MWS Cup 課題1

MWS Cup 2020 課題1 主担当
小池 倫太郎

2020年の問題担当

- 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

2021年の問題担当

- 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
  - 募集中！！！

課題1の問題傾向

- **Augma/D3M Datasetに関連する問題**
  - いわゆるDrive-by Download攻撃に関連するもの
  - データセットの利活用促進が目的

  - Drive-by Download攻撃の観測
    - （事前課題）2015年，2016年

  - トラフィックデータの解析
    - 2014年，2017年，2019年

  - 悪性スクリプトの解析
    - 2020年

2020年の問題

出題意図

- **リモート+CTF形式で、悪性スクリプトを解析**
  - スッパリと解答（flag）が何であるか分かるような形式
    - 記述や選択肢ではなく、問題に埋め込まれたflagのみを答える
    - 従来のpcapやsazのように「何か（ツール）が無いと解きにくい」ような問題ではなく、純粋に解析力で戦えるような問題
  - DbD解析だけではなく、汎用的に使える解析ノウハウを体験して欲しい
    - 幅広い難易度・テクニック
      - 直近1年間程度の間で実際に観測された攻撃を参考にして作成
      - 様々な場面で応用できるような汎用性
    - 単純な知識ではなく、如何にして手を動かすか？
      - 最終的には「自動化するにはどうしたらいいのか？」
  - （あわよくば `Augma Dataset` に興味を持ってもらいたい）

