

MWS Cup 2016

課題2: 静的解析

中津留 勇

2016/12/07

課題2 担当



課題2 の変わらぬテーマ

- 静的解析を通じ

マルウェアを正しく理解する

最新情報を得る

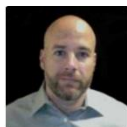
実務に近い作業

A nighttime photograph of a city street, likely in New York City, featuring light trails from traffic and illuminated buildings. A semi-transparent blue overlay covers the left side of the image, containing the title text.

MWS Cup 2016 課題2 解説

Tick/Daserf

• 日本国内で数年前から発生していた標的型攻撃



Jon DiMaggio
View Profile

Symantec Official Blog

Tick cyberespionage group zeros in on J

Compromised websites and spear-phishing emails used to infect

By: Jon DiMaggio SYMANTEC EMPLOYEE

Created 28 Apr 2016 0 Comments 简体中文, 繁體中文, 日本語, 한국어



<http://www.symantec.com/connect/blogs/tick-cyberespionage-group-zeros-japan>

標的型攻撃に用いられるマルウェア、Daserfとは

Daserf はバックドア機能を有するマルウェアで、Nioupale とも呼ばれています。Daserf については、2016年5月に Symantec がブログⁱⁱで報告していますが、それまではセキュリティベンダによる報告はほとんどなく、このマルウェアの存在自体、広く知られているとは言えない状況でした。一方、ラックは2013年1月頃以降に対応した複数の標的型攻撃事案において Daserf を確認しており、これらの分析を続けてきました。その結果、Daserf が日本の重要インフラを標的とした攻撃者に使用され、長期間にわたって標的組織に潜伏しつつ活動している可能性が高いことが明らかになりました。

図1は、ラックが対応した事案において Daserf が使われた業種を分類したグラフです。グラフ外周右側の枠に含まれるのが重要インフラに属する業種ⁱⁱⁱで、56%と過半数を占めていることがわかります。外周左側の枠は重要インフラで利用される機器を製造する事業者で、これらを含めるとすべての事案が重要インフラに直接的、間接的に関連していることがわかります。このことから、Daserf を使う攻撃者は、少なくとも日本においては重要インフラやその関連企業をターゲットとしている可能性が高いと考えられます。

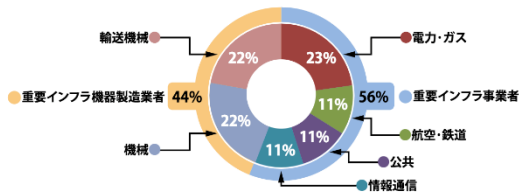


図1 ラックが対応した標的型攻撃事案のターゲット組織

http://www.lac.co.jp/security/report/pdf/20160802_cgview_vol2_a001t.pdf

課題2

1. コード難読化
2. 暗号アルゴリズム（選択式）
3. 暗号化/エンコード処理
4. 暗号化されたデータの復号
5. 設定情報の調査

ポイント

難しく捉えない

- アセンブリ言語そのものは単純

分かりやすい情報を探す

- 文字列、定数、API

すべてのコードを読まない

- 終わりません

1. コード難読化

- 難読化が行われている箇所を探す
 - 何が「普通のコード」かを知っておく必要がある

```
004232A0      public start
004232A0      start      proc near
004232A0 000      push     ebp
004232A1 004      mov     ebp, esp
004232A3 004      add     esp, 0FFFFFF0h
004232A6 014      mov     eax, offset dword_423208
004232AB 014      call   Sysinit:.__linkproc__ InitExe(void *)
004232B0 014      xor     eax, eax
004232B2 014      push   ebp
004232B3 018      push   offset loc_423326
004232B8 01C      push   dword ptr fs:[eax]
004232BB 020      mov     fs:[eax], esp
004232BE 020      sub     esi, 7
004232C1 020      add     eax, ecx
004232C3 020      dec     ebx
004232C4 020      jmp    short loc_4232E6
```


