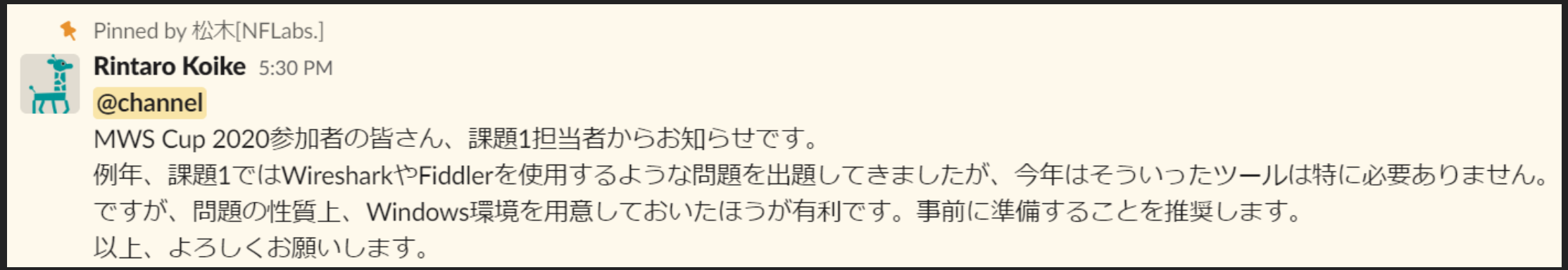
MWSCup 2020 課題1

nao_sec / NTTセキュリティ・ジャパン株式会社
小池 倫太郎

問題担当

- **主担当**
  - NTTセキュリティ・ジャパン株式会社　小池倫太郎
- **問題作成支援員**
  - 株式会社エヌ・エフ・ラボラトリーズ　保要隆明
- **レビュアー**
  - デロイトトーマツサイバー合同会社　高田雄太
  - nao_sec　メンバー一同

事前連絡



Pinned by 松木[NFLabs.]

Rintaro Koike 5:30 PM

@channel

MWS Cup 2020参加者の皆さん、課題1担当者からお知らせです。
例年、課題1ではWiresharkやFiddlerを使用するような問題を出題してきましたが、今年はそういったツールは特に必要ありません。
ですが、問題の性質上、Windows環境を用意しておいたほうが有利です。事前に準備することを推奨します。
以上、よろしくお願いします。

- **WiresharkやFiddlerは必要ない**
  - 例年のようにPCAPやSAZ形式での出題ではない
- **Windows環境の準備を推奨**
  - Windowsに依存するような何か？
  - 課題1と言えばDrive-by Download攻撃解析…

概要

- **難読化・解析妨害を含むJavaScriptコード解析**
  - スクリプトを用いた攻撃は依然として人気
    - JavaScript / JScript
    - VBScript / VBA
    - PowerShell

    -> スクリプトを解析する技能は重要

  - Augma Datasetから解析の練習になりそうなデータを抽出し模倣
    - 手動解析
    - 自動解析

    -> 実務では解析作業を手動で行うのは現実的ではない

概要

- **Exploit Kitのコードを模倣したJavaScriptの解析**
  - RIG Exploit Kit
  - Capesand Exploit Kit
  - PurpleFox Exploit Kit
  - Bottle Exploit Kit

