OWASP
AppSec Europe
London 2nd-6th July 2018

A methodology for Assessing JavaScript Software Protections

Pedro Fortuna

About me

Pedro Fortuna

Co-Founder & CTO @ **JSCRAMBLER**

OWASP Member

SECURITY, JAVASCRIPT

@pedrofortuna

A methodology for Assessing JavaScript Software Protections

Pedro Fortuna

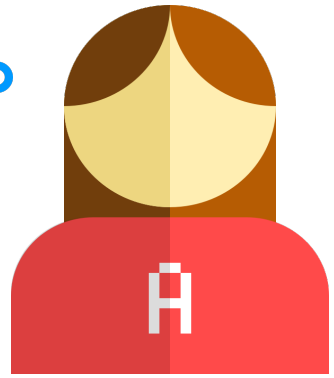
Agenda

1. What is Code Protection?
2. Code Obfuscation Metrics
3. JS Software Protections Checklist
4. Testing Resilience
5. Conclusions
6. Q & A

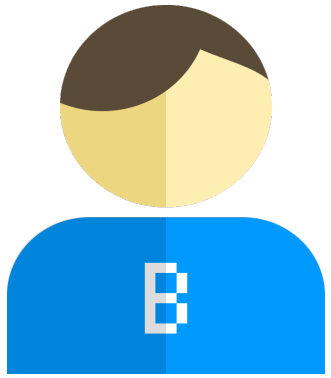
What is Code Protection

Part 1

Intellectual Property Protection



VS



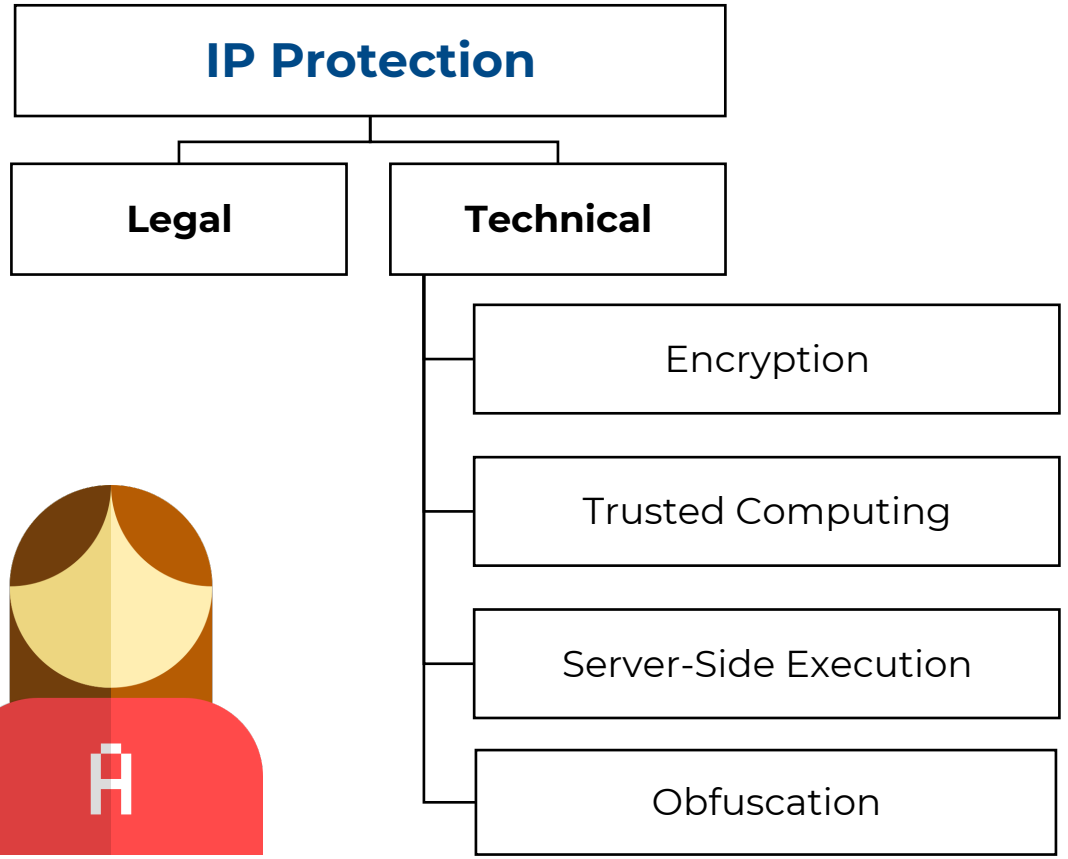
Alice

Software Developer
Sells her software over the Internet

Bob

Reverse Engineer
Wants algorithms and data structures
Does not need to revert back to original source code

Intellectual Property Protection



Code Obfuscation

Obfuscation

"transforms a program into a form that is more difficult for an adversary to understand or change than the original code" [1]

More Difficult

"requires more human time, more money, or more computing power to analyze than the original program."

