NEXT-GENERATION OF EXPLOIT KIT DETECTION BY BUILDING SIMULATED OBFUSCATORS

Tongbo Luo & Xing Jin
Palo Alto Networks
Agenda

- Background and Motivation
- Challenges
- Our Approach and Lessons We learned
- Result and Our Observation
Motivation

www.jsDarwin.com

Similarity-based Detection on malicious Exploit Kit JavaScript

Large Number of Samples Required
Exploit Kit Obfuscator

Malicious Payload

Encoder

Unpacker Template

EVAL Template

Obfuscation Engine

Encoded Payload

Unpacker

EVAL Trigger

Obfuscated Page

Reconstruct Encoder

Identify Template From Obfuscated Page

Reproduce Engine Logic
Obfuscator Reverse-Engineering

window ['e'+v+'a'+']
  [wsq(xyz, fxj)];

window ['ev'+al']
  [zxh(abc, sih)];

window ['aevala'.substr(1,4)]
  [zxh(abc, sih)];


EVAL Template
(version)

Obfuscation Engine
(variant)
Challenges

1. Code Complexity

Hundred lines of code
Random variables

```html
<script>
function pDA(AUT){
  Zzf = "lnFloa";
  function vzLh(Hpv){
    gEao = "oi" + "n", aOC = window, SLVmg = "e" + "x",
    bUdR = "R" + "e" + "g", FMwVS = "o" + "u" + "te" + "rH",
    HjEn = "u" + "s" + "r", Wyc = "b" + "o", Wbf = "e" + "R" + "x",
    qVa = new aOC[bUdR + 'Exp']('MSIE (\d+\.\d+);'),
    xngS = navigator[HjEn + 'Agent'],
    prC = qVa[(Wbf + 'ec').replace('R', '')](xngS);
    var $mrh = prC, ryG; $mrh = !$mrh;
    if ($mrh) {
      ryG = prC[1];
      GcSX = '\*' + '(' + '*' + ')') + '<\p';
    }
    else {
      VdZW = ['r' + 'e' + 'p' + 'l' + 'a' + 'c' + 'e'] + 'gEao();
    }
  }
  var UkP = document[Wyc + 'dy'],
  yh$f = UkP[FMwVS + 'TML'],
  wvnV = '',
  El = new aOC[bUdR + 'Exp'](Zzf + Hpv + GcSX, 'gi');
  wvnV = El[SLVmg + 'ec']((yh$f);
  if (pDA(GKcx(WMK))) alert("Error");
  vzLh("dNTvXuVoyq");
  vzLh("xWgaeJKYcclAu");
</script>
```
Challenges

2. Data Complexity

Big data set (~20000 Samples over 2 year period)
Mixed versions and variants
Overview of Our Approach

1. JavaScript Normalization
2. Hierarchical Clustering
3. Reproduce Obfuscator
JavaScript Normalization

- Ignore Superficial Obfuscation (e.g. Randomized variable names)
- Normalized Script $\Leftrightarrow$ Structure of the Code
**Statistics**

<table>
<thead>
<tr>
<th>Family</th>
<th>Angler</th>
<th>Nuclear</th>
<th>Rig</th>
<th>KaiXin</th>
<th>Fiesta</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL #</td>
<td>7834</td>
<td>1303</td>
<td>1793</td>
<td>10291</td>
<td>79</td>
</tr>
<tr>
<td>UNIQUE #</td>
<td>613</td>
<td>107</td>
<td>344</td>
<td>1543</td>
<td>48</td>
</tr>
</tbody>
</table>

**Total Number of samples we collected**

- **Function** `azy(Ag6) { return Ag6; }`
- **Function** `Fx3j(k2c5x) { return k2c5x; }`

**Function** `FI(1) { RI; }`
Statistics on Normalized Samples

Angler Exploit Kit

- Same Day: 36%
- 1 Day Apart: 24%
- 2 Days Apart: 17%
- 3 Days Apart: 8%
- 4 Days Apart: 5%
- 5 Days Apart: 4%
- 6 Days Apart: 3%
- 7 Days Apart: 2%
- 8+ Days Apart: 1%
Clustering

- Goal: Cluster Samples based on their obfuscator
- Observation: Similar Structure ➔ Generated by Similar Obfuscator
- Define Similarity.

\[
SimScore(norm1, norm2) = 1 - \frac{EditDist(norm1, norm2)}{\max(len(norm1), len(norm2))}
\]

- Similarity Score [ 0 , 1 ]
- 1 = Identical, 0 = Different
Hierarchical vs Flat

- Flat Model: (K-Means)
  - Easy and Efficient
  - Drawbacks

- Require predefined $K$ as input
  - $K \iff$ Number of Obfuscator Version
  - Hard to Predict, lack of knowledge
  - Will be Changed over time.
Agglomerative (Bottom-Up) Hierarchical Clustering

What is the proper Threshold to Identify Obfuscator Version/Variant?

[0.4 – 0.5] Identify Obfuscator Version
[0.78 ~ 0.85] Identify Obfuscator Variants
Clustering Result

1. JavaScript Normalization

2. Obfuscator Version

- Nuclear Variant 1
  - 1.1
  - 1.2
  - 1.3

- Nuclear Variant 2
  - 2.1
  - 2.2
  - 2.3
Reproduce Obfuscator for Each Cluster

1.1

window ['e'+ 'v'+ 'a'+ 'l'][wsq(xyz, fxj)];

window ['ev'+ 'al'][zxh(abc, sih)];

window [l28f + i8d4][zxh(abc, sih)];

\[I[‘S’+‘S’+‘S’+‘S’][l(l,l)];\]

\[I[‘S’+‘S’][l(l,l)];\]

\[I[l +l][l(l,l)];\]

Threshold 0.80 ➔ 80% Similar Within Cluster

Obfuscator Engine

Template
Why This Research

- Boost Samples Set
- Improve Detection Rate
- Better Understanding on Obfuscator
During December 2014, a new version of Nuclear Pack emerged. ... new version will completely replace the old version.

Life Cycle of **Angler Exploit Kit Obfuscator**

- Extremely Prevalent
- Aggressive tactics for evading detection
- Only few Obfuscator Versions / Variants.

# of Domains Deployed
- Version 1: 519
- Version 2: 89
- Version 3: 753
- Version 4: 68
- Version 5: 234
Evolution of Variants

Variants 1

Variants 2

Variants 3

Variants 4

Nuclear Obfuscator Version 2

2015/May

2015/June

2015/July

2015/August

Variants 1

Variants 2

Variants 3

Variants 4

0.81

0.85

0.75

0.67

0.66

0.47

TAKEAWAY

- A new Angle to Explore Exploit-Kit
- The novel method to boost sample set and improve detection rate by reproducing obfuscator.
- The Evolution of Obfuscator in the wild
https://github.com/irobert-tluo/rebuild_obfuscator