  -> 様々な難読化・解析妨害を含んでおり、それを適切に対処しつつ、スクリプトの本質的な部分を解析していく

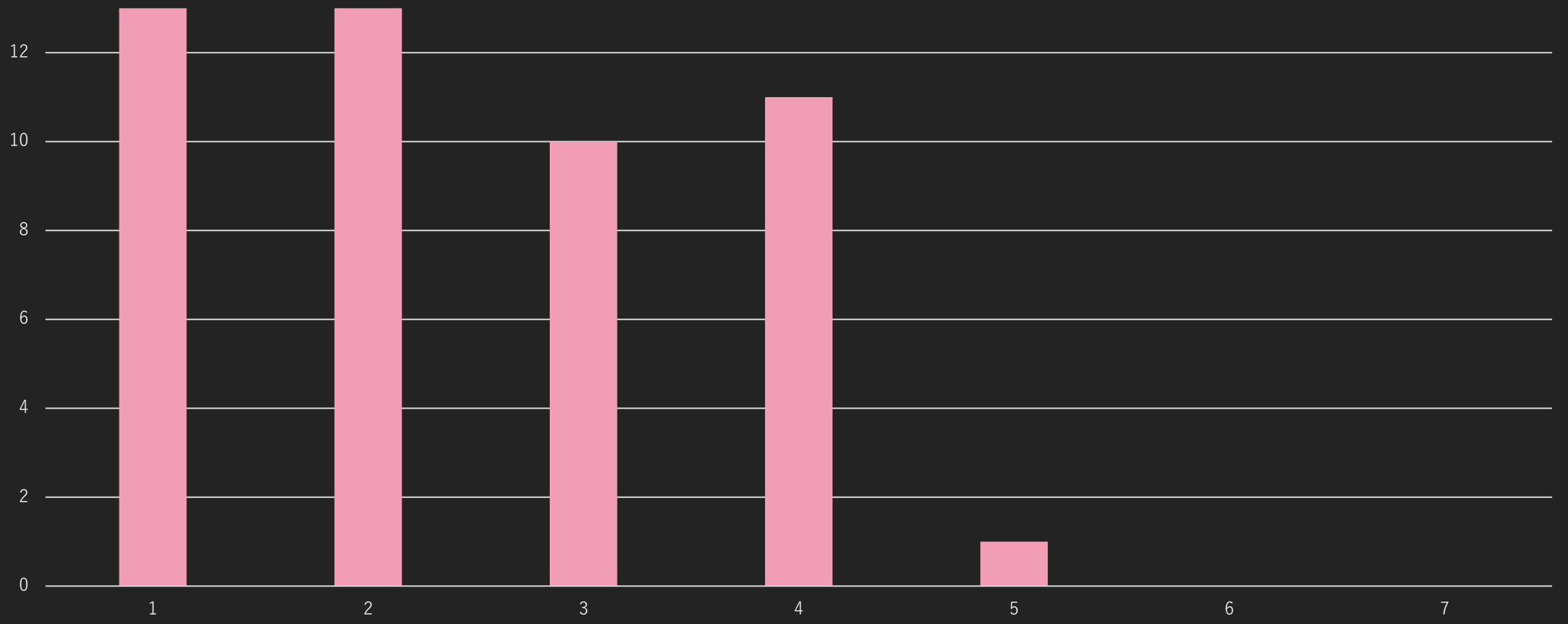
- アドバイス
  - 難読化・解析妨害は環境に依存させることが多い
  - Exploit Kitが動作する環境とは…？

競技中の解説はここまで

MWSCup 2020 課題1 解説

nao_sec / NTTセキュリティ・ジャパン株式会社
小池　倫太郎

解答について



Q1: BASE64

```
eval(atob
('aWYoJ0EnID09PSAnQicpIHsgdmFyIG13c2N1cF9xMV8xID0gIk1XU0N1cHtIZWxsbywgV29ybGQhfSI7IH0='
));
```

- **atob()**
  - Base64エンコーディングでエンコードされたデータの文字列をデコード
- **eval()**
  - 文字列として表現されたJavaScriptコードを評価
  - 多くの難読化・解析妨害では最終的に式・コードの評価を行い、動的に実行する
    - 最終的に実行される部分のみを抜き出せば解析を楽できる

Flag is MWS{Hello, World!}

Q2: SIMPLE

```
eval(function(p,a,c,k,e,d){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?
String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)){while(c--){d[e
(c)]=k[c]||e(c)}k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--)
{if(k[c]){p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}(atob
('MCgnMScgPT09ICcyJykgeyAzIDRfNV82ID0gJzd7OF85P30nOyB9'),62,10,'if|A|B|var|mwscup|q1|2|
MWSCup|SIMPLE|Obfuscation'.split('|'),0,{}))
```

- evalの中で色々な処理をしている
  - 最終的に実行されるコードのみを抽出すれば良いので、evalを書き換えるだけ
  - 単純にevalという文字列を削除するだけでいい

Flag is MWS{SIMPLE_Obfuscation?}

Q2: SIMPLE

```
(function (p, a, c, k, e, d) {
    // -- SNIP -- //
}(
    atob('MCgnMScgPT09ICcyJykgeyAzIDRfNV82ID0gJzd7OF85P30nOyB9'),
    62,
    10,
    'if|A|B|var|mwscup|q1|2|MWSCup|SIMPLE|Obfuscation'.split('|'),
    0,
    {}
))
```