1. コード難読化

- 答えはすぐ下に

無駄なデータを挿入し、
jmp 命令で飛ばしながら
実行する

```
CODE:004232C3 020          dec     eax
CODE:004232C4 020          jmp     eax
CODE:004232C4          ; -----
CODE:004232C6 020          dw     0C2A11h
CODE:004232C8 020          dd     0F303AD9Fh, 09F8E6h, 1DD5589Ah, 586627h
CODE:004232C8          dd     0DF0B6BEh, 0DF0B6E65h
CODE:004232E4 020          db     65h, 6Eh
CODE:004232E6          ; -----
CODE:004232E6          loc_4232E6:          ; CODE XREF: start+2
CODE:004232E6 020          add     ebx, 1
CODE:004232E9 020          add     esi, 3
CODE:004232EC 020          sub     eax, ecx
CODE:004232EE 020          add     esi, 4
CODE:004232F1 020          jmp     short loc_423313
CODE:004232F1          ; -----
CODE:004232F3 020          db     6Fh
CODE:004232F4 020          dd     8263B0F7h, 0BF8C3995h, 51C1A1B9h, 0BA881Eh
CODE:004232F4          dd     5D0C947Dh, 5DB96BDDh
CODE:00423310 020          db     0DDh, 6Bh, 0B9h
CODE:00423313          ; -----
CODE:00423313          loc_423313:          ; CODE XREF: start+5
CODE:00423313 020          call   sub_420534
```

1. コード難読化

- API 呼び出しの難読化も

```
004200C1 408      push    offset aInte    ; "Inte"
004200C4      eax, [ebp+ProcName]
004200C7      eax    ; lpString1
004200CA      lstrcpyA_0
004200CD      offset aRnet    ; "rnet"
004200D0      eax, [ebp+ProcName]
004200D3      ; lpString1
004200D6      lstrcatA_0
004200E3 408      push    offset aConn    ; "Conn"
004200E8 40C      lea    eax, [ebp+ProcName]
004200EE 40C      push    eax    ; lpString1
004200EF 410      call   lstrcatA_0
004200F4 408      push    offset aEcta    ; "ectA"
                                eax, [ebp+ProcName]
                                eax    ; lpString1
                                lstrcatA_0
                                eax, [ebp+ProcName]
                                eax    ; lpProcName
                                eax, [ebp+hModule]
                                eax    ; hModule
                                GetProcAddress_0
00420115 408      mov    ds:DWORD_425F98, eax
```

文字列を分割して
分かりづらくして
いる

動的な名前解決をする
ことで API 名が見え
ないようにしている

2. 暗号アルゴリズム

- AES, DES, 3DES, RC4, TEA?

```
CODE:004198B0 120      jl      loc_419955
CODE:004198B6 120      inc     eax
CODE:004198B7 120      mov     [ebp+var_18], eax
CODE:004198BA 120      mov     [ebp+var_10], 0
CODE:004198C1
CODE:004198C1      loc_4198C1:                                ; CODE XREF: sub_41
CODE:004198C1 120      inc     [ebp+var_1A]
CODE:004198C4 120      xor     eax, eax
CODE:004198C6 120      mov     al, [ebp+var_1A]
CODE:004198C9 120      mov     al, [ebp+eax+var_11A]
CODE:004198D0 120      mov     [ebp+var_11], al
CODE:004198D3 120      mov     al, [ebp+var_11]
CODE:004198D6 120      add    [ebp+var_19], al
CODE:004198D9 120      xor     eax, eax
CODE:004198DB 120      mov     al, [ebp+var_19]
CODE:004198DE 120      mov     al, [ebp+eax+var_11A]
CODE:004198E5 120      xor     edx, edx
CODE:004198E7 120      mov     dl, [ebp+var_1A]
CODE:004198EA 120      mov     [ebp+edx+var_11A], al
CODE:004198F1 120      xor     eax, eax
CODE:004198F3 120      mov     al, [ebp+var_19]
CODE:004198F6 120      mov     dl, [ebp+var_11]
```

2. 暗号アルゴリズム

各選択肢の調査

- AES: SPN構造のブロック暗号
- DES、3DES: Feistel構造のブロック暗号
- RC4: ストリーム暗号
- TEA: Feistel構造のブロック暗号