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?}

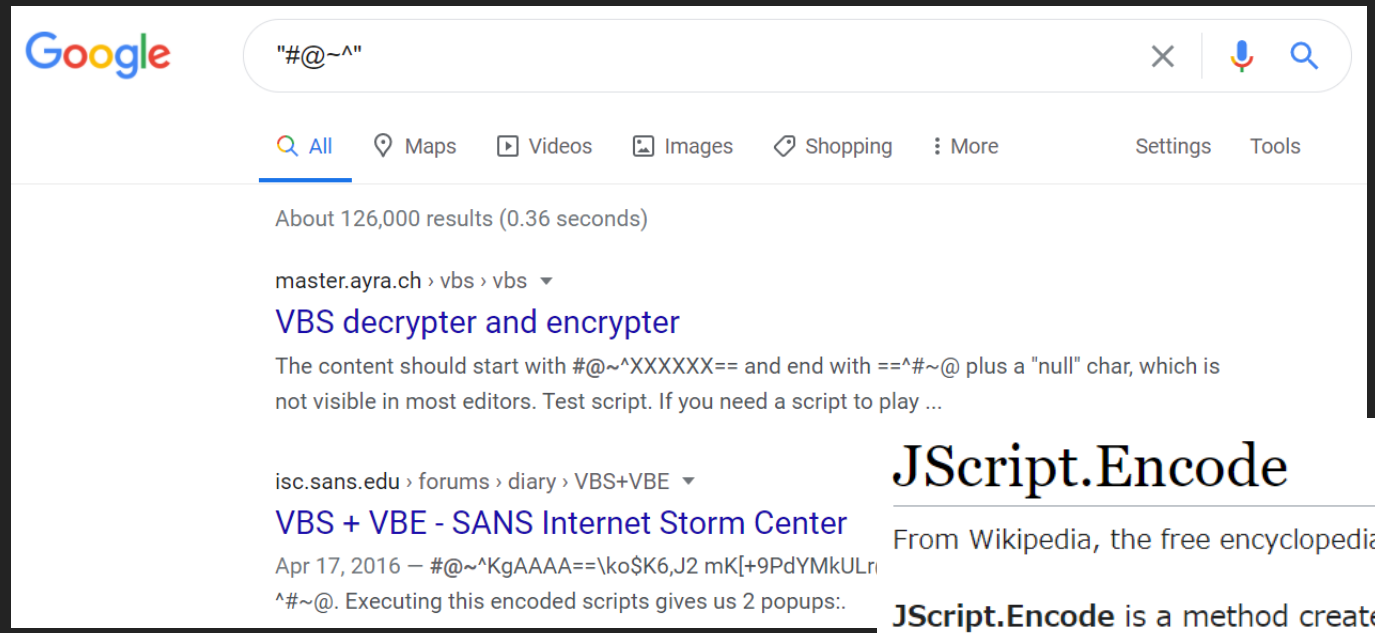
Q5: PurpleFox Exploit Kit

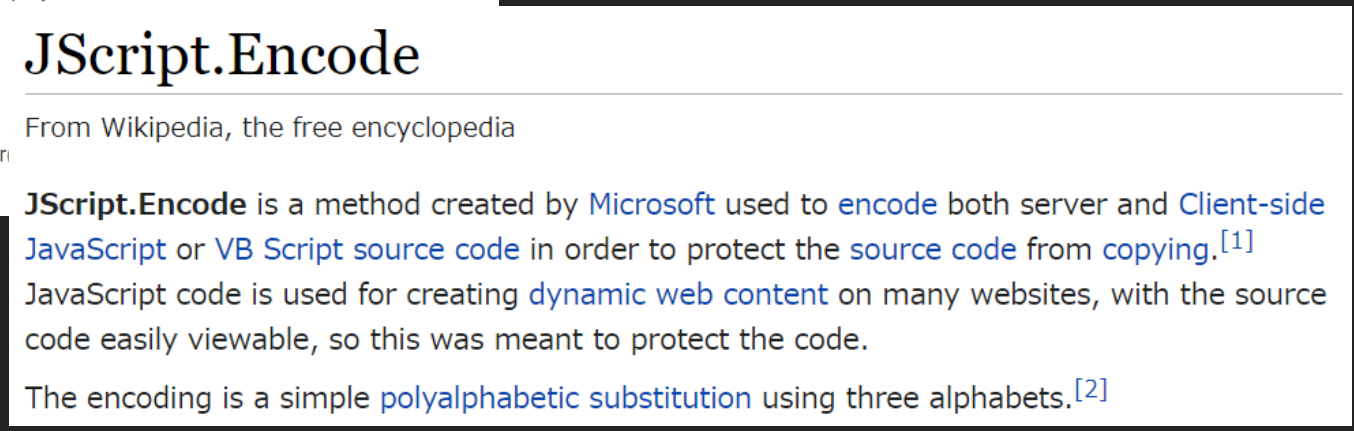
```
#@~^8MMAAA==Z6EU1YbWxvYBn#PJG4N+^Or''Dzw W0,nawKDD/_:G[!VnR awKDOd{+62GMYd' c*)E6E
mYbW    E'{Yzw WW~9+0bU+L[N Wrx Rm:[g[n6kU+v,TB+blDRZ.zaYG9Ux `b)`Dtkk~6;x1YrW  `b`7lD,
4~D~+B.SkBxB0SWSdBmSlBs~9~hSX~4SuBySbB;BwS|~7~XBoB$~S~V~U~/SG~2B]~t~sBKS BrBqSjSFB
(SSB%~g~PS$~}S#BMS9ByBpSI~DY~ YB.YBkO~  YSGD~/DSmD~lDS4YBVD~WYS[D~;YB2YB
{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

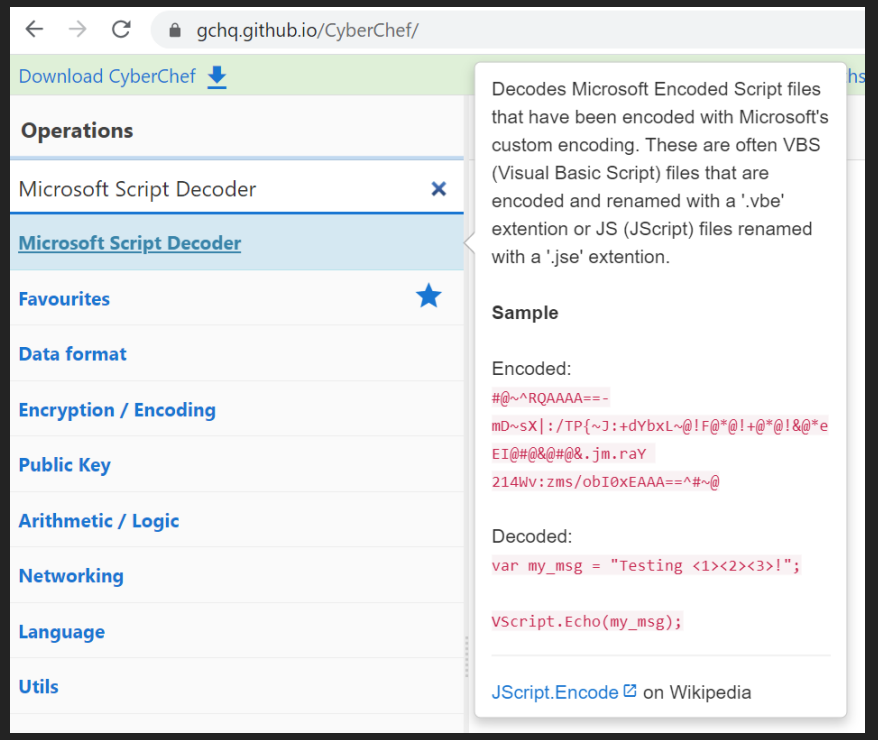
- 素直に検索してみる





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