- evalの中身
  - Base64文字列をデコードすると "0('1' === '2') { 3 4_5_6 = '7{8_9?}'; }"
  - 数字部分に第4引数の文字列配列を入れ込んでいく
    - 0 -> if, 1 -> A, 2 -> B, 3 -> var, 4 -> mwscup…

Q3: RIG Exploit Kit

```
IHtXbCTl = "ev\x61" + "l";
X8ntM_TH1bx = this[IHtXbCTl];
X8ntM_TH1bx(oQpj47sSzPL8sW7h);
```

- 典型的な文字列操作 -> evalによる動的実行
  - evalによる実行は基本的に最後に行われる
  - "eval"という文字列をいかに生成するか？
    - 今回は単純なエスケープと分割

Flag is MWS{rig_Rig_RIG!!!}

Q3: RIG Exploit Kit

- evalに至るまで
  - 2段階の文字列操作
    1. 2つの文字列データを順番に連結

```
ah0nDmZh8uO = " ;□!G!igrigSCu□3qscurm{□vB□□□f□□"
["spl" + "i" + "\x74"]("");
eGxX = "iA□□□□a□w□p_1_□□□MWp{_R_RI!}□}" ["sp" + "li" + "t"]
("");
for (oQpj47sSzPL8sW7h = '', IM8e = 15, fhs_ = 0; IM8e > -1, fhs_ <= 15; IM8e--, fhs_++)
 {
    oQpj47sSzPL8sW7h += eGxX[fhs_];
    if (typeof ah0nDmZh8uO[IM8e] != "u\x6e" + "d" + "e\x66" + "in" + "e" + "d") {
        oQpj47sSzPL8sW7h += ah0nDmZh8uO[IM8e];
    };
}
```

Q3: RIG Exploit Kit

- evalに至るまで
  - 2段階の文字列操作
    2. 合成された文字列データに含まれる文字を置換

```
otv5Tg = "□.□<□>□=□\"□\'□)□(□ □\t□\n□\r□\\";
for (Ybvw6J84WY = 0; Ybvw6J84WY <= otv5Tg.length - 1; Ybvw6J84WY++) {
    oQpj47sSzPL8sW7h = oQpj47sSzPL8sW7h["rep" + "l" + "ace"](new RegExp(otv5Tg["s" +
    "ub" + "\x73t" + "r"](Ybvw6J84WY, 1), "g"), otv5Tg["sub" + "st" + "\x72"]
    (Ybvw6J84WY + 2 - 1, 1));
    Ybvw6J84WY++;
}
```

```
000000B0  01 2E 02 3C 03 3E 04 3D 05 5C 22 06 5C 27 07 29    ...<.>.=.¥".¥'.)
000000C0  08 28 0F 20 10 5C 74 11 5C 6E 12 5C 72 13 5C 5C    .(. .¥t.¥n.¥r.¥¥
```

Q4: Capesand Exploit Kit

- evalを見つけるだけ

```
this["e" + "va" + "l"](sWUStA5cFe1);
```

- evalに至るまで
  - 単純な文字列操作

```
FXrtHm.forEach(function nIn(value) {
    sWUStA5cFe1 += String.fromCharCode(parseInt(atob(value).replace(/\D/g, '')) -
    42350068);
});
```