アセンブリ言語を読み特徴を抽出

- `sub_41978C` を呼んでから、XOR のループ
- AES などで使用される定数を参照していない
- 1バイトずつ XOR していてストリームっぽい

2. 暗号アルゴリズム

• sub 41978C

0xFF までを配置
した配列を作成し
ている

```
• CODE:004197A3 018  
• CODE:004197A6 018  
• CODE:004197AD 018  
• CODE:004197B0 018  
• CODE:004197B7 018  
CODE:004197BB  
CODE:004197BB Loc_4197BB: CODE  
• CODE:004197BB 018 xor eax, eax  
• CODE:004197BD 018 mov al, [ebp+var_A]  
• CODE:004197C0 018 mov edx, [ebp+var_4]  
• CODE:004197C3 018 mov cl, [ebp+var_A]  
• CODE:004197C6 018 mov [edx+eax], cl  
• CODE:004197C9 018 inc [ebp+var_A]  
• CODE:004197CC 018 cmp [ebp+var_A], 0  
• CODE:004197D0 018 jnz short loc_4197BB
```

暗号アルゴリズムは RC4

3. 暗号化/エンコード処理

- 暗号なのかエンコードなのか、そして鍵や変換テーブルは何か

```
004144A4 000      push     ebp
004144A5 004      mov      eax, [ebp+String1]
004144A7 004      add      edx, 80h
004144AD 0B4      push     eax
004144AE 0B8      mov      eax, [ebp+String1]
004144B1 0B8      mov      eax, [ebp+String1]
004144B4 0B8      mov      eax, [ebp+String1]
004144B7 0B8      mov      eax, [ebp+String1]
004144B9 0B8      mov      eax, [ebp+String1]
004144BA 0B8      mov      eax, [ebp+String1]
004144BA ; -----
004144BC 0B8      mov      eax, [ebp+String1]
004144D4 ; -----
004144D4 ; -----
004144D4 Loc_4144D4:
004144D4 0B8      add      eax, [ebp+String1]
004144D7 0B8      sub      eax, [ebp+String1]
004144D9 0B8      jmp      eax
004144D9 ; -----
004144DB 0B8      db      0BDh
004144DC 0B8      dd      74C6
004144DC 0B8      dd      0C5A
00414500 0B8      db      0D7h
00414503 ; -----
0041706F 0C4      lea     eax, [ebp+String1]
00417075 0C4      mov     edx, 80h
0041707A 0C4      call   @ZeroMemory
0041707F 0C4      push   offset aGklmop_0 ; "GKLmop"
00417084 0C8      lea     eax, [ebp+String1]
                                ; lpString1
                                lstrcpyA_0
                                offset aQr6vqb_0 ; "qr6vQB"
                                eax, [ebp+String1]
                                ; lpString1
                                lstrcatA_0
                                offset aRswxyz0h_0 ; "RSwxyz0H"
                                eax, [ebp+String1]
                                ; lpString1
                                lstrcatA_0
                                offset a1234zabt_0 ; "1234ZabT"
                                eax, [ebp+String1]
                                ; lpString1
                                lstrcatA_0
                                offset aCdefg789_0 ; "cdefg789"
                                eax, [ebp+String1]
                                ; lpString1
                                lstrcatA_0
```

