Emotet vs EmoCheck

Emotet開発者との戦い

NTT	社会情報研究所	谷　知亮
JPCERT/CC	インシデントレスポンスグループ	喜野　孝太
JPCERT/CC	インシデントレスポンスグループ	佐條　研

$whoami

Tomoaki Tani

NTT 社会情報研究所 社会理論研究プロジェクトに所属。主にマルウェア分析技術や脆弱性分析技術の研究を行っている。以前はJPCERT/CCでマルウェア分析やインシデント対応に従事していた。Virus Bulletin, CODEBLUE, BotConf, BsidesLV, BlackHat USA Arsenalなどで講演。

Kota Kino

国内ITベンダーでのメールセキュリティ製品の導入支援業務を経て、2019年よりJPCERT/CC。現在は、標的型攻撃への対応も含んだ、日本国内におけるさまざまなインシデントの分析業務に従事。CODE BLUE、Botconfなどで講演。

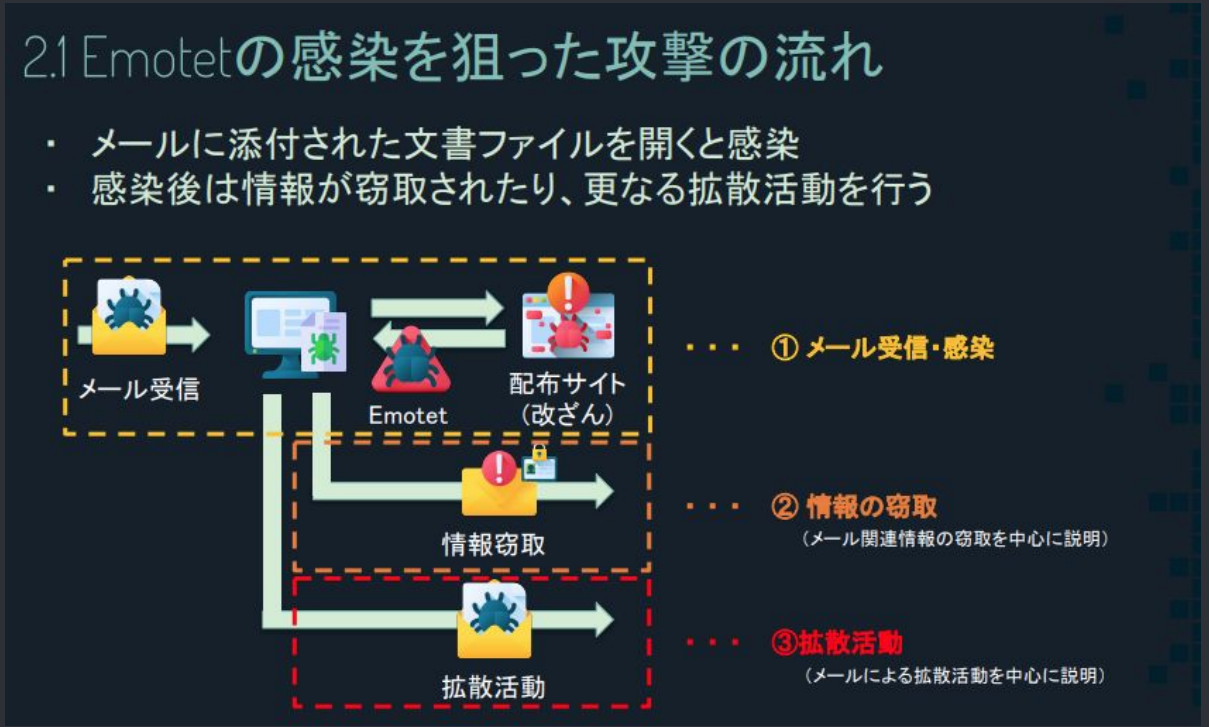
Ken Sajo

金融系企業でのセキュリティ監視業務を経て、現在はインシデント対応やマルウェア分析、脅威情報分析の業務に従事。傍ら、個人の活動としてばらまきメールの情報発信・分析も行っている。JSAC2020スピーカー、JSAC2021ベストスピーカー

本日の目的

- 最新のEmotetアップデート状況の共有
- Emotet検知ツールEmocheckの機能解説
- 攻撃者が使う難読化技術とその実装例の紹介
- マルウェア検知ツールを作る意義

Emotetとは？ (~2021/1)



（引用）https://jsac.jpcert.or.jp/archive/2021/pdf/JSAC2021_104_sajo-sasada_jp.pdf , p9

アジェンダ

1. 復活したEmotet(2021/11~)
2. EmoCheck概要
3. Emotet vs EmoCheck
4. Emotetのバイナリ難読化処理
5. EmoCheckのバイナリ難読化の実装手法
6. まとめ

- 復活したEmotet
(2021/11 ~)

Emotet本体の変更点

- 利用する公開鍵暗号方式の変更(RSAからECC)
- 利用する共通鍵暗号方式の変更(AES128からAES256)
- 通信プロトコルの変更(HTTPからHTTPS)
- 本体のコマンドセットの追加
- C&Cサーバーのリストの格納方式の変更

Emotet本体のコマンドセット

コマンド	内容
1	自身のアップデート
2	モジュールをロードし、実行
3	EXEファイルをダウンロードし、実行
4	EXEファイルをダウンロードし、（特定のユーザで）実行
5	DLLファイルをメモリ内にインジェクションし、実行
6	DLLファイルをダウンロードし、実行（ regsvr32.exe）
7 New	DLLファイルをダウンロードし、実行（ rundll32.exe）

モジュールの追加

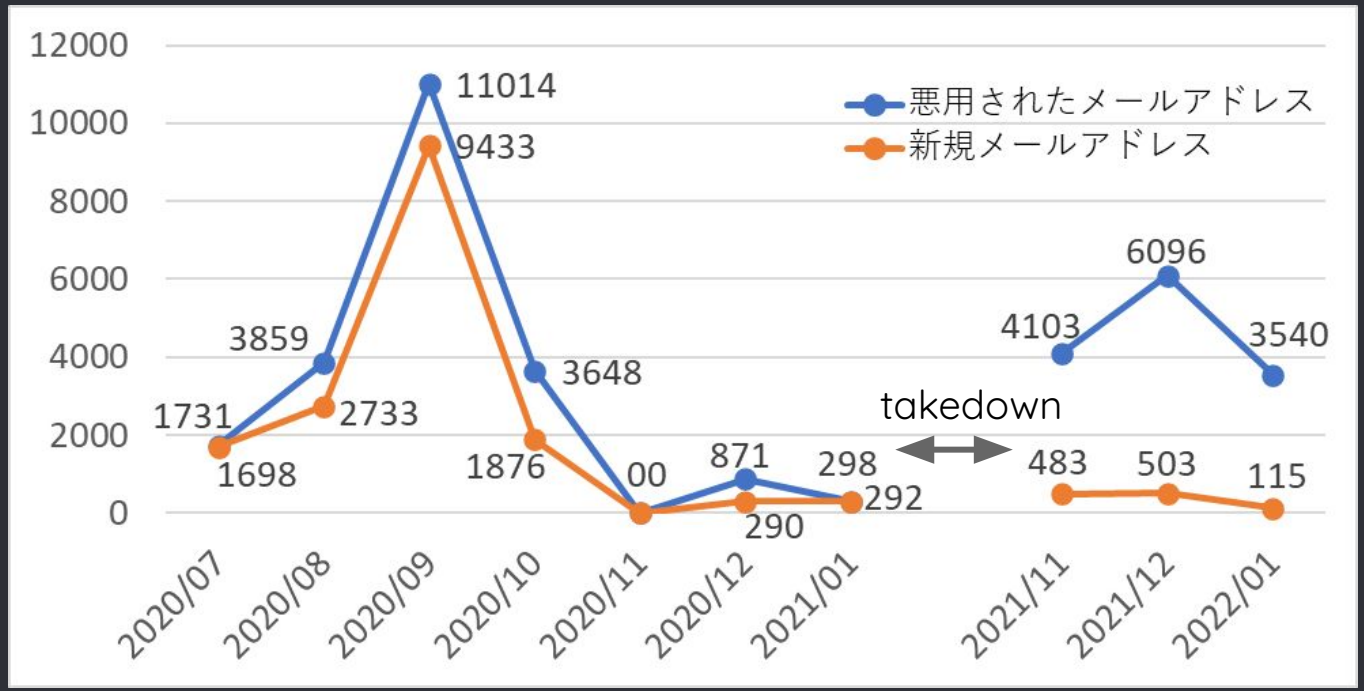
モジュール	内容
プロセスリスト New	プロセス一覧の取得
メール PassView	メールクライアントのパスワード窃取
Webブラウザ PassView	Webブラウザのパスワード窃取
Outlookアカウントスティーラー	Outlookに保存されている連絡先の窃取
Outlookメールスティーラー	Outlookに保存されているメールの窃取
Thunderbirdアカウントスティーラー New	Thunderbirdに保存されている連絡先の窃取
Thunderbirdメールスティーラー New	Thunderbirdに保存されているメールの窃取
メール送信モジュール	スパムメールの送信
返信型メール送信モジュール	窃取したメールから返信を装ったスパムメールの送信
ネットワーク拡散モジュール	ローカルネットワーク内をブルートフォース

Emotetが配信するマルウェア

- 再開前(~ 2021/1)
  - Trickbot (-> Cobalt Strike beacon)
  - Qakbot
  - IcedID
  - Ursnif　（日本のみ）
  - ZLoader（〃）
- 再開後(2021/11 ~)
  - Cobalt Strike beacon

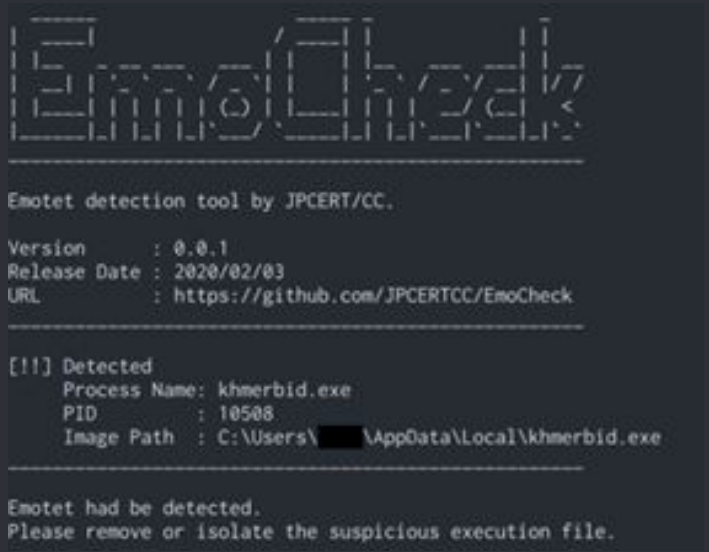
国内のEmotet感染状況 (~2022/01/13)

窃取され悪用された国内ドメイン(.jp)のメールアカウント数



EmoCheck概要

EmoCheckとは？



- Emotetのプロセスを検知するツール
  https://github.com/JPCERTCC/EmoCheck
- 初版リリース: 2020/2/3
- 最新バージョン: 2.0
- ダウンロード回数: > 60万回
- 多言語対応: 日/仏/英
- 想定利用者:
  - すべてのWindowsユーザー
  - EDRなどを導入していない人

私、Emotetに感染してますか？

なぜEmoCheckは作られたのか？

そんな疑問にお答えするためのツール

（インシデント対応を楽にする）


マルウエア Emotet の感染に関する注意喚起

最終更新: 2019-12-10

ツイート　メール

JPCERT-AT-2019-0044
JPCERT/CC
2019-11-27(新規)
2019-12-10(更新)

I. 概要

JPCERT/CC では、2019年10月後半より、マルウエア Emotet の感染に関する相談を多数受けています。特に実在の組織や人物になりすましたメールに添付された悪性な Word 文書ファイルによる感染被害の報告を多数受けています。

こうした状況から、Emotet の感染拡大を防ぐため、JPCERT/CC は本注意喚起を発行し、Emotet の主な感染経路、Emotet に感染した場合の影響を紹介した後、感染を防ぐための対策や、感染に気付くためにできること、感染後の対応方法などに関する情報を紹介します。


佐條 研(Ken Sajo)　2019/12/02

マルウエアEmotetへの対応FAQ

Emotet

ツイート　メール

最終更新日:2021.12.1

2019年10月以降、日本国内にてEmotetの感染事例が急増しています。JPCERT/CCでは、次の通り注意喚起を発行しています。

JPCERT/CC: マルウエア Emotet の感染に関する注意喚起
https://www.jpcert.or.jp/at/2019/at190044.html
JPCERT/CC: CyberNewsFlash マルウエア Emotet の感染活動について
https://www.jpcert.or.jp/newsflash/2019112701.html
JPCERT/CC: CyberNewsFlash マルウエア Emotet の感染に繋がるメールの配布活動の再開について (追加情報)
https://www.jpcert.or.jp/newsflash/2020072001.html
JPCERT/CC: CyberNewsFlash マルウエア Emotet の感染拡大および新たな攻撃手法について