Flag is MWS{Cape_sanD_CAPE_SAND}

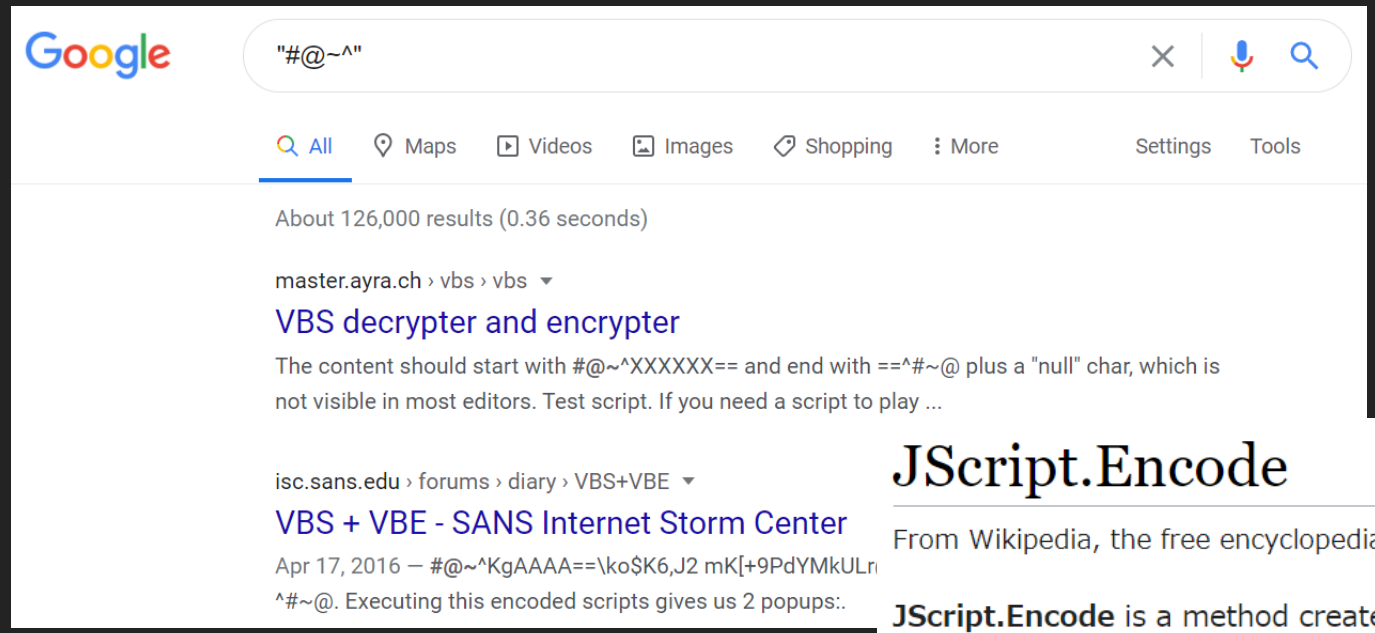
Q5: PurpleFox Exploit Kit

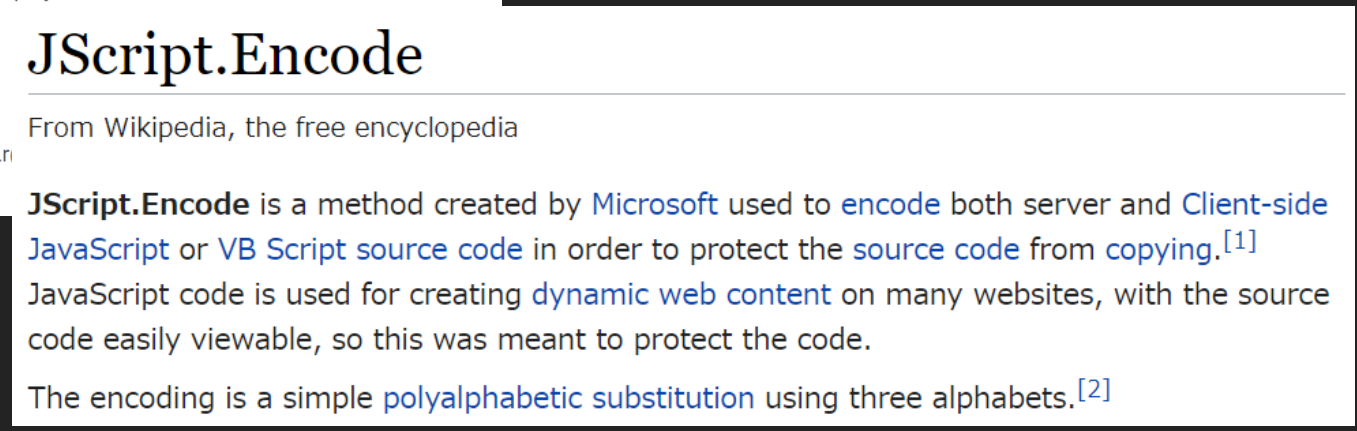
```
#@~^8MMAAA==Z6EU1YbWxvYBn#PJG4N+^Or''Dzw W0,nawKDD/_:G[!VnR awKDOd{+62GMYd' c*)E6E
mYbW    E'{Yzw WW~9+0bU+L[N Wrx Rm:[g[n6kU+v,TB+blDRZ.zaYG9Ux `b)`Dtkk~6;x1YrW  `b`7lD,
4~D~+B.SkBxB0SWSdBmSlBs~9~hSX~4SuBySbB;BwS|~7~XBoB$~S~V~U~/SG~2B]~t~sBKS BrBqSjSFB
(SSB%~g~PS$~}S#BMS9ByBpSI~DY~ YB.YBkO~  YSGD~/DSmD~lDS4YBVD~WYS[D~;YB2YB
{OS7Y~zOBoO~~OBhOB3D~?D~(O'(Yku6EU^DkW  cV* \m.~Ypk6`EEU[ 0rx [Je'Oza+WW~SkUNKAL
[Abx9WhcmMzwDW'[vYxAbxNKAR1DXaOG#B"D['J;U9+Wk   nNr"xOHw+GW,hrx9GS['Sk  NWSRsdZMX2YK
['cD'hbUNKhRsd/DHwDWb~ZOL[EE    [+6kUn9J"xOHwnW6~TVG(l^[[TVK8l^R^DHwOGL[`Dxo^W4ms
mMXaYG#SZD['J6;x1YrG    J'xOHwnW6~M+5!kM+#DDH`Y{Dn;!k.nvJmMzwDWJ*N^lDm4`O#`N6EUmDrW
Prc*    kWcD#`k6cr0;    mDkW    J{xYHwnW6PO T+Y"Cx9W:jCsE /*Y.X`. Y;D   ~YconO"lx
```