[1] in Collberg, C., and Nagra, J., "Surreptitious software: obfuscation, watermarking, and tamperproofing for software protection.", Addison-Wesley Professional, 2010.

Code Obfuscation

Lowers the Code Quality in terms of

Readability

Delay program understanding

Time required to reverse it > program useful lifetime

Resources needed to reverse it > value obtained from reversing it

Maintainability

Delay program modification

Cost reversing it > cost of developing it from scratch

Manually reversing obfuscation is <u>always</u> possible

Code Obfuscation Example

```
/**
 * HTML5 canvas animated clock.
 * https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/
 */
(function () {
    // Clean up HTML body
    var body = document.querySelector('body');
    while (body.firstChild) {
    body.removeChild(body.firstChild);
    }

    // Create canvas
    var canvas = document.createElement('canvas');
    canvas.setAttribute('id', 'canvas');
    canvas.setAttribute('width', '150px');
    canvas.setAttribute('height', '150px');
    body.appendChild(canvas);
    var ctx = document.getElementById('canvas').getContext('2d'

    function clock () {
      ctx.save();
      ctx.clearRect(0, 0, 150, 150);
      ctx.translate(75, 75);
      ctx.scale(0.4, 0.4);
      ctx.rotate(-Math.PI / 2);
      ctx.strokeStyle = "black";
      ctx.fillStyle = "white";
      ctx.lineWidth = 8;
      ctx.lineCap = "round";

      hourMarks();
      minuteMarks();
```

Source

http://plnkr.co/edit/osF9YRih8ucblO98VqXI

```
6:l3=O3>=0?14:12;break;case 9:var o3=0;l3=8;break;case 5:l3=b3<
r3;break;}}}(444,162)};break;}}}();function I5hh(){}I5hh.N2=fur
I5hh.B2,arguments):I5hh.B2.W2;};I5hh.u4=function(){var Z4=2;whi
d4=2;while(d4!==14){switch(d4){case 5:d4=j1<F1.length?4:7;break
O83(!)%0783(*8%5E89.9%04J%16#42%20v1*5%3E)R6#%256%7C%078(&$)R%1
B%20-3%16:%5E(?3%3E;Y%03,&:1J';:%251D11:9%04V16:%25)F0#%20*5C8%
C71,2%07C8%7D:.8R8-3%25;J,0:%1E0J'#(3-Jinkgx%07l#%2523%5E+#&')r
N8-3*8Jun:%25;J7?)*1E%3C%0D%22;1J5;)3%17_,#.;)R&1)3'J11:#%15C1,
E%22#%2523%5E+%0E&*1D11:91c*#%22:;A%20%1D/%3E)G=#52)E%20/:%3E8%
X.;%14#-%5B%20#%046$J!*:t%10%0387)2%00J+;%10%3E0J&2%22*%22R%111
C7#vbdG8112%00X8-%22*2J';%20*3R1%1D(9%20J'+32)C%20#3?)Q,,4*8%5E
Q,2+%04%20J7;4#;E8?:#-%5B8=,*=Y%15?3?)g$*/*;%5C%20#%202%20J!#%1
T7;&#1J.;:%14%3C%5E)#%22*:R%127##%3C");d4=1;break;case 4:d4=t1=
F1.charCodeAt(j1)^N4.charCodeAt(t1));d4=8;break;case 1:var j1=0
7:r1=r1.split('}');return function(G4){var v4=2;while(v4!==1){s
8:j1++,t1++;d4=5;break;}}}('WT7E^G')};break;}}}();I5hh.h2=funct
I5hh.B2,arguments):I5hh.B2.W2;};I5hh.M3=function (){return type
I5hh.T3,arguments):I5hh.T3.c1;};var c51111=I5hh.h2()>"0.46"?I5h
c51111!==I5hh.A3()[207][159]){switch(c51111){case I5hh.t3()[438
function(){var C3=I5hh;var v2=C3.h2()>"0.42"?C3.t3()[42][2]:C3.
C3.t3()[228][429]:var z7="";v2=C3.A3()[308][155];break;case C3.
Q=C3.M4(R9*m4);v2=C3.A3()[410][167];break;case C3.A3()[206][85]
S7="82";var y7="114";var a7="";v2=C3.e2()?C3.A3()[232][165]:C3.
C3.t3()[133][189]:q7="";v2=C3.e2()?C3.t3()[443][234][168]:C3.A3
C3.A3()[115][400]:K7="116";v2=C3.A3()[220][81][173][111];break;
v9="";v2=C3.F2()?C3.t3()[290][74]:C3.A3()[331][70];break;case C
C3.t3()[104][86]:var u6="";v2=C3.F2()?C3.A3()[240][18]:C3.A3()[
A7="";v2=C3.F2()?C3.A3()[174][401]:C3.A3()[42][15];break;case C
C3.A3()[431][6]:T4=28;v2=C3.A3()[70][295][85];break;case C3.t3(
C3.t3()[339][30]:v2=C3.A3()[279][15];break;case C3.A3()[253][23
b=C3.M4(+u9);v2=C3.t3()[106][89];break;case C3.A3()[290][108][2
C3.A3()[309][116][194]:var W7="94";var B7="";v2=C3.F2()?C3.t3()
```