鍵または変換テーブルの
ような文字列

3. 暗号化/エンコード処理

- 具体的な処理を「さらっと」読み解く

```
CODE:0041717A      Loc_41717A:      ; C
CODE:0041717A 0C4      xor     eax, eax
CODE:0041717C 0C4      mov     [ebp+var_24], eax
CODE:0041717F      Loc_41717F:
CODE:0041717F 0C4      mov     ecx, [ebp+var_24]
CODE:00417182 0C4      mov     ecx, [ebp+var_24]
CODE:00417185 0C4      mov     ecx, [ebp+var_24]
CODE:00417189 0C4      mov     ecx, [ebp+var_24]
CODE:0041718C 0C4      mov     ecx, [ebp+var_24]
CODE:0041718F 0C4      cmp     al, [edx+ecx]
CODE:00417192 0C4      jz     short loc_417192
CODE:00417194 0C4      inc     [ebp+var_24]
CODE:00417197 0C4      cmp     [ebp+var_24], 40h
CODE:0041719B 0C4      jnz    short loc_41719B
CODE:0041719D      Loc_41719D:      ; C
CODE:0041719D 0C4      mov     al, byte ptr [ebp+var_24]
CODE:004171A0 0C4      mov     edx, [ebp+var_18]
CODE:004171A3 0C4      mov     [ebp+edx+var_28], al
CODE:004171A7 0C4      inc     [ebp+var_10]
CODE:004171AA 0C4      inc     [ebp+var_10]
CODE:004171AD 0C4      cmp     [ebp+var_10], 40h
CODE:004171B1 0C4      jnz    short loc_4171B1
CODE:004171B1 0E0      shl     ecx, 2
CODE:004171B3 0C4      xor     ebx, ebx
CODE:004171B5 0C4      mov     bl, [ebp+var_27]
CODE:004171B8 0C4      shr     ebx, 4
CODE:004171BB 0C4      or     cl, bl
CODE:004171BD 0C4      mov     [eax+ecx-1], cl
CODE:004171BF 0C4      lea    eax, [ebp+var_2C]
CODE:004171C1 0C4      call   System::16809
CODE:004171C3 0C4      mov     edx, [ebp+var_14]
CODE:004171C6 0C4      mov     cl, [ebp+var_27]
CODE:004171C9 0C4      shl     ecx, 4
CODE:004171CB 0C4      xor     ebx, ebx
CODE:004171CD 0C4      mov     bl, [ebp+var_26]
CODE:004171D0 0C4      shr     ebx, 2
CODE:004171D3 0C4      or     cl, bl
CODE:004171D5 0C4      mov     [eax+ecx], cl
CODE:004171D7 0C4      lea    eax, [ebp+var_2C]
CODE:004171D9 0C4      call   System::16809
CODE:004171DB 0C4      mov     edx, [ebp+var_14]
CODE:004171DE 0C4      mov     cl, [ebp+var_26]
CODE:004171E0 0C4      shl     ecx, 6
```

