの文字列難読化解除

- 難読化された文字列が複数存在
- 復号後にメモリに格納される

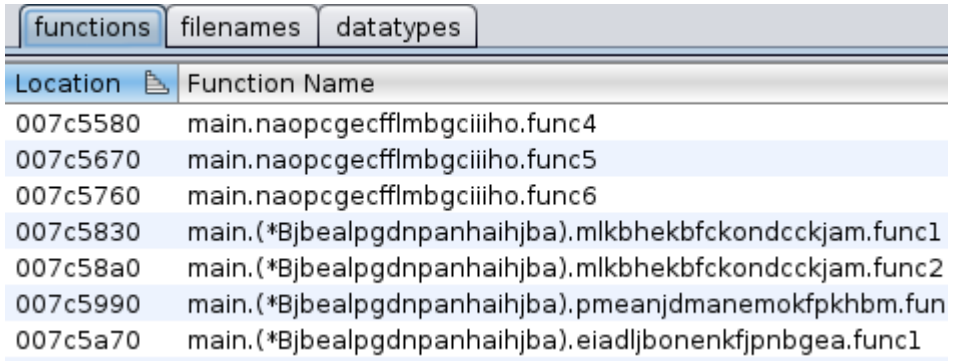
ChaChiのmain.init関数
（格納先メモリのラベルは手動で設定）

難読化されたマルウェアでの機能推定

- ChaChiの難読化された文字列は解決したが、まだ難読化された情報があり解析しづらい

- 難読化された検体を効率的に解析するために、機能と利用しているOSSライブラリの推定について紹介
  - 機能推定
    - 表層情報の収集
  - 利用しているOSSライブラリの推定
    - ファイル名や行番号などから推定

難読化された関数名からは何をしているかがわからない



機能の推測

- ファイル名列を確認する
- サービスの実行機能があると思われる
  - service.goファイルなどが存在
    - 内部の処理でOpenServiceを呼び出す