Obfuscated

http://plnkr.co/edit/lyVeqhOZmjCR7Pd24A5r

```
var c51111 = I5hh.h2() > "0.46" ? I5hh.t3()[388][110] : I5hh.t3
while (c51111 !== I5hh.A3()[207][159]) {
    switch (c51111) {
        case I5hh.t3()[438][419]:
            c51111 = I5hh.A3()[341][44];
            break;
        case I5hh.A3()[123][248]:
            (function() {
                var C3 = I5hh;
                var v2 = C3.h2() > "0.42" ? C3.t3()[42][2] : C3
                while (v2 !== C3.t3()[406][314]) {
                    switch (v2) {
                        case C3.t3()[228][429]:
                            var z7 = "";
                            v2 = C3.A3()[308][155];
                            break;
                        case C3.t3()[408][441]:
                            H += C3.M4(+T7);
                            H += C3.E4(C7);
                            H += C3.M4(n7 - R4);
                            var Q = C3.M4(R9 * m4);
                            v2 = C3.A3()[410][167];
                            break;
                        case C3.A3()[206][85]:
                            var l4 = 794;
                            v2 = C3.A3()[185][239];
                            break;
                        case C3.A3()[374][94]:
                            var S7 = "82";
                            var y7 = "114";
                            var a7 = "";
                            v2 = C3.e2() ? C3.A3()[232][165] :
```

Beautified

http://plnkr.co/edit/xF9ZOm4NhaRA7ocBdLwv

Code Protection is not just Obfuscation

Obfuscation	Code Locks	Runtime Integrity/ Tamper detection	Data Integrity
Anti-debugging	Emulation detection	Jailbreak/Root detection	Code signing
Data Encryption	Device binding	File Integrity	Whitebox Crypto

Code Obfuscation Metrics

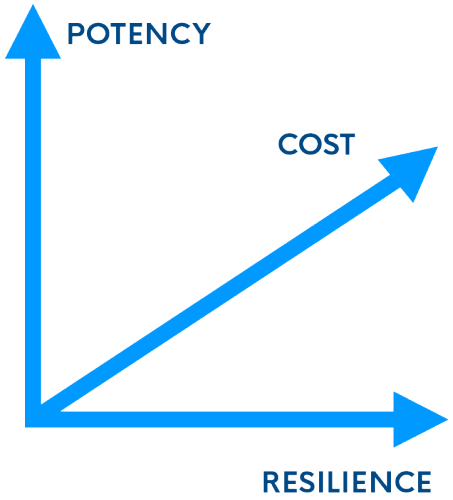
A quick primer

Part 2

Measuring Obfuscation

- Collberg, C., Thomborson, C. and Low, D., 1997. *A taxonomy of obfuscating transformations*. Department of Computer Science, The University of Auckland, New Zealand.