要求仕様

1. 簡単に利用できる
    a. Windows NT6.1以上で動作
    b. ポータビリティがある（no setup）
    c. 最小限のユーザー操作
    d. ファイルサイズが小さい
    e. 1秒以内に結果を出力
2. 確実に感染を検知する
    a. Emotetローダーのプロセスが動作 == 感染
    b. MalDocの実行 != 感染
3. 極力フォレンジックの阻害しない

開発言語の選定 (2019年12月時点)

	C++ (MSVC)	.NET	Go	PowerShell	Python
環境依存	○	△ .Net Core依存	○	△ 外部スクリプト実行	✗ 要インタープリタ
ポータビリティ	○	△ .Net Core依存	○	△	✗ 要インタープリタ
ファイルサイズ	○	✗ .Net Core含む	△ 肥大化しやすい	○	○
実行速度	○	○	○	?	✗
フォレンジックへの影響	○	○	○	△	✗
難読化の容易さ	△	○ ConfuseEx	?	○	○
Undocumented APIの利用	○	○	?	△	△

- # Emotet vs EmoCheck (検知ロジックについて)

検知の基本方針

Emotetは必ず永続化をする

★ **Malware-as-a-Service**であるEmotetのビジネスは**感染端末数の維持**が必須

EmoCheckは永続化手法の再現し、痕跡を探す

戦いの始まり

ドライブシリアルによるキーワード選択型

2020.1.21 -

- 独自のキーワードリストを保持
- ドライブシリアルからキーワードを複数選択しファイル名設定

キーワードリスト

```
keywords = "duck,mfidl,targets,ptr,khmer,purge,metrics,acc,inet,msra,symbol,driver,";
keywords += "sidebar,restore,msg,volume,cards,shext,query,roam,etw,mexico,basic,url,";
keywords += "createa,blb,pal,cors,send,devices,radio,bid,format,thrd,taskmgr,timeout,";
keywords += "vmd,ctl,bta,shlp,avi,exce,dbt,pfx,rtp,edge,mult,clr,wmistr,ellipse,vol,";
keywords += "cyan,ses,guid,wce,wmp,dvb,elem,channel,space,digital,pdeft,violet,thunk";
```

キーワード選択ロジック

```
seed = GetSystemDriveSerial();
q,r = divmod(seed/sizeof(keywords));
select_keyword(q,r);
```

環境ごとに固定のキーワードの組み合わせが選択される

EmoCheck Scanner 1

ver.0.0.1　2020.2.3

1. ドライブのシリアル番号を取得（例. 0x2EA4B3F0）

2. シリアル番号をキーワードリストの文字列長で除算し、剰余の値をポインタとして使用（例. 0x2EA4B3F0 % 18 = 8）

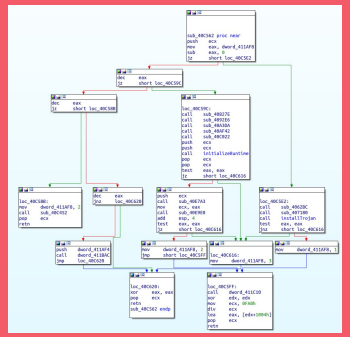
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
a	p	p	l	e	,	o	r	a	n	g	e	,	l	e	m	o	n

3. ポインタの位置にあるキーワードを選択（例. orange）

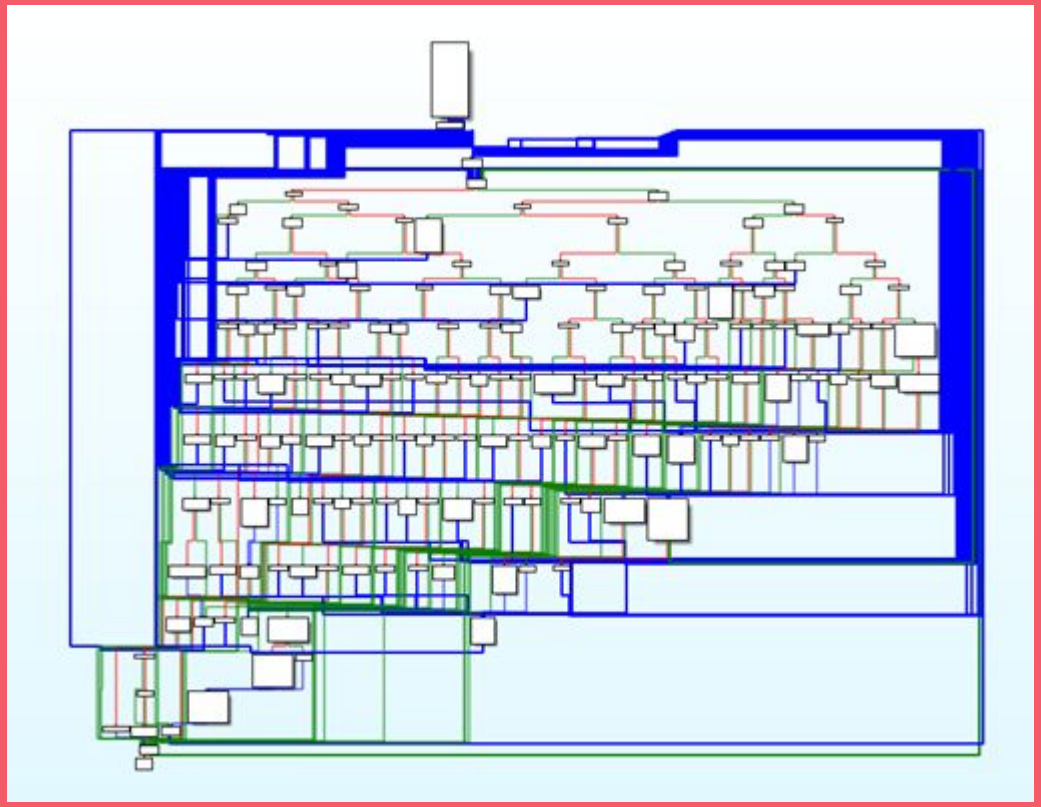
4. 1 ～ 3. で選択した2個のキーワードを結合した、[keyword1+keyword2].exeのプロセス名が存在すれば検知（例. orangeapple.exe）

３日後に永続化方式が置き換えられ、
EmoCheckで検知できなくなる。
※Emotet 開発モードへ移行

Emotet Update



Before



After

システムファイル名+レジストリにファイル名保存

2020.2.6 -

1. システムディレクトリ配下で拡張子がdllまたはexeのファイル名をランダムに選択し、実行ファイル名へ設定
2. 実行ファイル名を特定のレジストリへエンコードして保存（マーカー）

レジストリパス

```
HKLM(HKCU)\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer
```

キー

ドライブシリアル番号

値

```
key = GetSystemDriveSerial();
RegValue = XOR(FileNameOfEmotet,key);
```

EmoCheck Scanner 2

ver.0.0.2

2020.2.10

1. ドライブのシリアル番号を取得（例. 0x2EA4B3F0）
2. 以下のレジストリに格納されている値を取得

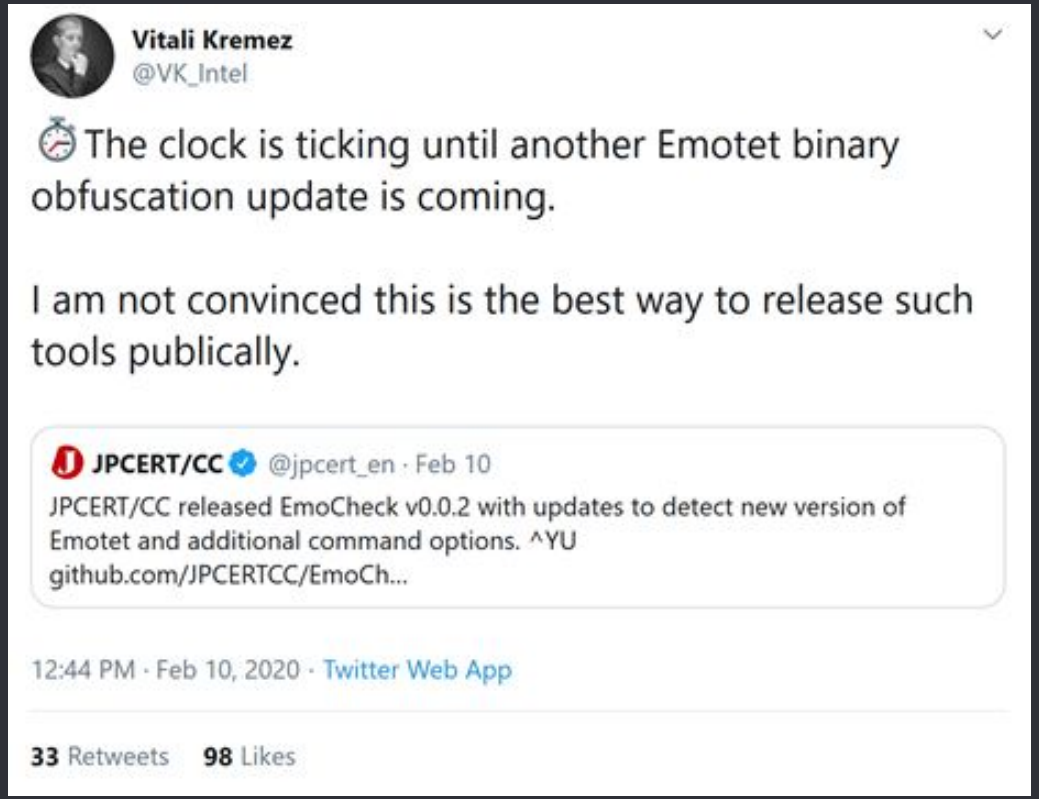
   レジストリパス

   `HKLM(HKCU)\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer`

   キー

   ドライブシリアル番号
3. 2. で取得した値をドライブシリアルでXORデコードしEmotetのファイル名を取得
4. Emotetのファイル名と同じプロセスが存在すれば検知

Twitterでの声



Vitali Kremez
@VK_Intel

⏱The clock is ticking until another Emotet binary obfuscation update is coming.

I am not convinced this is the best way to release such tools publically.

> JPCERT/CC @jpcert_en · Feb 10
> JPCERT/CC released EmoCheck v0.0.2 with updates to detect new version of Emotet and additional command options. ^YU
> github.com/JPCERTCC/EmoCh...

12:44 PM · Feb 10, 2020 · Twitter Web App

33 Retweets **98** Likes

(超意訳)
EmoCheckみたいなツールを
オープンソースで公開するのって
どうなのよ？
Emotetの難読化がさらにひどく
なる未来しか見えない。

EmoCheckに難読化を！

2020年4月 Emotet更新
※ばらまきメール配信停止中
※Emotet 開発モード中

システムファイル名からのランダム選択

2020.4.20 -

- システムディレクトリ配下で拡張子がdllまたはexeのファイル名[A]、[B]を選択（ランダム）
- %LOCALAPPDATA%\[A]\[B].exeにEmotetを保存
- Runキーまたはサービスへ永続化設定

EmoCheck Scanner 3

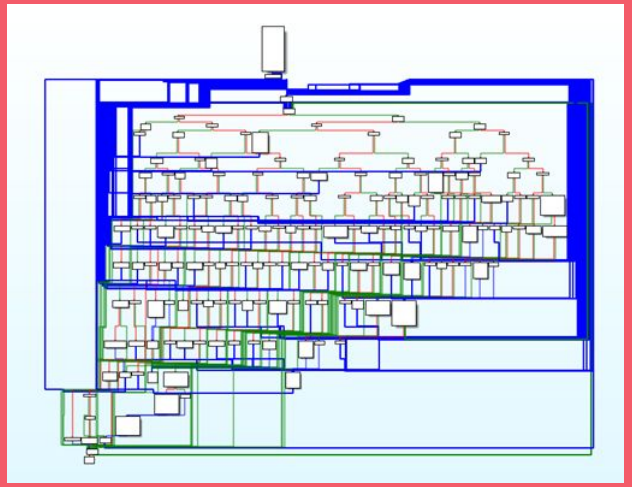
ver.1.0

2020.8.11

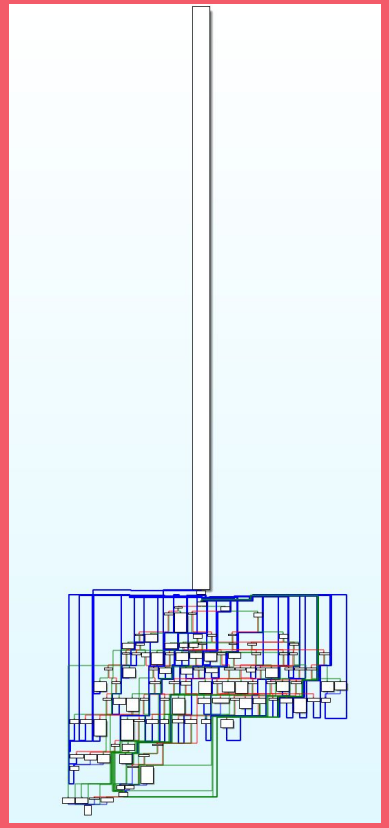
1. システムディレクトリ配下で拡張子がdllまたはexeのファイル名を全抽出
2. %LOCALAPPDATA%\[A]\[B].exe に該当するファイルを検索
    a. [A] == 1で取得したファイル名
    b. [B] == 1で取得したファイル名
3. 同一名の正規の実行ファイルと同一であれば除外
4. 実行中の全プロセスのコマンドラインを取得
5. 2のパスを含むプロセスを検知