- JavaScriptではないナニカ？
  - 先頭が#@~^、末尾が^#~@という特徴

Q5: PurpleFox Exploit Kit

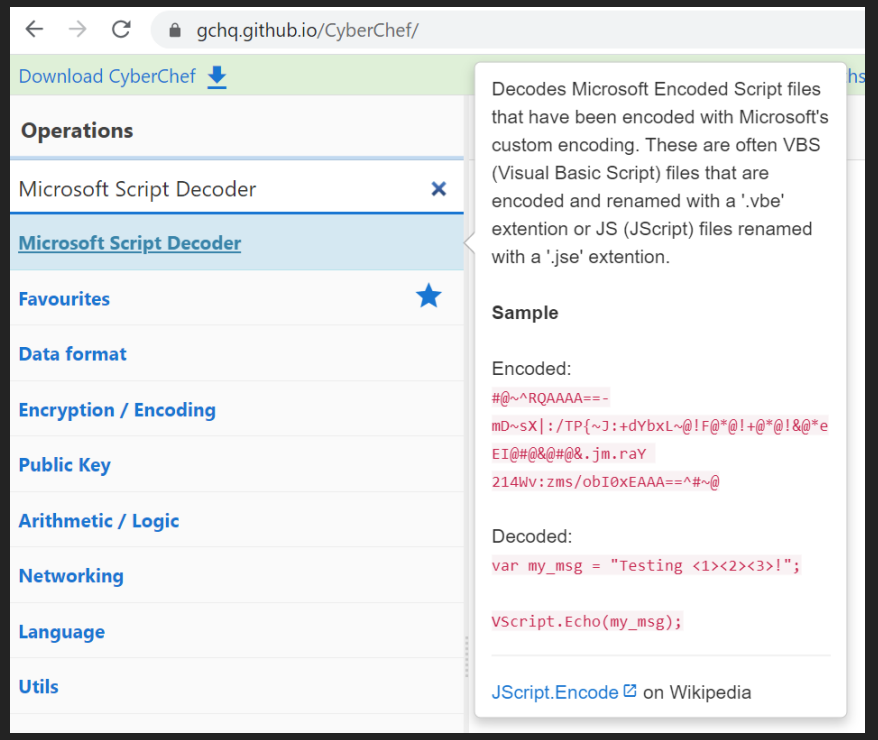
- 素直に検索してみる





Q5: PurpleFox Exploit Kit

- **Microsoft Script Encoder**
  - <script language="JScript.Encode">
  - 古のテクノロジーで、Internet Explorerなどで動作
  - 単純なエンコードで、昔からデコーダがいくつも存在
    - scrdec18-VC8.exe
      - https://gist.github.com/bcse/1834878
    - CyberChef
      - https://gchq.github.io/CyberChef/