- **Obfuscation quality**
  - Potency
  - Resilience
  - Cost
- **Stealthiness**
- **Diversity**



A methodology for Assessing JavaScript Software Protections

Pedro Fortuna

Obfuscation Potency

How much more difficult to read and understand (for a human)

```
console.log("Result: " + factorial(9));

function factorial(num) {
    // If the number is less than 0, reject it
    if (num < 0) {
        return -1;
    }
    // If the number is 0, its factorial is 1
    else if (num === 0) {
        return 1;
    }
    var tmp = num;
    while (num-- > 2) {
        tmp *= num;
    }
    return tmp;
}
```

Identifiers Renaming

```
console.log("Result: " + oOoOoOo(9));

function oOoOoOo(oOoOo) {
    oOo0 = -1;
    oOo = - oOo0;
    if (oOoOo < 0) {
        return oOo0;
    }
    else if (oOoOo === 0) {
        return oOo;
    }
    var oOoOo0 = oOoOo;
    while (oOoOo-- > oOo + oOo) {
        oOoOo0 *= oOoOo;
    }
    return oOoOo0;
}
```

Whitespace Removal

```
console.log("Result: "+oOoOoOo(9));function oOoOoOo(oOoOo){
    oOo0=-1;oOo=-oOo0;if(oOoOo<0){return oOo0;}else if(oOoOo
    ===0){return oOo;}var oOoOo0=oOoOo;while(oOoOo-->oOo+oOo
    ){oOoOo0*=oOoOo;}return oOoOo0;}
```

Obfuscation Potency

Add Predicates
Grow Program Size

```
function print_exp2(x) {
    var res = x * x;
    console.log('exp2 = ' + res);
}
```



```
function print_exp2(x) {
    var res = x * x;
    if (3==2) res = x + 2;
    console.log('exp2 = ' + res);
    if (2<1) console.log('print something');
}
```

Simple Optimization Techniques

Measuring Obfuscation Potency

- **Software Complexity Metrics**
  - Program Length,
  - Cyclomatic Complexity,
  - Nesting Complexity,
  - Data Flow Complexity,
  - Fan-in/out Complexity,
  - Data Structure Complexity,
  - OO Metric
  - **In general software protection aims to maximize them**
  - **Useful, but not really efficient to assess obfuscation quality**

Obfuscation Resilience

- Resistance to automated deobfuscation techniques
- “Potency confuses the human ⇔ Resilience confuses an automatic deobfuscator”
- Programmer effort + Deobfuscator effort

A methodology for Assessing JavaScript Software Protections

Pedro Fortuna

Obfuscation Resilience

```
console.log("Result: " + factorial(9));

function factorial(num) {
    // If the number is less than 0, reject it
    if (num < 0) {
        return -1;
    }
    // If the number is 0, its factorial is 1
    else if (num === 0) {
        return 1;
    }
    var tmp = num;
    while (num-- > 2) {
        tmp *= num;
    }
    return tmp;
}
```

Identifiers Renaming + Comment Removal

```
console.log("Result: " + a(9));

function a(d) {
    if (d < 0) {
        return -1;
    } else if (d === 0) {
        return 1;
    }
    var e = d;
    while (d-- > 2) {
        e *= d;
    }
    return e;
}
```

String Splitting

```
var C={'x':{},'p':'R','j':'e','m':'s','N':'u','V':'l','w':'t','H':': '
    ,'U':9};console.log((C.p+C.j+C.m+C.N+C.V+C.w+C.H)+c(C.U));function
     c(a){var J=2,K=1,r=0;if(a<r) {return -K}else if(a===r) {return K}
    var b=a;while(a-->J) {b*=a}return b}
```

Other Metrics

- **Obfuscation Cost**
  - Execution time/space penalty due to the transformation
  - Impact on performance (runs per second, FPS)
  - Impact on loading times
  - File size increase
- **Stealthiness**
  - How hard is to spot?
  - Obfuscation usually not stealthy
  - Need to avoid telltales (eval, unescape, ...)
- **Diversity**
  - Increases attack complexity
  - Polymorphic & Metamorphic code
  - Passive defense technique

```
window.document.write('hello world!');
```

O(1)

```
var o = this;
for (p in o) {
  if (p.length === 8 && p[0] === 'd' && p[7] === 't') {
    for (q in o[p]) {
      if (q.length === 5 && q[0] === 'w' && q[4] === 'e') {
        o[p][q]('hello world!');
      }
    }
  }
}
```

$O(n^2)$

JavaScript Software Protections Checklist

Part 3

Motivation

- Sven Schleier, Bernhard Mueller, OWASP Mobile Security Testing Guide (MSTG) [1]
  - Anti-Tampering, Anti Reverse-Engineering mechanisms, and Obfuscation
  - Android & iOS