2020年11月 Emotet活動停止。
※Emotet 開発モードへ移行

Emotet Update



Before



After

ランダム文字列 + rundll32経由での実行

2020.12.21 -

- 以下のパターンでEmotetを設置

```
%LOCALAPPDATA%\\[%s]{4,16}\\[%s]{3,15}.[%s]{3}
```

- 以下のコマンドラインで起動
  - rundll32.exe [Emotetのパス], Control_RunDLL
- コマンドラインをRunキーまたはサービスへ永続化設定

EmoCheck Scanner 4

ver.2.0

2021.1.27

1. 以下のパターンのファイルパスを検索
   a. `%LOCALAPPDATA%\\[%s]{4,16}\\[%s]{3,15}.[%s]{3}`
2. 実行中のプロセスのコマンドラインを取得
3. 以下の正規表現と一致するコマンドラインを検知
   a. `^.*rundll32\\.exe.*,.*RunDLL.*$`
   b. `^.*rundll32\\.exe.*,.*ShowDialogA.*$`
   c. `^.*rundll32\\.exe.*,.*[A-Za-z]{4,15}.*$`

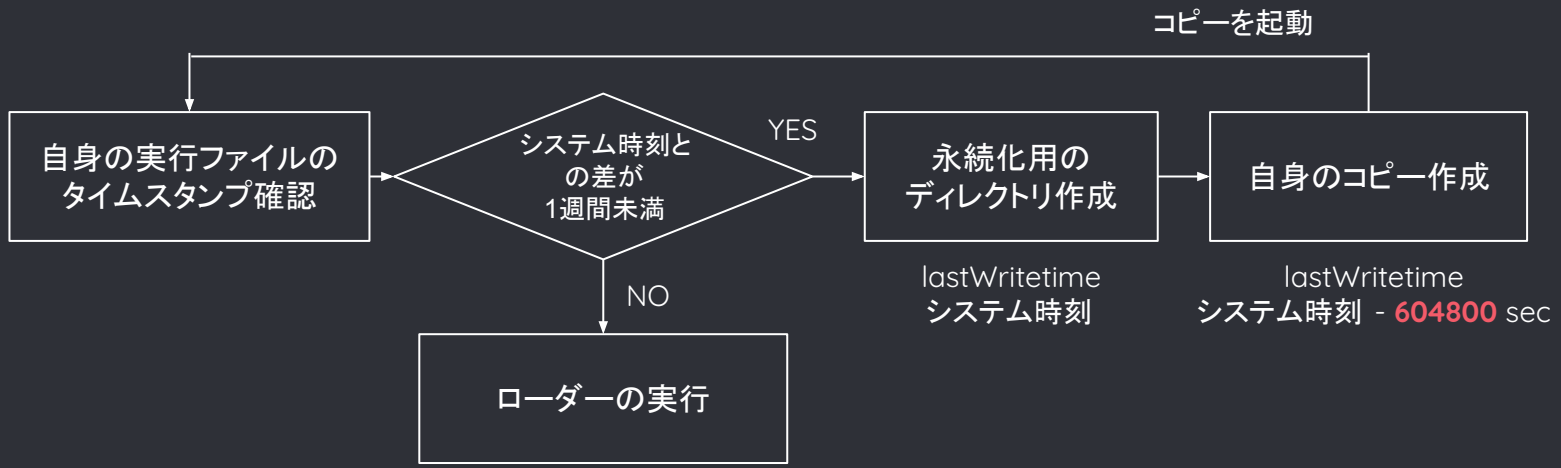
2021.11以降のEmotetも検知可能

正規表現での検知....?

(False Positive/False Negative)

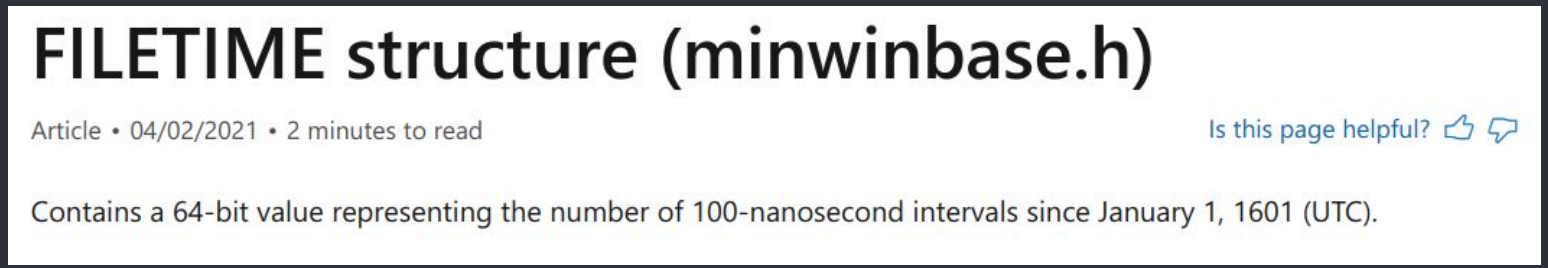
変わらないEmotetの挙動

特徴的なローダー実行判定を行う。



FILETIME構造体

Windowsは時刻情報を100ナノ秒単位で管理



FILETIME structure (minwinbase.h)

Article • 04/02/2021 • 2 minutes to read

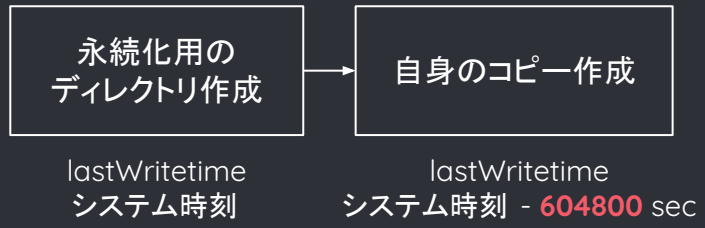
Is this page helpful?

Contains a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC).

https://docs.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-filetime

検証1: 環境差による差分時間を範囲

FILETIME構造体による時刻情報の差はどうなるか？



検証環境: **物理環境**(CPU: Intel Core i7 3770) / **仮想環境** (VMWare)

検証結果:

[Emotetが作成するディレクトリのlastWritetime] - [本体コピーのlastWritetime] ≒ **604800.000sec ± 0.090 sec** に収束。(10回ずつ実行した平均値)

EmoCheck Scanner 5

ver.2.0

2021.1.27

1. 以下格納フォルダとファイルのLastWriteTimeの差分が**604800.000sec ± 0.100 sec未満**のファイルを列挙
   a. %LOCALAPPDATA%
   b. %WINDIR%\system32
2. 実行中のプロセスをコマンドラインを取得
3. 列挙したファイルを実行するプロセスを検知

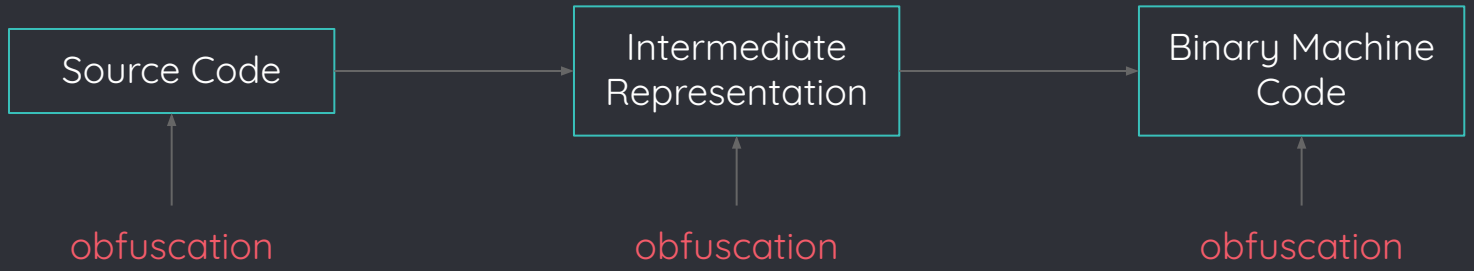
2021.11以降のEmotetも検知可能

- Emotetの難読化処理

一般的なC/C++プログラムの難読化の流れ

各抽象レイヤーでそれぞれ難読化処理（&最適化処理）が施される。

Abstruction



Emotet(Loader)に施されている難読化

2020.12.21 -

Method	Abstruction	Unit	Target
String Obfuscation	maybe Source Code	Instruction	Data Constant
Mixed Boolean Arithmetic	Intermediate Representation	Basic Block	Data Constant
Control Flow Flattening	Intermediate Representation	Function	Code Logic
Win32API Hashing Obfuscation	Source Code	Function	Code Abstraction
Function Argument Randomization	Intermediate Representation	Function	Code Abstraction

String Obfuscation

文字列をXORエンコードし、バイナリ内に格納。文字列の利用時のみに Heap領域へ復号。
XOR-keyは文字列ごと/検体ごとに異なる。

難読化前（pseudo-code）

```
reg_path = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";
```

難読化後 (pseudo-code)

```
reg_path = mal_string_decrypt(&enc_registry_path);
```

&enc_registry_path(0x100014F8)　　XOR-Key　　XORed Size(=45)

```
100014F0  00 00 00 00 00 00 00 00  5F FE 4D 32 72 FE 4D 32
10001500  0C B1 0B 66 08 BF 1F 77  03 B3 24 51 2D 91 3E 5D
10001510  39 8A 11 65 36 90 29 5D  28 8D 11 71 2A 8C 3F 57
10001520  31 8A 1B 57 2D 8D 24 5D  31 A2 1F 47 31 60 5A 72
10001530  97 DB FB 1E 33 ED 59 B7  93 AD B5 39 38 04 AE 32
```

XORed String

Mixed Boolean Arithmetic

定数値/変数の初期値と同値となるような算術演算とビット演算の組み合わせ
(XOR,ShiftLeft,ShiftRight,Addなど)に置換し、データを難読化。

難読化前

```
mov     [esp+84h+regsam], 02h   ; KEY_SET_VALUE
```

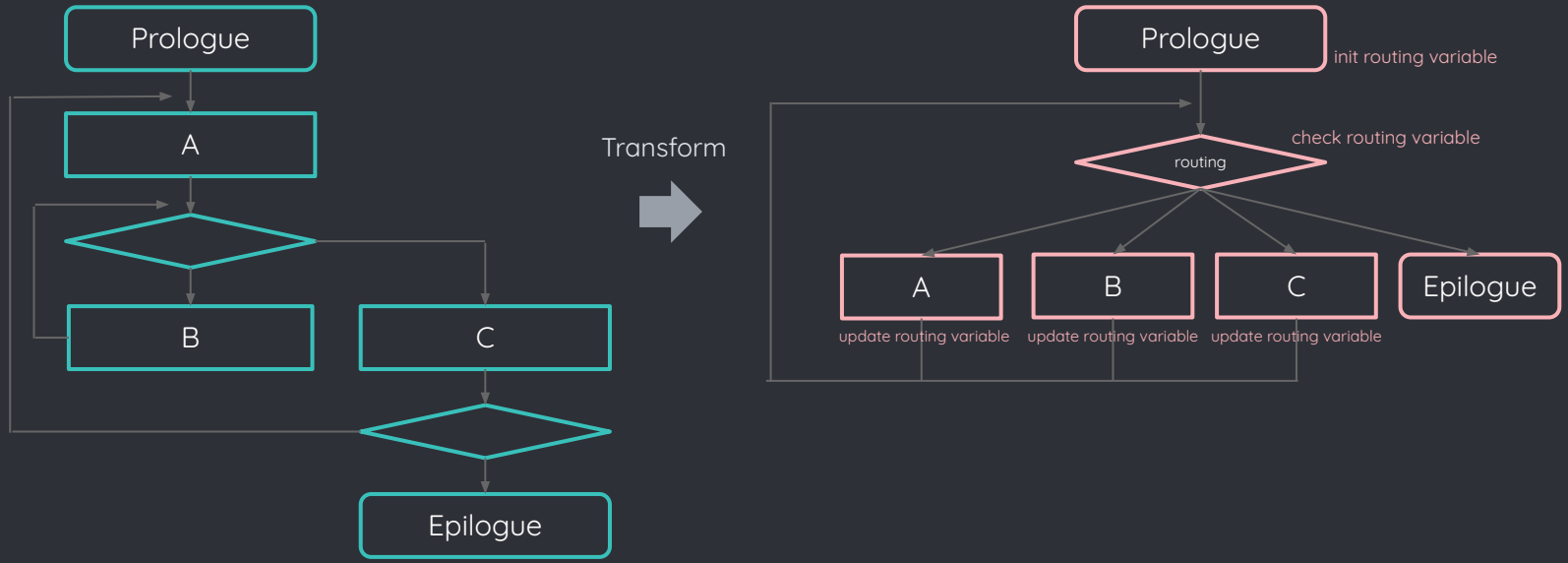
難読化後

```
mov     [esp+84h+regsam], 0D4B91Fh      ; [esp+84h+regsam] => 00D4B91Fh
shl     [esp+84h+regsam], 0Eh           ; [esp+84h+regsam] => 2E47C000h
shr     [esp+84h+regsam], 0Dh           ; [esp+84h+regsam] => 0001723Eh
add     [esp+84h+regsam], 0FFFF4A0Bh    ; [esp+84h+regsam] => 0000BC49h
xor     [esp+84h+regsam], 0BC4Bh        ; [esp+84h+regsam] => 00000002h
```

```
((((((0xD4B91F << 0xE) & 0xFFFFFFFF) >> 0xD) & 0xFFFFFFFF) + 0xFFFF4A0B) & 0xFFFFFFFF) ^ 0xBC4B = 2
```