Q5: PurpleFox Exploit Kit

- デコード後のデータ
  - 先頭に大きなライブラリみたいなコード
    - 暗号系のライブラリ？
  - とりあえず無視して先を見る

```
!function(t,e){"object"==typeof exports?module.exports=exports=e():"function"==typeof
define&&define.amd?define([],e):t.CryptoJS=e()}(this,function(){var h,t,e,r,i,n,f,o,s,
c,a,l,d,m,x,b,H,z,A,u,p,_,v,y,g,B,w,k,S,C,D,E,R,M,F,P,W,O,I,U,K,X,L,j,N,T,q,Z,V,G,J,$,
Q,Y,tt,et,rt,it,nt,ot,st,ct,at,ht,lt,ft,dt,ut,pt,_t,vt,yt,gt,Bt,wt,kt,St,bt=bt||
function(l){var t;if("undefined"!=typeof window&&window.crypto&&(t=window.crypto),!t&&
"undefined"!=typeof window&&window.msCrypto&&(t=window.msCrypto),!t&&
"undefined"!=typeof global&&global.crypto&&(t=global.crypto),!t&&"function"==typeof
require)try{t=require("crypto")}catch(t){}function i(){if(t){if("function"==typeof
```

Q5: PurpleFox Exploit Kit

- デコード後のデータ
  - evalを発見 -> 実行されるコードを取得

```
this["eva" + "l"](eEePTp4eXxk1(C44q2jxHQW));
```

  - もしInternet Explorer以外のブラウザを使用していた場合
    - デコードに失敗
    - なぜ失敗したのか？

```
> eEePTp4eXxk1(C44q2jxHQW)
<· ""
```

Q5: PurpleFox Exploit Kit

- 細かくコードを見ていく
  - evalの直前で実行されていた関数
    - CryptoJSを使ったAES-256 CBCによるデコード
    - つまり先頭にあった大きなライブラリはCryptoJS
    - 必要なもの
      - 暗号化されたデータ
      - 暗号鍵
      - IV

```
function eEePTp4eXxk1(b) {
    var a = CryptoJS.AES.decrypt(b, CryptoJS.enc.Utf8.parse(key), {
        iv: CryptoJS.enc.Utf8.parse(iv),
        mode: CryptoJS.mode.CBC,
        padding: CryptoJS.pad.Pkcs7
    });
    return a.toString(CryptoJS.enc.Utf8)
};
```

Q5: PurpleFox Exploit Kit

- 細かくコードを見ていく
  - 暗号鍵とIVはすぐに見つかる
  - 暗号化されたデータ
    - Base64デコードしているだけかと思いきや、何か処理をしている

```
var iv = "CaMJPtNxY16xA4V4";
var key = "4d364bd04069c6e8";
```

```
var C44q2jxHQW = 'Wh1UPzACCV01BTZRF1IePwwcQys
+XF9QBVYVXioTNywQXlQHAloQEjQKGEoULFYlNigQAFVVDQQzKxUxHCIDFxdWChAELFQqPC1PVi0aBQIhIx8gEy
dUUw==';
C44q2jxHQW = atob(C44q2jxHQW);
var key = typeof ([].find);
var V9DVL = "";
for (var i = 0; i < C44q2jxHQW.length; i++) {
    V9DVL += String.fromCharCode(C44q2jxHQW.charCodeAt(i) ^ key.charCodeAt(i %
    key.length));
}
C44q2jxHQW = V9DVL;
```

Q5: PurpleFox Exploit Kit

- 細かくコードを見ていく
  - 暗号化されたデータに対する処理
    - typeofの結果をもとに、XORで更にデータを加工
    - typeof([].find)とは？

```
var key = typeof ([].find);
```

```
> typeof([].find)
<· "function"
```

Chrome

```
typeof([].find)
"undefined"
```

Internet Explorer

- 古いJavaScriptエンジンにはArray.findは実装されていない
- （思い出しポイント）最初のJScript.EncodeはInternet Explorerなどで実行可能
  - Chromeでデコードに失敗 -> Internet Explorerで実行するのが正しい

Flag is MWS{Purp1e_F0X_4K}

Q5: PurpleFox Exploit Kit

- ブラウザや環境に依存する値を使った解析妨害
  - 標的を絞り込んだ攻撃ではよく見られる
    - いわゆる標的型攻撃以外にも、Region-Specificなバラマキ攻撃でも
  - よくあるテクニック
    - 実装差異
      - Internet Explorerと他のブラウザの差
    - 環境値
      - ブラウザやOSの言語設定
      - IPアドレスのGeolocation
      - エラーメッセージ
        - コマンドプロンプトなどで意図的にエラーを起こし、返ってくるエラーメッセージを使って暗号鍵を生成するなど