ファイル名列

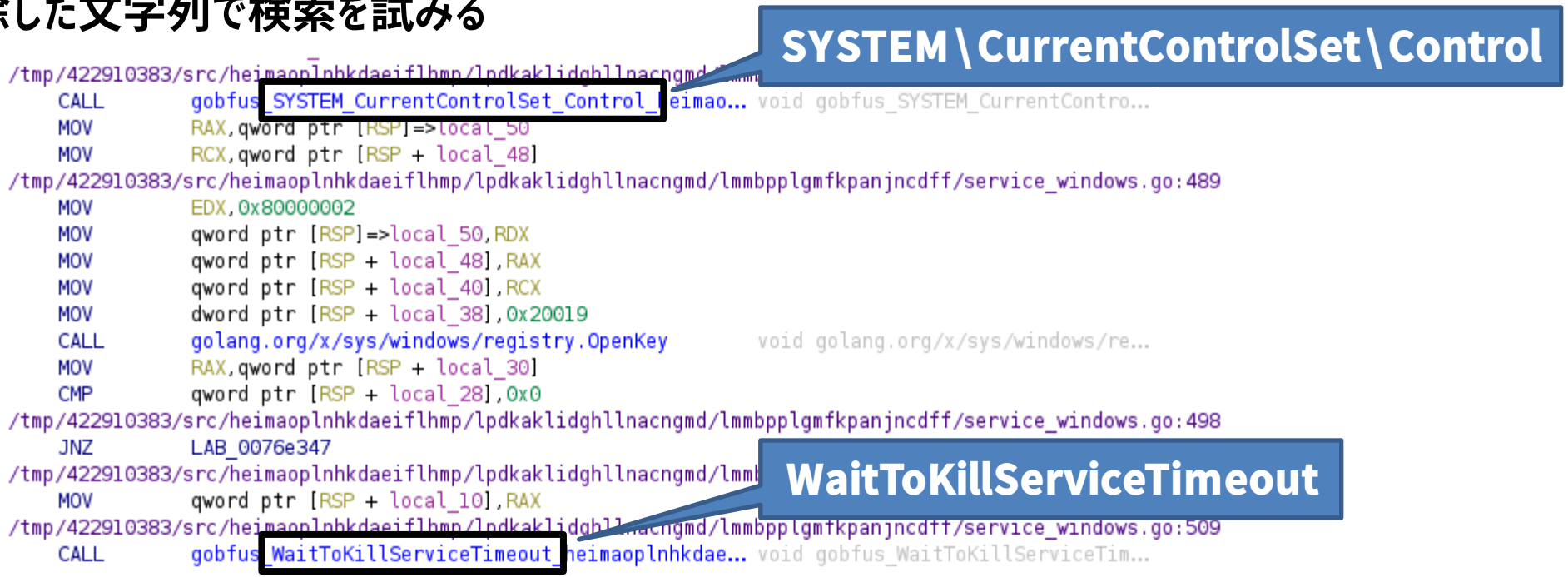
functions | filenames | datatypes

Filename
/tmp/422910383/src/heimaoplnhkdaeiflhmp/lpdkaklidghllnacngmd/lmmbpplgmfkpanjncdff/console.go
/tmp/422910383/src/heimaoplnhkdaeiflhmp/lpdkaklidghllnacngmd/lmmbpplgmfkpanjncdff/service.go
/tmp/422910383/src/heimaoplnhkdaeiflhmp/lpdkaklidghllnacngmd/lmmbpplgmfkpanjncdff/service_go1.8.go
/tmp/422910383/src/heimaoplnhkdaeiflhmp/lpdkaklidghllnacngmd/lmmbpplgmfkpanjncdff/service_windows.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/chisel.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/command.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/commands_windows.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/constants_windows.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/daemon_windows.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/error.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/fake.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/is_admin_windows.go
/tmp/422910383/src/lnjmnamogplhgkllpmbe/main.go

OSSライブラリの推定

- heimaoplnhkdaeiflhmp/lpdkaklidghllnacngmd/lmmbpplgmfkpanjncdffについて調査する
  - おそらくgithub.comで公開されているOSSライブラリ
- 上記パスを特定できれば、ソースコードを読んで内部の処理と使われ方が分かり解析が捗る

- 以下の難読化解除した文字列で検索を試みる



ライブラリの特定

- 検索：「site:github.com golang WaitToKillServiceTimeout SYSTEM\CurrentControlSet\Control」
  - https://github.com/kardianos/service/blob/master/service_windows.go
    - ファイル名が一致
    - 行番号もある程度一致
      - 文字列難読化により行数がずれる
  - https://github.com/takama/daemon/blob/master/daemon_windows.go
    - ファイル名が一致しない

- 「github.com/kardianos/service」が利用されているOSSライブラリ

補足：ChaChiで利用されているOSSライブラリ

- github.com/rs/xid
  - 一意なIDを取得
- github.com/fasthttp/websocket
  - fasthttpをサポートするgorilla/websocketのフォーク
- github.com/armon/go-socks5
  - SOCKS5サーバー
- github.com/fsnotify/fsnotify
  - ファイルシステム通知
- github.com/jpillora/backoff
  - 指数バックオフカウンタ
- github.com/Jeffail/tunny
  - Goroutineプールを生成し管理するためのライブラリ
- etc

まとめ

振り返りと課題

- 問題
  - Go言語製マルウェアには解析を困難にする課題がいくつか存在

- 解析能力向上のために紹介した内容
  - 基本的な解析フローとTips
  - Go言語製バイナリ内のメタデータとそれを利用するツールの改造方法
  - Go言語のバージョンアップへの対応
  - 難読化された検体への対策

- 今後の課題
  - Ghidraなどのデコンパイル性能が良くない
  - 動的解析ツールの不足

利用したツール等

- https://github.com/mooncat-greenpy/Ghidra_GolangAnalyzerExtension
  - 本発表で主に利用したGo言語製バイナリ解析用のGhidraプラグイン

- https://github.com/FFRI/JSAC2023-GolangMalwareAnalysis
  - バージョンアップへの対応で修正したradare2のスクリプト
  - gobfuscateによる文字列難読化を解除するGhidraのスクリプト

Thank you for listening!

APPENDIX

Appendix：init関数

- init関数
  - 初期化などに使用される関数である
  - main関数よりも前に呼び出される
  - 複数定義できる

- アセンブリでの表現
  - 関数名は「＜モジュール名＞.init.%d」または「＜モジュール名＞.init」という形式になる
    - 例：main.init.0
  - init関数はruntime.doInit関数によって呼び出される
    - runtime.doInit関数はmain関数を呼び出すruntime.main関数から呼ばれる

Appendix ：Goroutine

- Goroutine
  - Go独自の軽量なスレッド
  - 例：go sub_func(0x1, 0x10, 0x100, 0x1000) // sub_func関数を実行

- アセンブリでの表現
  - runtime.newproc関数によりGoroutineが起動
  - バージョンによる引数の違い
    - Go 1.17以下
      - 第２引数に「呼び出し関数ポインター、引数1、引数2、…」が設定されているメモリへのポインターを渡す
    - Go 1.17以上
      - newprocに対象関数のラッパー関数が渡される
        - » いくつかの引数はラッパー関数内で用意される
    - Go 1.18以上
      - 第1引数に上述のポインターを渡す

Appendix：バージョンの判定

- Go言語製バイナリのビルドに使用されたGo言語のバージョンを取り出す方法を以下に示す