Control Flow Flattening

ジャンプテーブルを作成し、制御フローを平坦化し、BasicBlockの実行順を難読化。
実行順の制御にはRouting変数を用いる。



Control Flow Flattening

Emotetでは単一のRouting変数によってBasic Blockの実行順を難読化。

Emotetの関数 CFF解除（pseudo-code）

```
BOOL mal_DeletePersistentRegistry(){
    LPCWSTR lpValueName;
    const WCHAR *reg_path;
    HKEY phkResult;
    BOOL status;
    lpValueName = NULL;

    // Block 1: Get registry value name from global data section.
    for (int i = pGlobalData + 564; *i != '\\'; i += 2 );
    lpValueName = (i + 2);

    // Block 2: Open Windows registry key handle
    reg_path = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"
    res = RegCreateKeyExW(0,lpSubKey,0,0,0,regsam,0,phkResult,0);
    HeapFree(reg_path);

    if (res == ERROR_SUCCESS)
        // Block 3: Delete registry value
        status = RegDeleteValueW(phkResult, lpValueName) == 0;

    // Block4: Close Windows registry key handle
    RegCloseKey(phkResult);
    return status;
}
```

Emotetの関数: CFF (pseudo-code)

```
BOOL mal_DeletePersistentReg(){
  int route;
  LPCWSTR lpValueName;
  const WCHAR *reg_path;
  HKEY phkResult;
  BOOL status;
  lpValueName = NULL;
  route = 0xC62CE70;   // init routing variable
  while ( 1 ) {
    while ( 1 ){
      while ( 1 ) {
        while ( route == 0x6EF1DF9 ){
          reg_path = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"
          route = 0x9084733;   // update routing variable
          if(!RegCreateKeyExW(hKey, lpSubKey, 0,0,0, regsam,0, phkResult,0))
            route = 0x7FE3D14; // update routing variable
          HeapFree(reg_path);
          if ( route == 0x9084733 ) return status;
        }
        if ( route != 0x7FE3D14 ) break;
        route = 0xE13E5E7;     // update routing variable
        status = RegDeleteValueW(phkResult, lpValueName) == 0;
      }
      if ( route != 0xC62CE70 ) break;
      …(snip)
```

Win32API Hashing Obfuscation

Win32API名のHash値を使い、PEBから関数ポインタを探し、API呼び出しを行うことで、プログラムが使用するAPIを難読化。

難読化前（pseudo-code）

```
RegDeleteValueW(hKey, lpValueName);
```

難読化後（pseudo-code）

```
sub_1001BCBC(hKey, lpValueName);

sub_1001BCBC(HKEY hKey, LPCWSTR lpValueName){
  LPVOID func;
  DWORD ApiHash = 0xCDA8113B;
  DWORD LibHash = 0x004E81E0;
  func = GetFunctionPtrByHash(idx, dwLibHash, dwApiHash);
  return func(hKey,LPCWSTR lpValueName);
}
```

Function Argument Randomization

関数コールの引数に不必要な定数と追加し、引数の順番を入れ替える。追加された引数はプログラムの動作には無関係であり、呼び出し元の関数のスタックに保持される。

難読化前（pseudo-code）

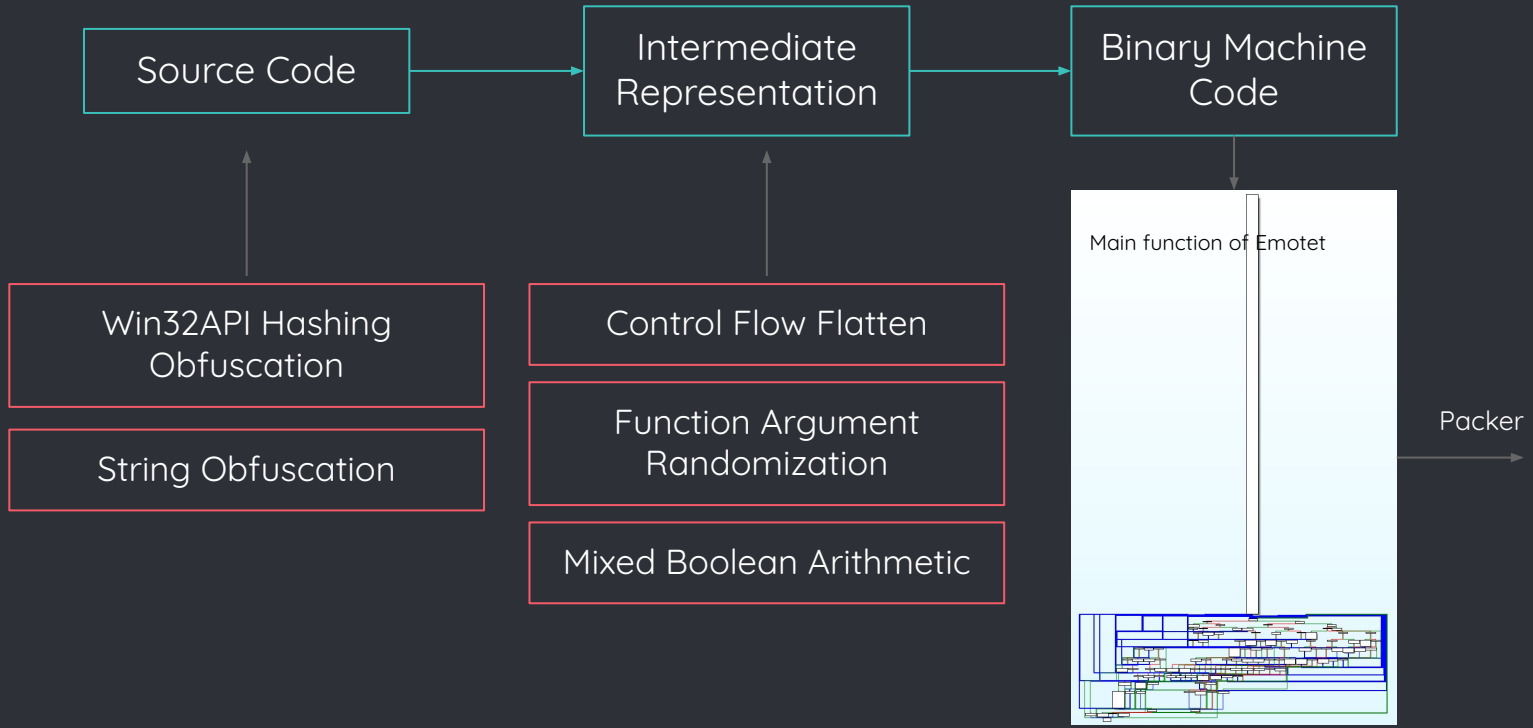
```
reg_path = mal_DecryptString(&enc_registry_path);  // エンコードされた文字列を復号
```

難読化後（pseudo-code）

```
reg_path = mal_DecryptString(732570, 117490, 388602, &enc_registry_path);
```

Emotetの難読化処理

それぞれの難読化が各抽象化レイヤーで適用されている。



- EmoCheckのバイナリ難読化の実装手法

“

いたちごっこに歯止めを。

難読化実装の要件

1. Emotetと同等の難読化を含む (2020.2.6時点)
    a. Control Flow Flatten
    b. Win32API Hashing Obfuscation
    c. String Obfuscation
2. プロジェクト単位で難読化できる
3. 自動で難読化コンパイル
    a. 1コマンドで難読化コンパイルができる
4. 無償でできる

難読化関連の利用プロダクト

C++プロジェクト管理	• CMake (>3.15) • Ninja build
コンパイラ	• LLVM 9.0（通常コンパイル用） • O-LLVMとObfuscator Hikari (難読化コンパイル用) [13] [14]
ライブラリ	• API難読化ライブラリ（別途実装) 参考:[9] [10] • 文字列難読化ライブラリ(別途実装) 参考:[9] [10]

難読化の流れ

Phase1 ソースコード 難読化	C++プリプロセッサ テンプレートメタプログラミング /constexpr ● String Obfuscation ● Win32API Hashing Obfuscation
Phase2 中間表現 難読化	コンパイラ機能 (LLVM-pass) ● Control Flow Flattening ● String Obfuscation ● Split Basic Blocks etc

難読化の実装

ソースコードレイヤー

C++プリプロセッサとテンプレートメタプログラミング/constexpr

C++のソースコード難読化 [7][8]ではマクロなどのプリプロセッサ機能とメタプログラミング技法であるテンプレートメタプログラミング [5]とconstexpr[6]を利用。

C++プリプロセッサ

文字列操作を利用して、
コンパイル時に
ソースコードの変換を行う。

テンプレートメタプログラミング

テンプレートを利用して、
コンパイル時に
（チューリング完全な）
演算処理を行い、
ソースコードの生成と変換を行う

メタプログラミングの例

例：階乗の計算（メタプログラミング なし）

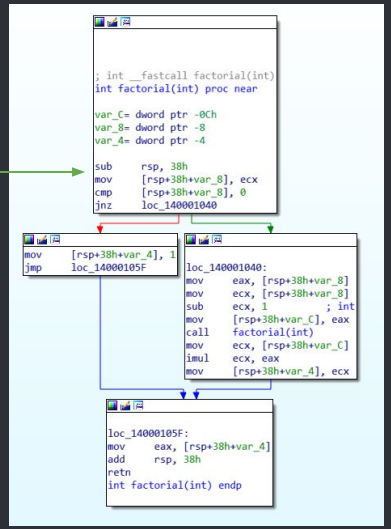
```
int factorial(int n) {
 if (n == 0)
   return 1;
 return n * factorial(n-1)

int main(int, char**){
 int x = factorial(4);  // == 4*3*2*1 == 24
 int y = factorial(0);  // == 1
}
```

コンパイル

main関数

```
; int __fastcall main(int, char **)
main proc near

var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= qword ptr -8

sub     rsp, 38h
mov     [rsp+38h+var_8], rdx
mov     [rsp+38h+var_C], ecx
mov     ecx, 4          ; int
call    factorial(int)
xor     ecx, ecx        ; int
mov     [rsp+38h+var_10], eax
call    factorial(int)
xor     ecx, ecx
mov     [rsp+38h+var_14], eax
mov     eax, ecx
add     rsp, 38h
retn
main endp
```

factorial関数



メタプログラミングの例

例：階乗の計算（メタプログラミング あり）

```
template <int N>
struct Factorial {
 enum {value = N * Factorial<N-1>::value};
};
template <>
struct Factorial<0> {
 enum {value = 1};
};
int main(int, char**){
 int x = Factorial<4>::value;  // 4*3*2*1 == 24
 int y = Factorial<0>::value;  // == 1
}
```

コンパイル

一時ソースコード

```
int main(int, char**){
 int x = 24;  // == 4*3*2*1 == 24
 int y = 1;  // == 1
}
```