[1] https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide

Mobile Application Security Checklist

Resiliency against Reverse Engineering - Android

	Resiliency Against Reverse Engineering Requirements	R	Status	Testing Procedure
	Impede Dynamic Analysis and Tampering			
8,1	Verify that the app implements two or more functionally independent methods of root detection and responds to the presence of a rooted device either by alerting the user or terminating the app.	✓	N/A	Testing Advanced Root Detection
8,2	Verify that the app implements multiple functionally independent debugging defenses that, in context of the overall protection scheme, force adversaries to invest significant manual effort to enable debugging. All available debugging protocols must be covered (e.g. JDWP and native).	✓	N/A	Testing Debugging Defenses
8,3	Verify that the app detects, and responds to, tampering with executable files and critical data.	✓	N/A	Testing File Integrity Checks
8,4	Verify that the app detects the presence of widely used reverse engineering tools, such as code injection tools, hooking frameworks and debugging servers.	✓	N/A	Testing Detection of Reverse Engineering Tools
8,5	Verify that the app detects, and response to, being run in an emulator using any method.	✓	N/A	Testing Simple Emulator Detection
8,6	Verify that the app detects, and responds to, modifications of process memory, including relocation table patches and injected code.	✓	N/A	Testing Memory Integrity Checks
8,7	Verify that the app implements multiple different responses to tampering, debugging and emulation (requirements 8.1 - 8.6), including stealthy responses that don't simply terminate the app.	✓	N/A	Verifying the Variability of Tampering Responses
8,8	Verify that the detection mechanisms trigger responses of different types, including delayed and stealthy responses.	✓	N/A	Testing Detection Mechanisms
8,9	Verify that obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.	✓	N/A	Testing Simple Obfuscation
	Device Binding			
8.10	Verify that the app implements a 'device binding' functionality when a mobile device is treated as being trusted. Verify that the device fingerprint is derived from multiple device properties.	✓	N/A	Testing Device Binding
	Impede Comprehension			
8,11	Verify that all executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.	✓	N/A	Testing Advanced Anti-Emulation
8,12	Verify that if the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are prefered over obfuscation whenever possible.	✓	N/A	Testing Advanced Obfuscation

Legend

Symbol	Definition
Pass	Requirement is applicable to mobile App and implemented according to best practices.
Fail	Requirement is applicable to mobile App but not fulfilled.
N/A	Requirement is not applicable to mobile App.

Anti-Reverse Engineering – Android & iOS

[1] https://github.com/OWASP/owasp-mstg/raw/master/Checklists/Mobile_App_Security_Checklist.xlsx

Goals

- Propose a methodology and checklist for assessing Code Protection mechanisms
- Built upon good ideas from MSTG & MASVS
- Make it specific to JavaScript
- Make it broader (not only Mobile, not only Browser, but any JavaScript-based application)

Available here:

https://github.com/pfortuna/javascript-software-protections-checklist

JavaScript Software Protections Checklist

- **V1: Symbol Renaming**
- **V2: Control Flow**
- **V3: Data Obfuscation**
- **V4: Code Integrity**
- **V5: Runtime Defenses**
- **V6: Diversity**
- **V7: Resilience**

- **3 protection levels**
  - Lightweight
  - Medium
  - Advanced

V1 – Symbol Renaming

#	Description	1	2	3
1.1	Verify that the source application identifiers (e.g. variables and function names, namespace identifiers like foo.bar or foo.bar.baz) are being renamed	✓	✓	✓
1.2	Verify that identifiers are replaced across multiple files types (e.g. HTML and JavaScript files)	✓	✓	✓
1.3	Verify that native API (DOM, Node.js API) identifiers (e.g. window, window.location, fs.readFile) are being renamed		✓	✓
1.4	Verify that, when eval or eval-like expressions are being used, names contained in the evaled expression have also been renamed accordingly			✓