Q6: Bottle Exploit Kit

- evalっぽい処理を探してみるが、それっぽいものは見つからない
  - もしここで見つけられたとしても、解析妨害が邪魔をする可能性が高い
    - 適切な環境で、適切にeval処理を対処できればflagはすぐに手に入る

- 先頭から雑に実行していく
  - 適切な環境以外で実行すると、無限ループに突入しクラッシュ
    - 適切な環境とは？
  - 解析妨害処理が複数含まれていると思われる
  - 当たりをつけて、必要な箇所だけ実行していく必要がある

Q6: Bottle Exploit Kit

- ざっとコードを眺めて、重要そうな部分のみを読み解いていく
  - 先頭に重要そうなデータが定義されている

```
document["cookie"] = "VCB6bwLARvUdqy9e";
var _0x069034 =
"PyVqESNQbHxvS3VDM14QRRFBWysidi41RQECPB4jRwpSUUQZQic2fgklJjgAWRYDICQNJjkFHxxmLzocCnFwKw
==";
```

  - document.cookieと_0x069034に着目して処理を追っていく
    - document.cookie
      - 直後の即時関数内でcookieという文字列がいくつか存在 -> これは罠
    - _0x069034
      - switch文の中でデータ処理らしき操作をしている

Q6: Bottle Exploit Kit

```
var _0x57a0b5 = "0|4|5|3|2|1" [_0x2b13(61, "Tnr_")]('|'),
    _0x49c44b = 0x0;
while (!![]) {
    switch (_0x57a0b5[_0x49c44b++]) {
        case '0':
            var _0x043756 = ![];
            continue;
        case '1':
            for (i = 0; i < _0x069034.length; i++) _0x052845 += String
            ["fromCharCode"](_0x069034["charCodeAt"](i) ^ _0x064183["charCodeAt"]
            (i % _0x064183.length));
            continue;
        case '2':
            _0x069034 = atob(_0x069034);
            continue;
        case '3':
            var _0x052845 = "";
            continue;
        case '4':
            if (navigator["vendor"]["length"] > 0) _0x043756 = !_0x043756;
            continue;
        case '5':
            if (_0x043756) _0x064183 += navigator["vendor"];
            continue;
    }
    break;
}
```

Control Flow Flattening

Q6: Bottle Exploit Kit

- ざっとコードを眺めて、重要そうな部分のみを読み解いていく
  - _0x069034に対する操作
    - switch文の中でデータ処理らしき操作をしている
      - なんとなくは理解できるが、一部の情報はエンコードされてしまっている

      -> _0x2b13という関数の中で処理

  - _0x2b13()
    - _0x5e6f4bと_0x52b995の2つの引数を受け付ける
    - 序盤で定義されている大きな文字列配列_0x1833をデコード
    - その前に即時関数でなんらかの操作が行われている

    -> 即時関数と_0x2b13の処理を正確に追いかける必要がある

Q6: Bottle Exploit Kit

- ざっとコードを眺めて、重要そうな部分のみを読み解いていく
  - 即時関数
    - 大量のノイズコードが存在するが、やっていることは極めてシンプル
    - 文字列配列_0x1833をshiftしてpushする

    -> つまり順番を入れ替えているだけ

  - 0x2b13
    - 即時関数と同様に大量のノイズコードを削除
    - 第1引数で渡される_0x5e6f4bをindexとして文字列配列_0x1833のデータを取得
    - エンコード処理
      - Base64 Decode
      - URL Decode
      - RC4 Calc
        - 第2引数の_0x52b995が鍵データとなる

Q6: Bottle Exploit Kit

- ざっとコードを眺めて、重要そうな部分のみを読み解いていく
  - 改めて_0x069034に対する操作を見る
    - Control Flow Flattening
      - 制御フローを平坦化して、読みにくくする難読化手法
      - https://github.com/obfuscator-llvm/obfuscator/wiki/Control-Flow-Flattening
    - 単純化すると

```
var _0x043756 = ![];
if (navigator["vendor"]["length"] > 0) _0x043756 = !_0x043756;
if (_0x043756) _0x064183 += navigator["vendor"];
var _0x052845 = "";
_0x069034 = atob(_0x069034);
for (i = 0; i < _0x069034.length; i++) _0x052845 += String["fromCharCode"](_0x069034
["charCodeAt"](i) ^ _0x064183["charCodeAt"](i % _0x064183.length));
```