コンパイル

main関数

Win32API Hashing Obfuscationの実装

- ソースコード内のAPI文字列からハッシュ値を計算する — メタプログラミング
- ソースコードのWin32APIの呼び出しをハッシュ値による呼び出しに置換 — マクロ
- ハッシュ値によるWin32API呼び出しの実装 — MSVC
  - PEBからAPIアドレステーブルとAPI名一覧取得
  - API名のハッシュ値を逐次計算し、一致するAPIを探索
  - 一致したAPI名の関数ポインタを返す

Win32API Hashing Obfuscationの実装

FNV-1a ハッシュをメタプログラミング用コード

```
// FNV hash parameters
// https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function#FNV-1a_hash
constexpr uint32_t val_32_const = 0x811c9dc5;
constexpr uint32_t prime_32_const = 0x1000193;
constexpr uint64_t val_64_const = 0xcbf29ce484222325;
constexpr uint64_t prime_64_const = 0x100000001b3;

inline constexpr uint32_t Hash32Fnv1aConst(const char* const str, const uint32_t value = val_32_const) noexcept {
   return (str[0] == '\0') ? value : Hash32Fnv1aConst(&str[1], (value ^ uint32_t(str[0])) * prime_32_const);
}
```

コンパイル時にAPI名がハッシュ値に変換される

MessageBoxA → fnv-1a → 0x23a979e4

Win32API Hashing Obfuscationの実装

ハッシュ値からAPIの関数ポインタを取得するテンプレート

```
template <uint32_t hash>
static void* GetApiPtrByHash(const wchar_t* dll_name = NULL) {
   static void* function_ptr;
   if (!function_ptr)
       function_ptr = GetFuncByHash(hash, dll_name);
   return function_ptr;
}
```

Win32APIをハッシュ値によるAPI呼び出しに置換するマクロ

```
#define OBF_CALL(dll_name, name)
(reinterpret_cast<decltype(&name)>(obfuscation::GetApiPtrByHash<obfuscation::FnvHash(#name)>(dll_name)))
```

Win32API Hashing Obfuscationの実装

メタプログラミングとプリプロセッサによるソースコード生成イメージ

```
MessageBoxA(0, "World!", "Hello", MB_OK);
```

マクロ適用
（ソースコード書き換え）

```
OBF_CALL(L"user32.dll", MessageBoxA)(0, "World!", "Hello", MB_OK);
```

コンパイル
（メタプログラミングによるソースコード生成）

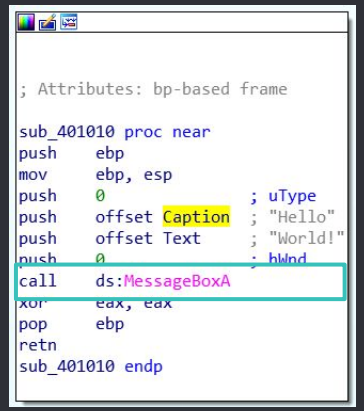
```
static void* function_pointer = GetApiPtrByHash(0x23a979e4, L"user32.dll");
function_pointer(0, "World!", "Hello", MB_OK);
```

Win32API Hashing Obfuscationの実装

コンパイル結果の比較

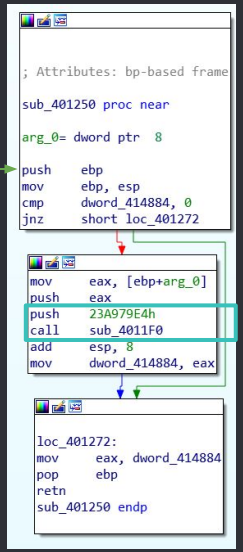
```
MessageBoxA(0, "World!", "Hello", MB_OK);
```

```
OBF_CALL(L"user32.dll", MessageBoxA)(0, "World!", "Hello", MB_OK);
```



API名が見える

API名が見えない



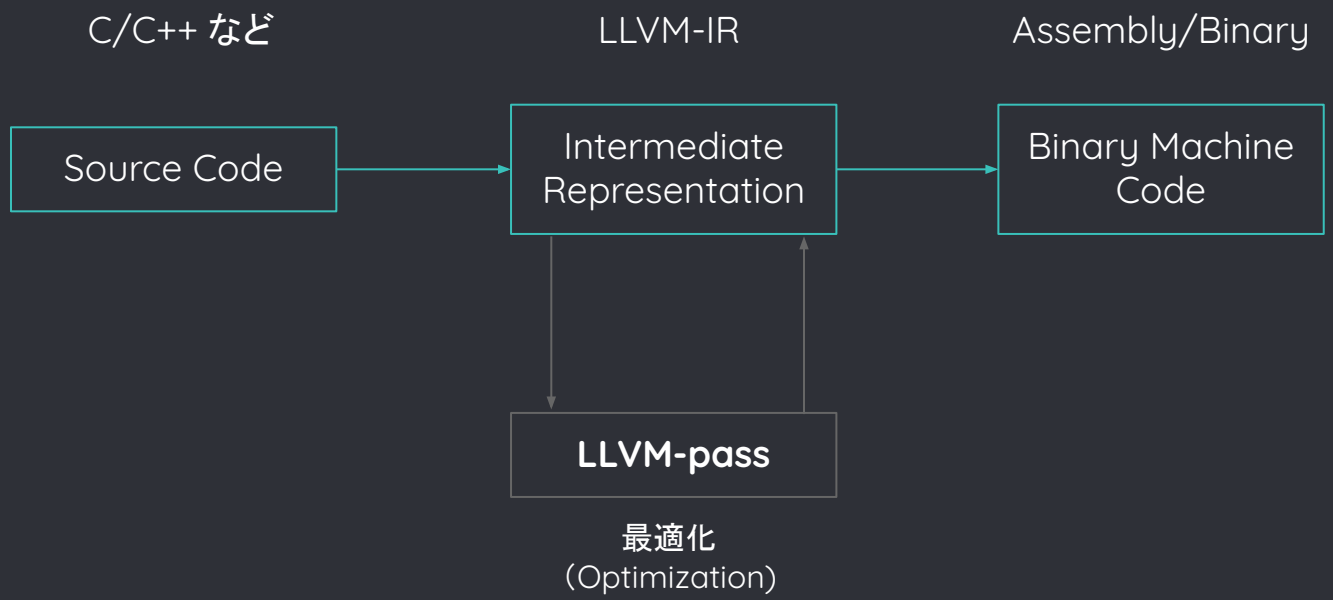
ハッシュ値による
API解決

難読化の実装

中間表現レイヤー

LLVM-pass

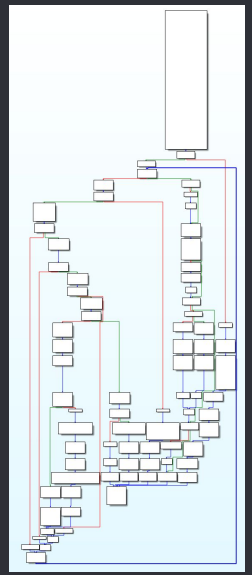
LLVM-passはコンパイラの最適化機能で用いられる。[11]
Intermediate Representationにおける難読化の実装ではLLVM-passの拡張が利用される。



Control Flow Flattening

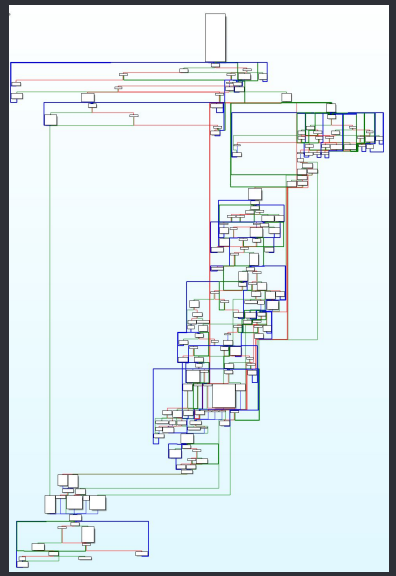
LLVM-passを利用したOSSからOLLVM[13]を選定して利用。

EmoCheck main関数
(難読化なし)



Transform

EmoCheck main関数
(Control Flow Flatten)



Strings Obfuscation

Emotetではソースコードレイヤーでの難読化が採用されていたがEmoCheckではHikari[14]のIntermediate Representationレイヤーでの難読化を採用。

難読化前

```
parm = &byte_1400D5774;  // "quiet"
```

```
                          q  u  i  e   t  \0
14000D5700   00 00 00 00 71 75 69 65  74 00 00 00 00 00 00 00
```

難読化後

```
byte_1400D5774 ^= 0xB0u;
byte_1400D5775 ^= 0x96u;
byte_1400D5776 ^= 0xDEu;
byte_1400D5777 ^= 0x13u;
byte_1400D5778 ^= 0xDAu;
byte_1400D5779 ^= 0x3Du;
parm = &byte_1400D5774;  // quiet
```

文字列を1byteずつでXORする復号する命令を関数の先頭に追加

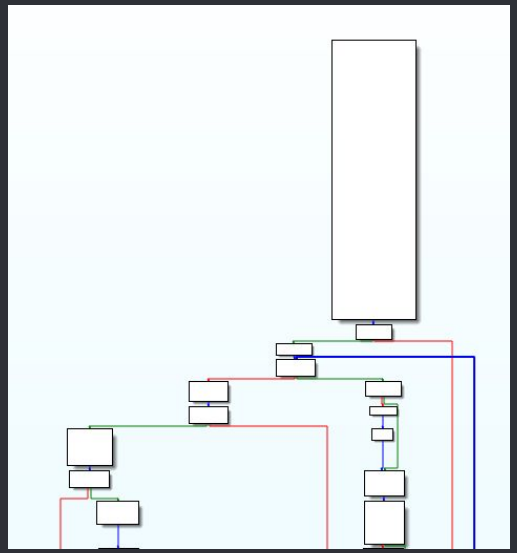
```
14000D5700   00 00 00 00 C1 E3 B7 76  AE 3D 00 00 00 00 00 00
```

元の文字列をランダムなbyteでXORしたデータに書き換え

Strings Obfuscation

最適化処理で削除することができない命令を多数追加することにより、他の中間言語レイヤの難読化処理の効果の増加が見込める。

EmoCheck main関数
(難読化なし)



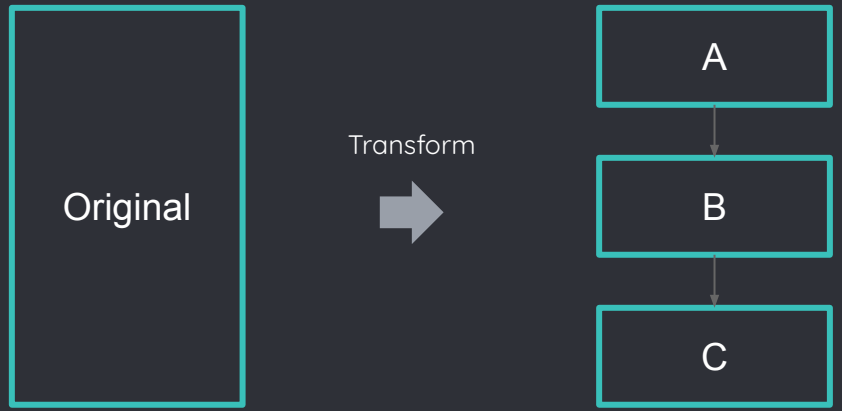
Transform

EmoCheck main関数 (String Obfuscation)

文字列を復号する処理

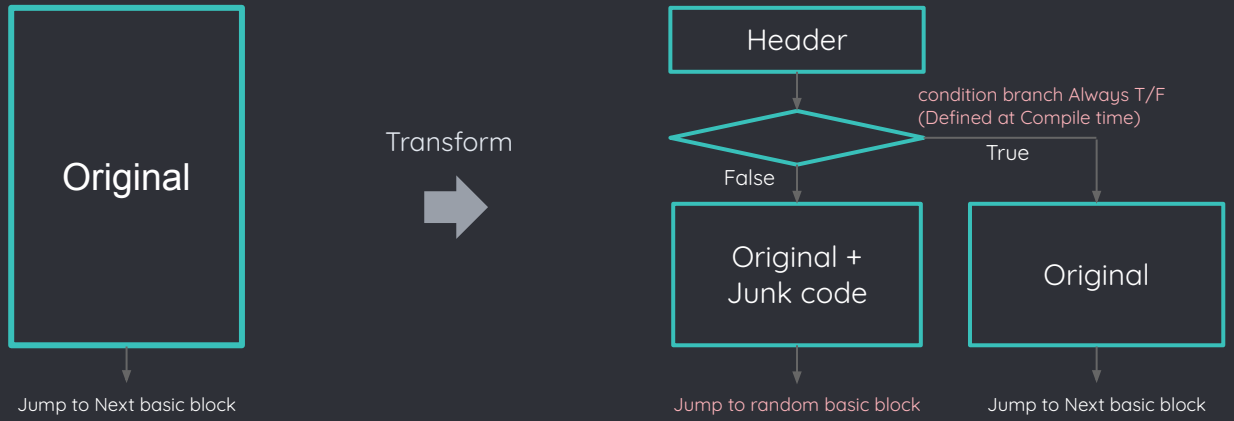
Split Basic Blocks

EmoCheckではControl Flow Flattenをより複雑化するために、O-LLVM[13]のBasic Blockの分割を採用。
String Obfuscationで生成された多数の命令を分割することを狙っている。