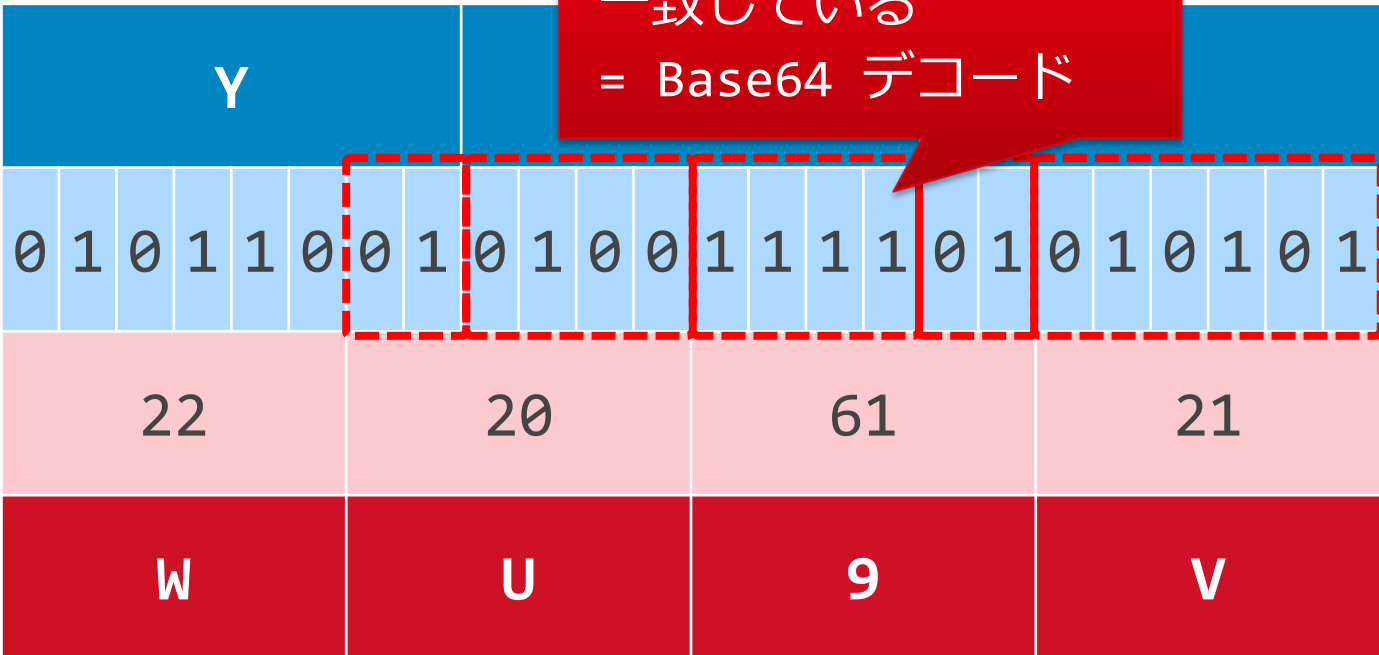
0x40 = 64

シフト演算が多い

3. 暗号化/エンコード処理

- Base64

シフトするビット数と一致している
= Base64 デコード



3. 暗号化/エンコード処理

- 具体的な処理を「さらっと」読み解く

```
0041721A 0C4      mov     edx, [ebp+var_4]
0041721D 0C4      mov     eax, offset dword_417308
00417222 0C4      call   __linkproc__L3trPos
00417227 0C4      mov
0041722A 0C4      cmp
0041722E 0C4      jz
00417230 0C4      mov
00417233 0C4      sub
00417236 0C4      inc
00417237 0C4      mov     [ebp+var_10], eax
00417237 00417300      dd     0FFFFFFFFh, 1
00417237 00417308      dword_417308      dd     21h
```

0x21 = !
Base64 のパディング

変則 Base64 デコード

変換テーブル:

GKLMopqr6vQBRSwxyz0H1234ZabTcdefg789^/CDAENOPUIVWYJhijklXstu5mnF!

4. 暗号化されたデータの復号

- デコードスクリプトの実装
 - 問2、3 の解析結果 + α で復号可能

```
00419C98 1134      lea    edx, [ebp+var_18]
00419C9B 1134      mov    eax, [ebp+arg_8]
00419C9E 1134      call  sub_41999C
00419CA3 1134      mov    [ebp+lpAddress], eax
00419CA7 1134      lea    ecx, [ebp+var_1C]
00419CAB 1134      mov    edx, [ebp+var_18]
00419CBE 1134      mov    eax, [ebp+lpAddress]
00419CBF 1134      call  sub_4144A4
```

途中で RC4 を
呼ぶ

変則 Base64

4. 暗号化されたデータの復号

• sub_41999C の前半

```
00419A44 438      push    offset aRtlCo ; "RtlCo"
00419A49 43C      lea     eax, [ebp+ProcName]
00419A4F      ; lpString1
00419A50      ; "mpre"
00419A55      ; "mpre"
00419A5A      ; lpString1
00419A60      ; "ssB"
00419A66      ; "ssB"
00419A6B 43C      lea     eax, [ebp+ProcName]
00419A71 43C      push    eax
00419A72 440      mov     eax, [ebp+var_8]
00419A77 438      mov     eax, [eax]
00419A7C 43C      push   eax
00419A82 43C      mov     eax, [ebp+var_14]
00419A83 440      push   eax
00419A88 438      mov     eax, [ebp+var_8]
00419A8E 438      mov     eax, [eax]
00419A8F 43C      push   eax
00419A92 43C      mov     eax, [ebp+var_4]
00419A93 440      push   eax
00419A94 43C      push   102h
00419A95 440      call   [ebp+RtlCompressBuffer]
```

RtlCompressBuffer
で圧縮

圧縮フォーマットは
LZNT1

4. 暗号化されたデータの復号

• sub_41999C の後半

RC4鍵:

bn32gkj324kgd43<dkh6Jdhghsd?dgh;y4dsgserd

```
00419B69 464          [ebp+var_18], eax
00419B6C 464          mov     eax, [ebp+var_18]
00419B6F 464          mov     byte ptr [eax], 1
00419B72 464          mov     eax, ds:off_4243A0
00419B77 464          push   eax
00419B78 468          mov     edx, [ebp+var_18]
00419B7B 468          inc     edx
00419B7C 468          mov     ecx, [ebp+dwSize]
00419B7E 468          mov     eax, [ebp+var_14]
00419B81 468          call   sub_419888
```