Q6: Bottle Exploit Kit

- ざっとコードを眺めて、重要そうな部分のみを読み解いていく
  - 改めて_0x069034に対する操作を見る
    - _0x069034をBase64デコードし、それに対してXOR
      - 鍵となっているデータは_0x064183

  - _0x064183に対する操作を見る
    - switch文の直前でdocument.cookieの値を代入
    - switch文の中ではnavigator.vendorの値を使っている
      - navigator.vendorのlengthが0以上の場合、それを_0x064183に連結
      - navigator.vendorとは？
        - ブラウザベンダー情報
        - ブラウザ依存のデータで、Internet Explorerの場合は空文字列

Q6: Bottle Exploit Kit

- ざっとコードを眺めて、重要そうな部分のみを読み解いていく
  - document.cookieに対する操作を見る
    - cookieという文字列で見ると解析妨害に巻き込まれる（前述）
    - documentという文字列で探していく

```
var _0x5ada84 = _0x7e8435[_0x2b13(45, "Yp6S")][_0x2b13(61, "Tnr_")]("|"),
    _0x432b59 = 0x0;
var _0x064183 = document[_0x2b13(62, "nh98")];
while (!![]) {
    switch (_0x5ada84[_0x432b59++]) {
        case '0':
            if (typeof (navigator["systemLanguage"]) !== "undefined" && navigator
            ["systemLanguage"]["indexOf"]("ja") > -1) _0x048529 = !_0x048529;
            continue;
        case '1':
            if (_0x048529) _0x064183 += navigator["systemLanguage"];
            continue;
        case '2':
            var _0x048529 = ![];
            continue;
    }
    break;
}
document[_0x2b13(62, "nh98")] = _0x064183;
```

Q6: Bottle Exploit Kit

- ざっとコードを眺めて、重要そうな部分のみを読み解いていく
  - document.cookieに対する操作を見る
    - 処理順としては今回のswitch文のほうが先
    - 単純化すると

```
var _0x064183 = document["cookie"];
var _0x048529 = ![];
if (_0x048529) _0x064183 += navigator["systemLanguage"];
if (typeof (navigator["systemLanguage"]) !== "undefined" && navigator["systemLanguage"]
["indexOf"]("ja") > -1) _0x048529 = !_0x048529;
document["cookie"] = _0x064183;
```

    - navigator.systemLanguageとは？
      - その名のとおり、システム言語を返す
      - Internet Explorerには実装されているが、Chromeなどには存在しない
        - Internet Explorerの場合 -> "ja-JP"

Q6: Bottle Exploit Kit

- ざっとコードを眺めて、重要そうな部分のみを読み解いていく
  - 最終的な鍵データ
    - 最初にdocument.cookieに代入された文字列 + "ja-JP"
      - 日本語のInternet Explorer以外で実行すると、異なる文字列となってしまう
  - 得られた鍵データを使って、データをデコードすることでflagが得られる
    - これら以外にも大量の解析妨害が含まれている
    - 詳細はこちら
      - https://nao-sec.org/2019/12/say-hello-to-bottle-exploit-kit.html

Flag is MWS{Bottle_Plane_JP}

Q7： 解析の自動化

- Q1 ~ 6までの解析作業を自動化して欲しい
  - 実務では毎回手動で解析することは現実的ではない
  - 5分間で全てをデコードすればいいので、半自動 / 手動でもOK

- アプローチ
  - ブラウザやエミュレータを使ったJavaScript解析
    - Headless ChromeやSelenium / WebDriver
    - JS-Walkerのようなツールの活用
      - https://www.nttsecurity.com/docs/librariesprovider3/resources/jswalker
  - 気合いで文字列操作

Flag is **MWS{Enj0y_EK_0bfuscat0r}**

まとめ

- **難読化・解析妨害を含むJavaScriptコード解析**
  - 単純な難読化の他に、様々な環境依存値や解析妨害
  - 要点を見極め、必要な部分だけを効率的に解析
- **解析の自動化**
  - 実務では全てを手動で解析することは現実的ではない
  - 自動化できることは、なるべく自動化する
- **JavaScriptに限らず、他の言語でも解析経験は活用できる**
  - 悪性ドキュメントファイルや標的型攻撃で使用されるPowerShellなど

Any Questions?

@nao_sec
info@nao-sec.org