Bogus Control Flow

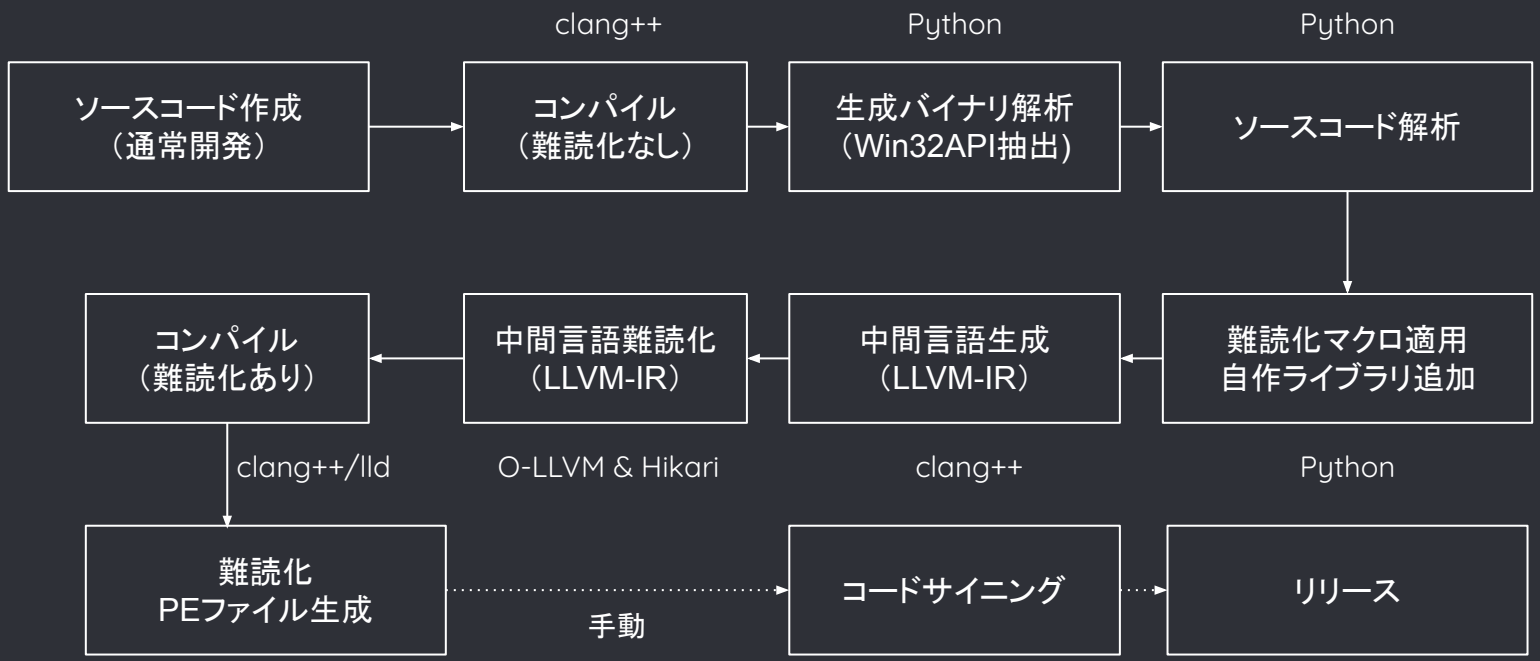
EmoCheckではControl Flow Flattenをより複雑化するために、Hikari[14]の Bogus Control Flowを採用。String Obfuscation + Split Basic Blocksで生成された多数のBasic Blockに一意に決まるかつ削除が困難な条件分岐を追加しロジックを複雑化。



難読化の実装

ビルドの自動化

難読化ビルドのパイプライン



Step1: 利用するWin32APIの特定

ソースコードへ難読化マクロ適用を自動化するため、通常コンパイルしたEファイルのIATから、利用しているWin32APIを特定。

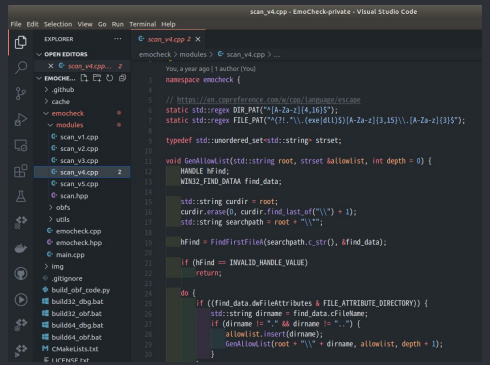
ソースコード作成
（通常開発）

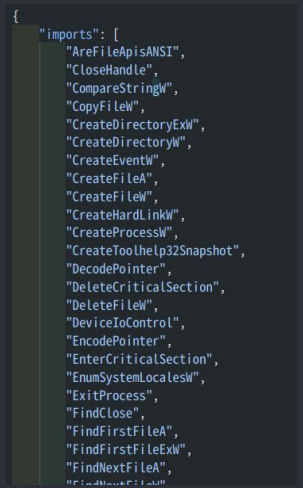
→

コンパイル
（難読化なし）

→

生成バイナリ解析
（Win32API抽出）





Step2: ソースコードへの難読化マクロ適用

ソースコードへ難読化マクロ適用し、難読化ライブラリを読み込み設定を追加。

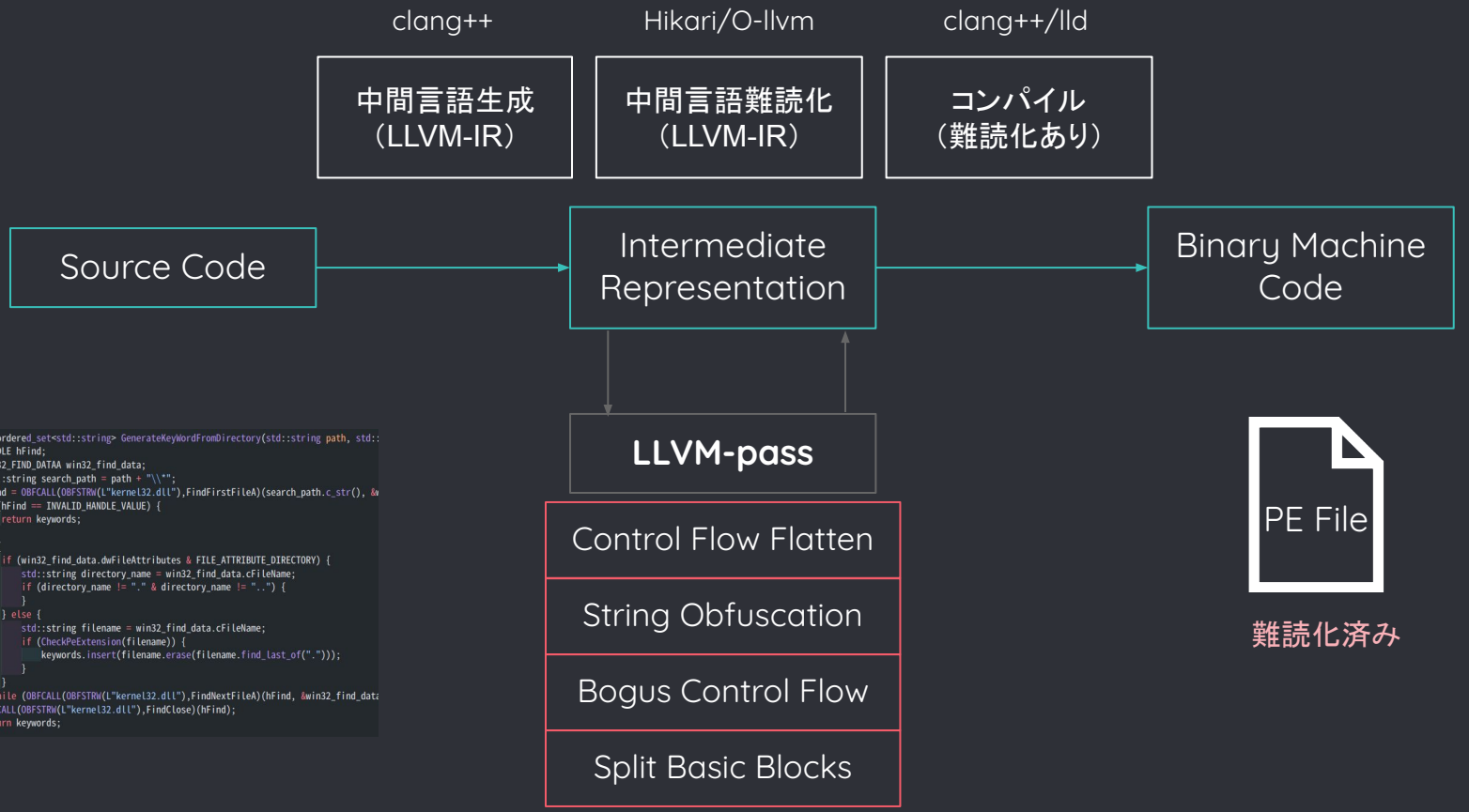
Pythonスクリプトによるソースコード変換前

```
HANDLE hFind;
WIN32_FIND_DATAA win32_find_data;
std::string search_path = path + "\\*";
hFind = FindFirstFileA(search_path.c_str(), &win32_find_data);
/* snip */
FindClose(hFind);
/* snip */
```

Pythonスクリプトによるソースコード変換後

```
HANDLE hFind;
WIN32_FIND_DATAA win32_find_data;
std::string search_path = path + "\\*";
                                  マクロ適用
hFind = OBFCALL(OBFSTRW(L"kernel32.dll"),FindFirstFileA)(search_path.c_str(), &win32_find_data);
/* snip */
                                  マクロ適用
OBFCALL(OBFSTRW(L"kernel32.dll"),FindClose)(hFind);
return keywords;
/* snip */
```

Step3: Intermediate Representationレイヤーの難読化

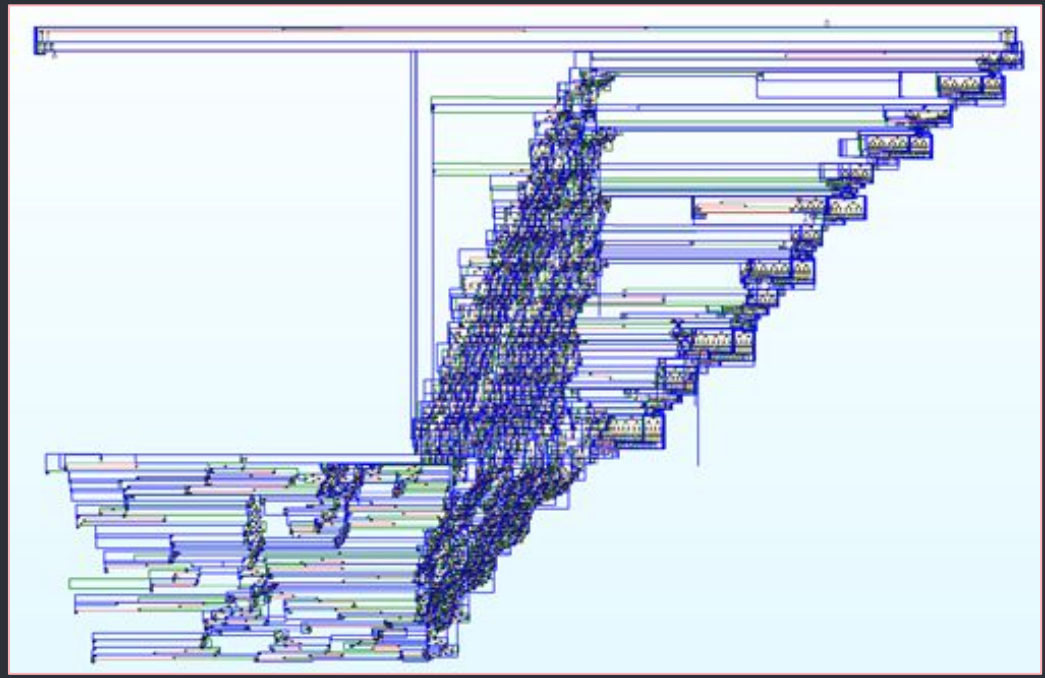



```
std::unordered_set<std::string> GenerateKeyWordFromDirectory(std::string path, std::
    HANDLE hFind;
    WIN32_FIND_DATAA win32_find_data;
    std::string search_path = path + "\\*";
    hFind = OBFCALL(OBFSTRW(L"kernel32.dll"),FindFirstFileA)(search_path.c_str(), &
    if (hFind == INVALID_HANDLE_VALUE) {
        return keywords;
    }
    do {
        if (win32_find_data.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            std::string directory_name = win32_find_data.cFileName;
            if (directory_name != "." & directory_name != "..") {
            }
        } else {
            std::string filename = win32_find_data.cFileName;
            if (CheckPeExtension(filename)) {
                keywords.insert(filename.erase(filename.find_last_of(".")));
            }
        }
    } while (OBFCALL(OBFSTRW(L"kernel32.dll"),FindNextFileA)(hFind, &win32_find_data
    OBFCALL(OBFSTRW(L"kernel32.dll"),FindClose)(hFind);
    return keywords;
}
```

難読化の比較

EmoCheck main関数（難読化前）

EmoCheck main関数（難読化後）



EmoCheckの難読化

Emotetよりは複雑になった

- max(関数のBasic Block数) > 4000
  - IDA7.7(Hex-rays cloud decompiler)ではdecompile不可
  - Ghidraで最大15000行/関数のpseudo-code

- ソースコードレイヤ/中間言語レイヤの難読化を適切に組み合わせによる難読化効果Up!