RC4 で暗号化しているのは
2バイト目から

4. 暗号化されたデータの復号

変則 Base64 デコード



2バイト目以降を RC4 復号



復号したデータを LZNT1 で展開

4. 暗号化されたデータの復号

```
In [1]: %paste
import string, base64, lznt1, re, struct
from Crypto.Cipher import ARC4
standard_b64table = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
daserf19_b64table = 'GKLMopqr6vQBRswxyz0H1234ZabTcdefg789^/CDAENOPUIVWYJhijklXstu5mnF!'
rc4key = 'bn32gkj324kgd43<dkh6Jdhghsd?dgh;y4dsgserd'

def custom_b64decode(s, custom_table):
    s = s.translate(string.maketrans(custom_table, standard_b64table))
    return base64.b64decode(s)

def decrypt(data):
    enc = custom_b64decode(data, daserf19_b64table)
    rc4 = ARC4.new(rc4key)
    dec = rc4.decrypt(enc[1:])
    out = lznt1.dCompressBuf(dec)
    return out

## -- End pasted text --

In [2]:
decrypt("GeB57VGaerHM0y8Wg73sZE^gpfmCpDPAYUBCIU32ahwKDKje/^BNlnav4u9hs^Gwo9GQq7L3XXs6AK
oHPvVF/DPSGJC8")
Out[2]: "Congratulations! Now you can decrypt Daserf's GET request!"
```

5. 設定情報の調査

• 格納先、暗号/エンコード方式

```
004188AB 968      call    CreateFileA
004188B0 94C      mov     [ebp+hFile], eax
004188B3 94C      cmp     [ebp+hFile], 0FFFFFFFFh
004188B7 94C      jz      loc_418B39
004188BD 94C      push   2                ; dwMoveMethod
004188BF 950      push   0                ; lpDistanceToMoveHigh
004188C1 954      push   0FFFFFFE16h     ; lDistanceToMove
004188C6 958      mov     eax, [ebp+hFile]
004188C9 958      push   eax              ; hFile
004188CA 95C      call   SetFilePointer
004188CF 94C      lea    eax, [ebp+Buffer] ; void *
004188D5 94C      mov     edx, 80h        ; unsigned int
004188DA 94C      call   Windows::ZeroMemory(void *,uint)
004188DF 94C      push   0                ; lpOverlapped
004188E1 950      lea    eax, [ebp+NumberOfBytesRead]
004188E4 950      push   eax              ; lpNumberOfBytesRead
004188E5 954      push   18h             ; nNumberOfBytesToRead
004188E7 958      lea    eax, [ebp+Buffer]
004188ED 958      push   eax              ; lpBuffer
004188EE 95C      mov     eax, [ebp+hFile]
```


5. 設定情報の調査

• データの読み込み 1

```
004188A6 954 push offset FileName ; lpFileName
004188A7 954 call CreateFileA
004188A8 954 mov [ebp+hObject], eax
004188A9 954 cmp [ebp+hObject], 0FFFFFFFFh
004188B7 93C jz loc_418B39
004188BD 93C push 2 ; dwMoveMethod
004188BF 940 push 0 ; lpDistanceToMoveHigh
004188C1 944 push 0FFFFFFE16h ; lDistanceToMove
004188C2 944 mov eax, [ebp+hObject]
004188C3 944 mov eax ; hFile
004188C4 944 call SetFilePointer
004188C5 944 mov eax, [ebp+String]
004188C6 944 mov edx, 80h
004188C7 944 call @ZeroMemory
004188DF 93C push 0 ; lpOverlapped
004188E1 940 lea eax, [ebp+NumberOfBytesRead]
004188E4 940 push eax ; lpNumberOfBytesRead
004188E5 940 push 18h ; nNumberOfBytesToRead
004188E6 940 lea eax, [ebp+String]
004188E7 940 push eax ; lpBuffer
004188E8 940 mov eax, [ebp+hObject]
004188E9 940 push eax ; hFile
004188F1 94C call ReadFile
004188F2 950
```