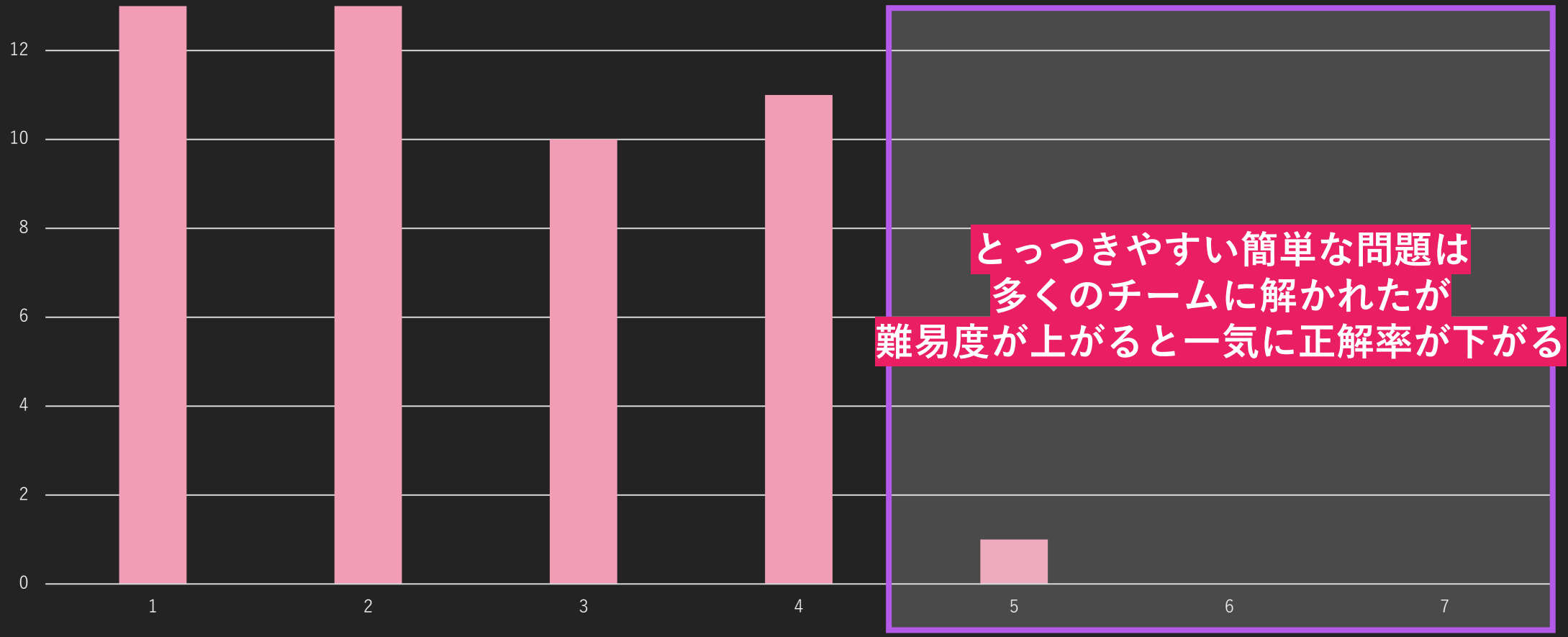
Q7： 解析の自動化

- **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}

結果


12
10
8
6
4
2
0
1
2
3
4
5
6
7
とっつきやすい簡単な問題は
多くのチームに解かれたが
難易度が上がると一気に正解率が下がる

勉強方法

勉強方法

- まずデータセットの中身をちゃんと把握する
  - MWS Cupはデータセット利活用促進が目的
- 過去問を解いてみる
  - データセットに過去問が含まれている
  - 解答や解説はWebにも
    - https://www.iwsec.org/mws/mwscup.html
- 一般に公開されているデータを活用する
  - https://www.malware-traffic-analysis.net
  - https://traffic.moe <-古いので要注意

勉強方法

- 解析レポートを読みながら、実際に解析する
  - “Exploit Kit” + “Analysis” とかで検索
    - 1次ソースを見る
- 最新動向を追う
  - Twitterで有益な情報を発信しているアカウントをフォロー
    - 日本語で発信している人は極めて少ない
    - 英語でも積極的に見よう！
  - あわよくば仲良くなって、分からないことを根掘り葉掘り聞く
- 何かあれば @nao_sec へご相談ください