- Take-away

Take-Away

- 検知手法の紹介
  - マルウェアの目的にあわせて、検知手法を決める
  - インシデント調査やフォレンジックへ応用可
- 検知ツールの開発
  - 簡単に使用できるツールがないと調査できない人も多い
  - **Human vs Humanのいたちごっこ**
- 難読化手法とその実装例の理解
  - マルウェアに利用される難読化の多くは既存手法の組み合わせ
  - 複数手法を組み合わせることで難読化は強化される
  - 難読化を解除するために必要な知見
  - 難読化を利用する敷居は下がっている

- Any Questions?

References

Emotetについて

K. Sajo and S. Sasada, "とあるEmotetの観測結果", JSAC 2021

https://jsac.jpcert.or.jp/archive/2021/pdf/JSAC2021_104_sajo-sasada_jp.pdf

Cryptolaemus - Pastedump (accessed 2022-01-16)

https://paste.cryptolaemus.com/

L.Nagy, "Exploring Emotet, an elaborate everyday enigma", VirusBulletin 2019

https://www.virusbulletin.com/virusbulletin/2019/10/vb2019-paper-exploring-emotet-elaborate-everyday-enigma/

流行マルウェア「EMOTET」の内部構造を紐解く- MSBD Blog (accessed 2022-01-16)

https://www.mbsd.jp/blog/20181225_2.html

JPCERTCC/EmoCheck - GitHub (accessed 2022-01-16)

https://github.com/JPCERTCC/EmoCheck

References

難読化の分類について

[1] Collberg et al. A Taxonomy of Obfuscating Transformations. Jan, 1997

https://researchspace.auckland.ac.nz/bitstream/handle/2292/3491/TR148.pdf

[2] Banescu. A Tutorial on Software Obfuscation. 2017

https://mediatum.ub.tum.de/doc/1367533/file.pdf

[3] Xu et al. Layered obfuscation: a taxonomy of software obfuscation techniques for layered security. Apr,2020

https://cybersecurity.springeropen.com/articles/10.1186/s42400-020-00049-3

[4] Transformations -Tigress (accessed 2022-01-16)

https://tigress.wtf/transformations.html

References

Metaprogramingによる難読化について

[5] Template Metaprograming - Wikipedia (accessed 2022-01-16)

https://en.wikipedia.org/wiki/Template_metaprogramming

[6] constexpr (C++) - Microsoft Technical documentation (accessed 2022-01-16)

https://docs.microsoft.com/en-us/cpp/cpp/constexpr-cpp

[7] Samuel, N. and Filip, A., Binary code obfuscation through C++ templatemetaprogramming, Proceedings of the 4th INForum, 2012

https://www.researchgate.net/publication/233997753_Binary_code_obfuscation_through_C_template_metaprogramming

[8] S.Andrivet, "C++11 metaprogramming applied to software obfuscation", BlackHat EU 2014

https://www.blackhat.com/docs/eu-14/materials/eu-14-Andrivet-C-plus-plus11-Metaprogramming-Applied-To-software-Obfuscation-wp.pdf

[9] andrivet / ADVobfuscator - GitHub (accessed 2022-01-16)

https://github.com/andrivet/ADVobfuscator

[10] fritzone / obfy - GitHub (accessed 2022-01-16)

https://github.com/fritzone/obfy

References

Intermediate Representationによる難読化について

[11] Writing an LLVM Pass - LLVM (accessed 2022-01-16)

https://llvm.org/docs/WritingAnLLVMPass.html

[12] Junod, P., Rinaldini, J., Wehrli, J. and Michielin, J.: Obfuscator-LLVM: Software Protection for the Masses, Proceedings of the 1st International Workshop on Software Protection, SPRO'15, Piscataway, NJ, USA, IEEE Press, pp. 3-9 (online), available from <http://dl.acm.org/citation.cfm?id=2821429.2821434> (2015).

[13] obfuscator-llvm / obfuscator -GitHub (accessed 2022-01-16)

https://github.com/obfuscator-llvm/obfuscator

[14] HikariObfuscator / Hikari - GitHub (accessed 2022-01-16)

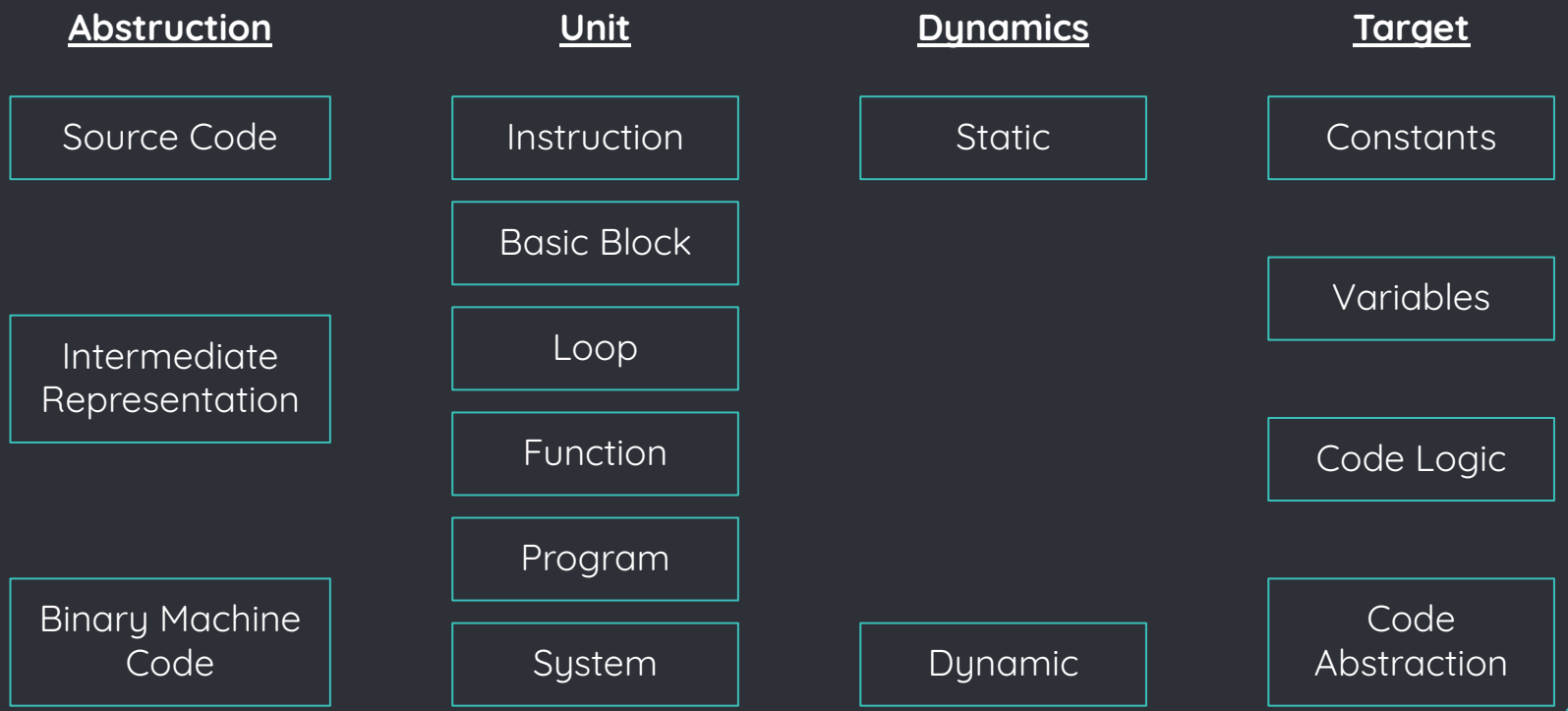
https://github.com/HikariObfuscator/Hikari

[15] T. Haruyama. Defeating APT10 Compiler-level Obfuscations. Virus Bulletin Conference 2019

https://www.virusbulletin.com/uploads/pdf/conference_slides/2019/VB2019-Haruyama.pdf

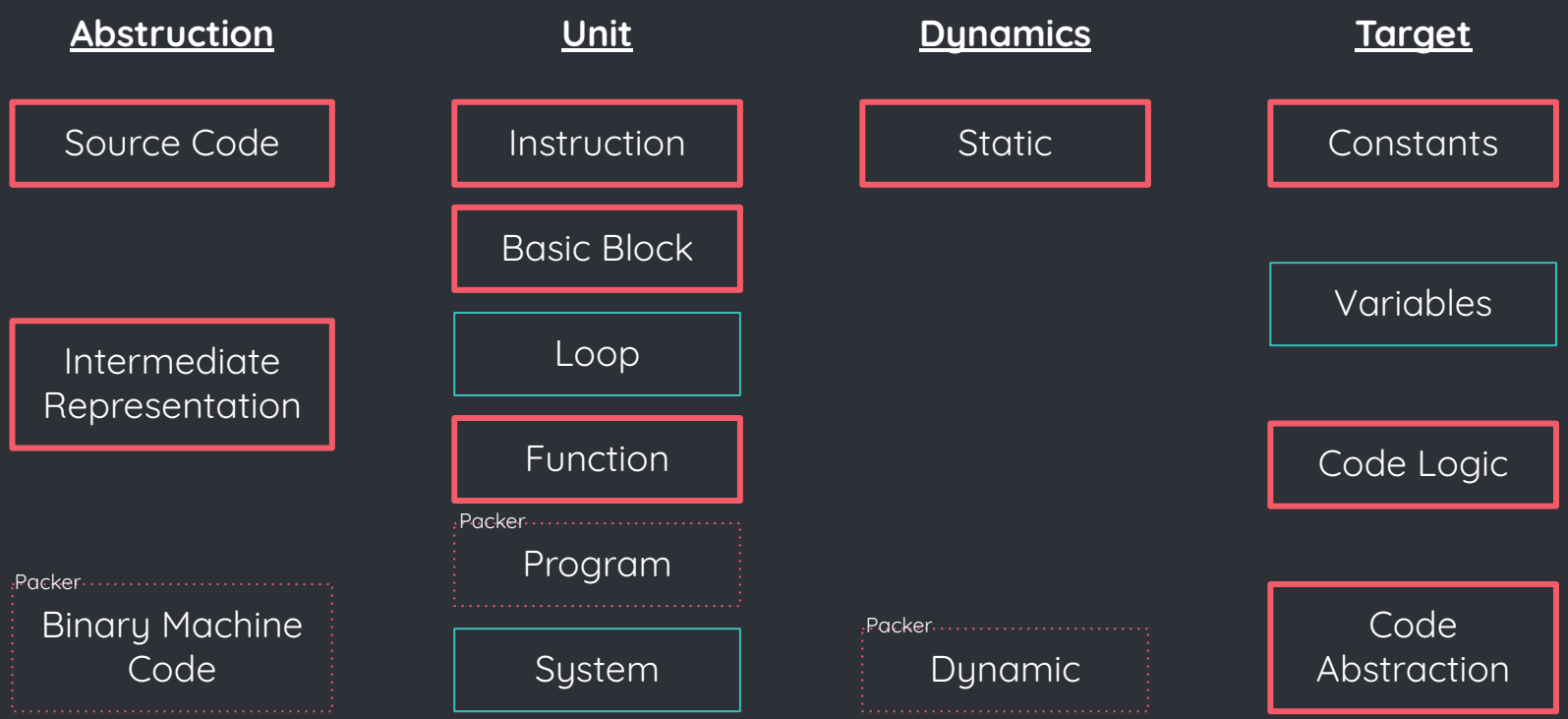
Appendix: 一般的な難読化手法の分類

難読化処理を施す抽象レイヤ、単位、手法、対象によって分類される。 [1] [2] [3]