自身をオープン

FILE_END から
-490 バイト目に
ファイルポインタを移動

24バイト
読み込み

5. 設定情報の調査

• データの読み込み 2

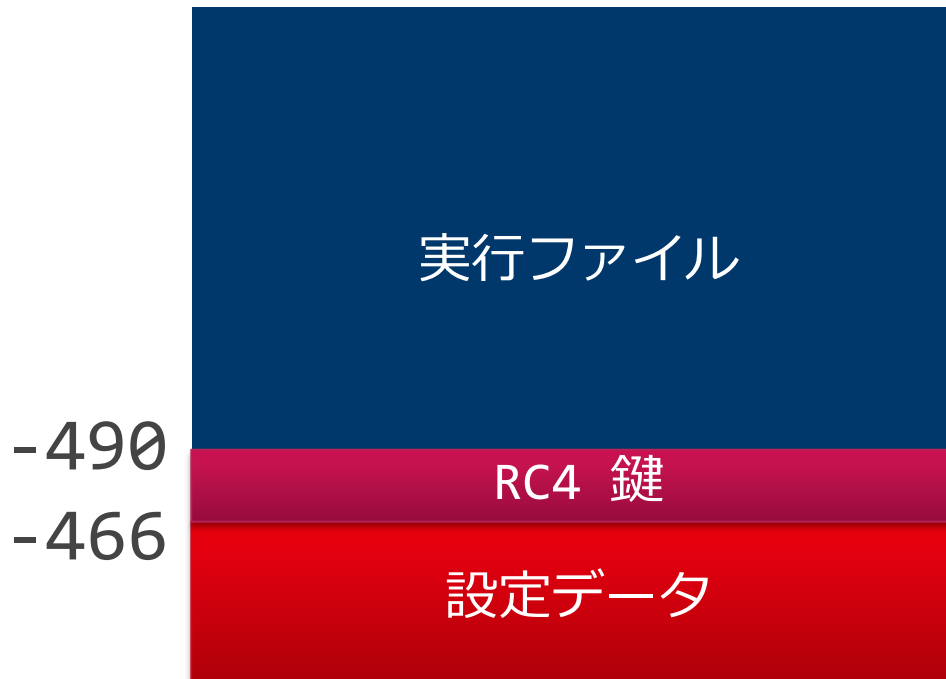
```
00418940 93C      push     0 ; lpOverlapped
00418942 93C      lea     eax, [ebp+NumberOfBytesRead]
00418944 93C      push     eax ; lpNumberOfBytesRead
00418946 93C      push     1D2h ; nNumberOfBytesToRead
00418948 93C      lea     eax, [ebp+Buffer]
0041894A 93C      push     eax ; lpBuffer
0041894C 93C      mov     eax, [ebp+hObject]
0041894E 93C      push     eax ; hFile
00418950 950      call    ReadFile
0041895B 93C      mov     eax, [ebp+var_C]
0041895E 93C      push     eax
0041895F 940      lea     edx, [ebp+var_92A]
00418965 940      lea     eax, [ebp+Buffer]
0041896B 940      mov     ecx, 1D2h
0041896D 940      mov     edi, sub_419888
```