Appendix: Emotet(Loader)に施されている難読化の分布

Emotetの難読化がカバーする範囲。Dropper(e.g. デコイ文書)やPackerはスコープ外。



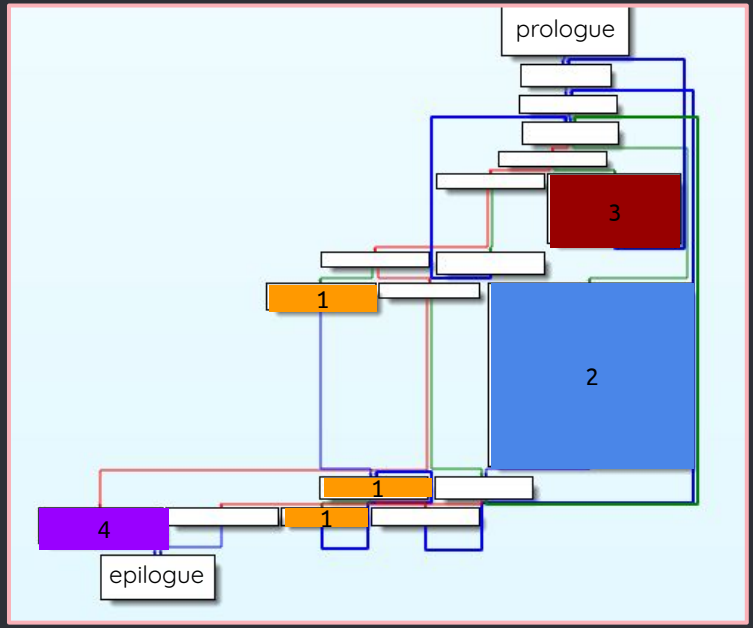
Appendix: Control Flow Flattening

Emotetでは単一のRouting変数によってBasic Blockの実行順を難読化。

Emotetの関数 CFF解除（pseudo-code）

```
BOOL mal_DeletePersistentRegistry(){
    LPCWSTR lpValueName;
    const WCHAR *reg_path;
    HKEY phkResult;
    BOOL status;
    lpValueName = NULL;

    // Block 1: Get value name from global data section.
    for (int i = pGlobalData + 564; *i != '\\'; i += 2 );
    lpValueName = (i + 2);

    // Block 2: Open Windows registry key handle
    reg_path = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"
    res = RegCreateKeyExW(0, lpSubKey, 0, 0, 0, regsam, 0, phkResult, 0);
    HeapFree(reg_path);

    if (res == ERROR_SUCCESS)
        // Block 3: Delete registry value
        status = RegDeleteValueW(phkResult, lpValueName) == 0;

    // Block4: Close Windows registry key handle
    RegCloseKey(phkResult);
    return status;
}
```

Emotetの関数 (Control Flow Graph)



Appendix: Function Argument Randomization(アセンブリ)

関数コールの引数に不必要な定数と追加し、引数の順番を入れ替える。追加された引数はプログラムの動作には無関係であり、呼び出し元の関数のスタックに保持される。

難読化後

```
.text:10001DB2                          mal_DeletePersistentReg proc near       ; CODE XREF: sub_1001A614+4B6↓p
.text:10001DB2
.text:10001DB2                          junk_15         = dword ptr -70h
.text:10001DB2                          junk_10         = dword ptr -6Ch
.text:10001DB2                          regsam          = dword ptr -68h
.text:10001DB2                          junk_19         = dword ptr -64h
.text:10001DB2                          junk_23         = dword ptr -60h
.text:10001DB2                          junk_6          = dword ptr -5Ch
.text:10001DB2                          junk_5          = dword ptr -58h
.text:10001DB2                          junk_14         = dword ptr -54h
.text:10001DB2                          hKey            = dword ptr -50h
.text:10001DB2                          junk_12         = dword ptr -4Ch
.text:10001DB2                          junk_9          = dword ptr -48h
.text:10001DB2                          junk_8          = dword ptr -44h
.text:10001DB2                          junk_18         = dword ptr -40h
.text:10001DB2                          junk_21         = dword ptr -3Ch
.text:10001DB2                          junk_17         = dword ptr -38h
.text:10001DB2                          junk_16         = dword ptr -34h
.text:10001DB2                          junk_7          = dword ptr -30h
.text:10001DB2                          junk_13         = dword ptr -2Ch
.text:10001DB2                          junk_11         = dword ptr -28h
.text:10001DB2                          junk_4          = dword ptr -24h
.text:10001DB2                          junk_20         = dword ptr -20h
.text:10001DB2                          junk_22         = dword ptr -1Ch
.text:10001DB2                          phkResult       = dword ptr -18h
.text:10001DB2                          status          = dword ptr -14h
.text:10001DB2                          junk_3          = dword ptr -10h
.text:10001DB2                          junk_1          = dword ptr -0Ch
.text:10001DB2                          junk_2          = dword ptr -8
.text:10001DB2                          junk_0          = dword ptr -4
.text:10001DB2
.text:10001DB2 000 83 EC 70                             sub     esp, 70h
.text:10001DB5 070 C7 44 24 60 20 CC CF 00              mov     [esp+70h+junk_3], 0CFCC20h
.text:10001DBD 070 33 C0                                xor     eax, eax
.text:10001DBF 070 89 44 24 6C                          mov     [esp+70h+junk_0], eax
.text:10001DC3 070 33 D2                                xor     edx, edx
.text:10001DC5 070 C7 44 24 64 6C F0 C1 00              mov     [esp+70h+junk_1], 0C1F06Ch
.text:10001DCD 070 C7 44 24 68 9C 26 71 00              mov     [esp+70h+junk_2], 71269Ch
.text:10001DD5 070 C7 44 24 20 92 41 1F 00              mov     [esp+70h+hKey], 1F4192h
.text:10001DDD 070 C1 64 24 20 0B                       shl     [esp+70h+hKey], 0Bh
.text:10001DE2 070 53                                   push    ebx
.text:10001DE3 074 55                                   push    ebp
.text:10001DE4 078 56                                   push    esi
.text:10001DE5 07C 57                                   push    edi
.text:10001DE6 080 89 44 24 6C                          mov     [esp+80h+status], eax
.text:10001DEA 080 BF 70 CE 62 0C                       mov     edi, 0C62CE70h
.text:10001DEF 080 8B 44 24 30                          mov     eax, [esp+80h+hKey]
.text:10001DF3 080 6A 14                                push    14h
```

Junk local variables for Function Argument Randamization

Junk引数を含む関数呼び出し

```
push    esi             ; lpSubKey
push    [esp+84h+junk_14] ; a14
push    [esp+88h+hKey]  ; hKey
push    ecx             ; a12
push    [esp+90h+junk_13] ; a11
push    [esp+94h+junk_12] ; a10
push    [esp+98h+junk_11] ; a9
push    ecx             ; a8
push    ecx             ; a7
push    eax             ; phkResult
push    [esp+0A8h+junk_10] ; a5
push    ecx             ; a4
push    [esp+0B0h+junk_9] ; a3
push    ecx             ; a2
push    [esp+0B8h+junk_8] ; a1
mov     edx, [esp+0BCh+junk_4] ; junk
mov     ecx, [esp+0BCh+regsam] ; regsam
call    mal_RegCreateKeyExW
```

Junk Aruguments

```
push    ebx
push    [esp+84h+junk_17] ; a2
push    [esp+88h+junk_16] ; a1
mov     edx, [esp+8Ch+junk_15]
mov     ecx, [esp+8Ch+phkResult] ; hKey
call    mal_RegDeleteValueW
```

Junk Aruguments

```
push    offset enc_registry_path
push    [esp+84h+junk_7] ; junk2
mov     edx, [esp+88h+junk_6] ; junk1
mov     ecx, [esp+88h+junk_5] ; junk0
call    mal_DecryptString ; SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

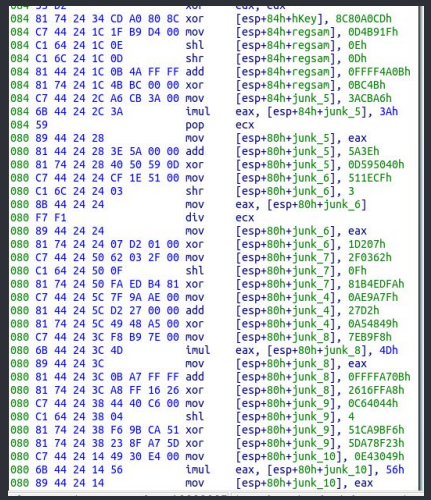
Junk Aruguments

Appendix: EmoCheckで採用しなかった難読化

Mixed Boolean Arithmetic

コンパイラ/デコンパイラの最適化処理で難読化が消失する可能性があるため。

IDA View



```
084 81 74 24 34 CD A0 80 8C xor     [esp+84h+hKey], 8C80A0CDh
084 C7 44 24 1C 1F B9 D4 00 mov     [esp+84h+regsam], 0D4B91Fh
084 C1 64 24 1C 0E          shl     [esp+84h+regsam], 0Eh
084 C1 6C 24 1C 0D          shr     [esp+84h+regsam], 0Dh
084 81 44 24 1C 0B 4A FF FF add     [esp+84h+regsam], 0FFFF4A0Bh
084 81 74 24 1C 4B BC 00 00 xor     [esp+84h+regsam], 0BC4Bh
084 C7 44 24 2C A6 CB 3A 00 mov     [esp+84h+junk_5], 3ACBA6h
084 6B 44 24 2C 3A          imul    eax, [esp+84h+junk_5], 3Ah
084 59                      pop     ecx
080 89 44 24 28             mov     [esp+80h+junk_5], eax
080 81 44 24 28 3E 5A 00 00 add     [esp+80h+junk_5], 5A3Eh
080 81 74 24 28 40 50 59 0D xor     [esp+80h+junk_5], 0D595040h
080 C7 44 24 24 CF 1E 51 00 mov     [esp+80h+junk_6], 511ECFh
080 C1 6C 24 24 03          shr     [esp+80h+junk_6], 3
080 8B 44 24 24             mov     eax, [esp+80h+junk_6]
080 F7 F1                   div     ecx
080 89 44 24 24             mov     [esp+80h+junk_6], eax
080 81 74 24 24 07 D2 01 00 xor     [esp+80h+junk_6], 1D207h
080 C7 44 24 50 62 03 2F 00 mov     [esp+80h+junk_7], 2F0362h
080 C1 64 24 50 0F          shl     [esp+80h+junk_7], 0Fh
080 81 74 24 50 FA ED B4 81 xor     [esp+80h+junk_7], 81B4EDFAh
080 C7 44 24 5C 7F 9A AE 00 mov     [esp+80h+junk_4], 0AE9A7Fh
080 81 44 24 5C D2 27 00 00 add     [esp+80h+junk_4], 27D2h
080 81 74 24 5C 49 48 A5 00 xor     [esp+80h+junk_4], 0A54849h
080 C7 44 24 3C F8 B9 7E 00 mov     [esp+80h+junk_8], 7EB9F8h
080 6B 44 24 3C 4D          imul    eax, [esp+80h+junk_8], 4Dh
080 89 44 24 3C             mov     [esp+80h+junk_8], eax
080 81 44 24 3C 0B A7 FF FF add     [esp+80h+junk_8], 0FFFFA70Bh
080 81 74 24 3C A8 FF 16 26 xor     [esp+80h+junk_8], 2616FFA8h
080 C7 44 24 38 44 40 C6 00 mov     [esp+80h+junk_9], 0C64044h
080 C1 64 24 38 04          shl     [esp+80h+junk_9], 4
080 81 74 24 38 F6 9B CA 51 xor     [esp+80h+junk_9], 51CA9BF6h
080 81 74 24 38 23 8F A7 5D xor     [esp+80h+junk_9], 5DA78F23h
080 C7 44 24 14 49 30 E4 00 mov     [esp+80h+junk_10], 0E43049h
080 6B 44 24 14 56          imul    eax, [esp+80h+junk_10], 56h
080 89 44 24 14             mov     [esp+80h+junk_10], eax
```

Hex-rays Decompiler

```
15  junk_3 = 13618208;
16  junk_0 = 0;
17  junk_1 = 12709996;
18  junk_2 = 7415452;
19  status = 0;
20  route = 0xC62CE70;
21  lpValueName = 0;
```

Function Argument Randomization

難読化実装当時にまだEmotetの難読化に含まれていなかった。

Appendix: EmoCheckの難読化

EmoCheckの検知ロジックを保護する機能は果たせていると考えられる。

Abstruction	Unit	Dynamics	Target
Source Code	Instruction	Static	Constants
	Basic Block		Variables
Intermediate Representation	Loop		
	Function		Code Logic
	Program		
Binary Machine Code	System	Dynamic	Code Abstraction