-466バイトの位置から
466バイト読み込み

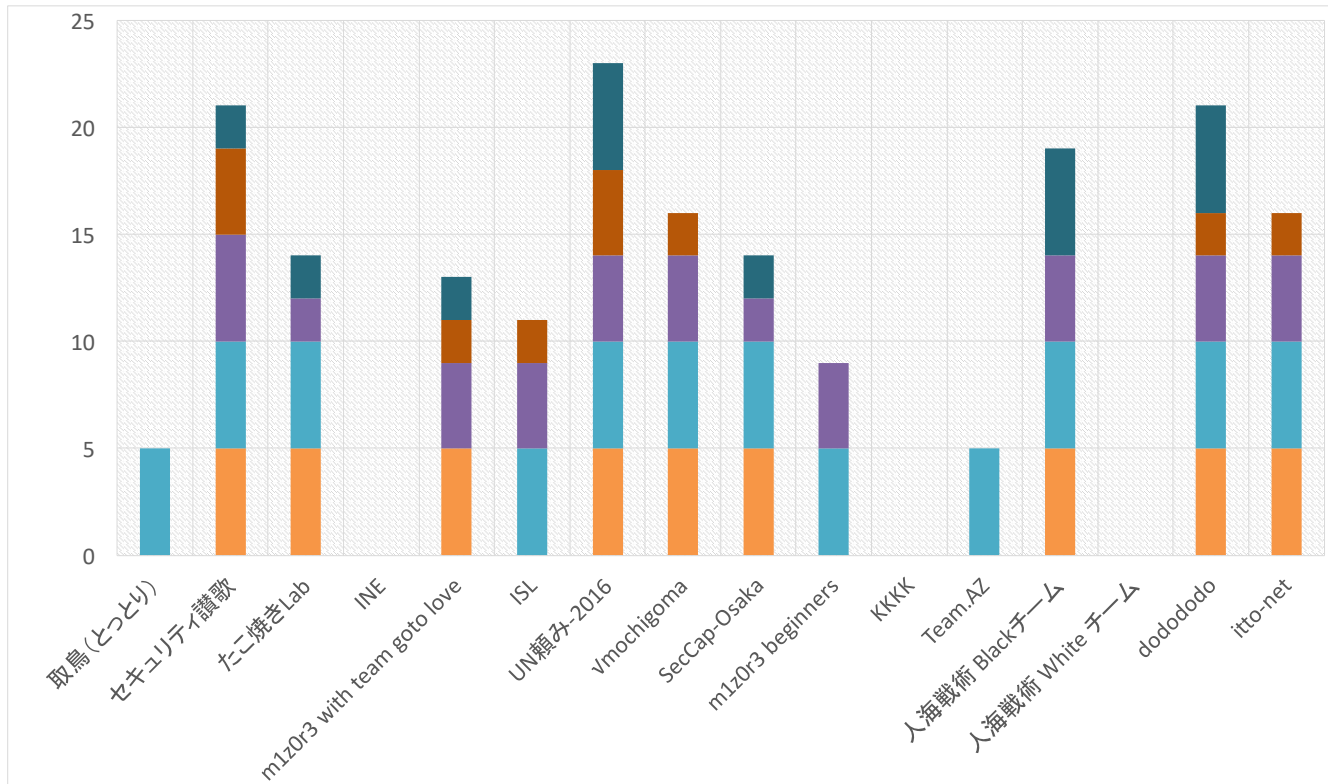
466バイトのデータを
24バイトの鍵で RC4

5. 設定情報の調査

- ファイル末尾に設定データを保持



採点結果





来年度に
向けて

自分たちの課題

0 か 1 か問題の解決

- ある程度解消できたがまだまだ

テーマ選定の難しさ

- 自動化などを静的解析で適用することの困難さ
- 現状のやり方では現場でリアルなマルウェアを見ている人が必須

作成委員不足

- みんな静的解析やろう！
- セキュリティ賛歌・UN頼みあたりが良さそう

みんなの課題

• 静的解析の知見を溜めよう

Web

- セキュリティ・キャンプ全国大会2015でのマルウェア分析講義 (2015-09-10)
<https://www.jpccert.or.jp/magazine/acreport-seccamp.html>
- リバースエンジニアリング入門 - @IT
<http://www.atmarkit.co.jp/ait/series/2614/>
- Edomae 2015 - マルウェアを解析してみよう
<http://www.slideshare.net/SatoshiMimura/edomae-2015>

書籍

- Practical Malware Analysis | No Starch Press
<https://www.nostarch.com/malware>
- O'Reilly Japan - アナライジング・マルウェア
<https://www.oreilly.co.jp/books/9784873114552/>

過去問

- ML に投げてるので、それを入手する
- データセットに入っている資料を読む



Thank you!

SecureWorks®
A Dell Company