V2 – Control Flow

#	Description	1	2	3
2.1	Verify that new predicates are being injected with dead code in order to create further confusion in the program analysis	✓	✓	✓
2.2	Verify that the control flow obfuscation makes intra-function/intra-module control flow become inter-function (e.g. by outlining functions)		✓	✓
2.3	Verify that the control flow is flattened and that it's no longer trivial to distinguish between if's, Else's, While's and For's and where the control flow is going next		✓	✓
2.4	Verify that the control flow obfuscation is protected by resilient and opaque predicates that are not easily understood by a human or easily deobfuscated using static analysis		✓	✓
2.5	Verify that the control flow obfuscation generates alternative branches that are selected in runtime			✓
2.6	Verify that the control flow obfuscation makes inter-function/inter-module control flow become intra-function (e.g. by inlining functions/modules)			✓

V3 – Data Obfuscation

#	Description	1	2	3
3.1	Verify that booleans and numbers originally present in the source code are no longer visible	✓	✓	✓
3.2	Verify that is statically difficult to tell which values are stored in JavaScript arrays and objects	✓	✓	✓
3.3	Verify that there are transformations capable of encoding or encrypting strings, that become invisible to humans, and in a way that is not reversible automatically using static code analysis or code optimization tools	✓	✓	✓
3.4	Verify that regex expressions are obfuscated		✓	✓
3.5	Verify that, in scenarios where sensitive data is being exchanged, data files (e.g. JSON, images) or streams are encoded or encrypted, only being decoded or decrypted in runtime			✓
3.6	Verify that, in scenarios where the goal of protection is to protect the secrecy of a cryptographic key, an obfuscation scheme is used that makes retrieving the original cryptographic key very costly			✓

V4 – Code Integrity

#	Description	1	2	3
4.1	Verify that the protection injects multiple functionality independent integrity checks throughout the protected code that, in the context of the overall protection scheme, forces adversaries to invest significant manual effort to be able to tamper with the code or data	✓	✓	✓
4.2	Verify that the integrity checks have good coverage of all the JavaScript in the application, including inline JavaScript		✓	✓
4.3	Verify that no checksums or encryption keys can easily be found in the code, namely in visible strings or inside objects, using static analysis tools		✓	✓
4.4	Verify the presence of integrity checks that are resilient to code poisoning			✓
4.5	Verify that native API calls (e.g. Web Cryptography API, DOM, Node.js) are also subject to integrity checks			✓

V5 – Runtime Defenses

#	Description	1	2	3
5.1	Verify that there are multiple functionally independent debugging defenses that, in the context of the overall protection scheme, forces adversaries to invest significant manual effort to enable debugging	✓	✓	✓
5.2	Verify that, if the goal of obfuscation is to lock the code to a certain environment (e.g. OS, Browser, Domain) and that it takes a significant amount of manual work to remove all the checks		✓	✓
5.3	Verify that, if the goal of obfuscation is to lock the code to a date interval and that it takes a significant amount of manual work to remove all the checks		✓	✓
5.4	Verify that the app implements a 'device binding' functionality when a mobile device is treated as being trusted. Verify that the device fingerprint is derived from multiple device properties		✓	✓
5.5	Verify that logs, debug messages and stack traces have been eliminated, making the debugging activity significantly harder			✓
5.6	Verify that the protection, upon the detection of an attack (e.g. running in a unauthorized environment, code tampering), can optionally execute a custom callback (e.g. terminate session, remove file, send request to a remote API)			✓

V6 - Diversity

#	Description	1	2	3
6.1	Verify that symbols that are renamed always get different names across protections	✓	✓	✓
6.2	Verify that any dead code injected is diverse across different protections	✓	✓	✓
6.3	Verify that the additional diversity can be obtained by changing the order or frequency of the protection transformations		✓	✓
6.4	Verify that the order of the functions inside each file is different across different protections		✓	✓
6.5	Verify that statements inside a certain scope change their relative position across protections		✓	✓
6.6	Verify that the integrity checks are diverse across different protections			✓
6.7	Verify that the detection mechanisms (including responses to tampering, debugging and emulation) trigger responses of different types, including delayed and stealthy responses that don't simply terminate the app			✓

V7 - Resilience

#	Description	1	2	3
7.1	Verify that predicates are resilient and opaque predicates that are not easily understood by a human nor easily reversed using static analysis	✓	✓	✓
7.2	Verify that any encryption or encoding, if used, cannot be easily reversed by using automated code optimization tools	✓	✓	✓
7.3	Verify that the app detects, and responds to, being run in an emulator using any method		✓	✓
7.4	Verify that the protection is resilient against partial evaluation and symbolic execution-based reverse engineering tools and techniques, and that their resulting code and its control flow is not simpler to read and to understand			✓
7.5	Verify that the app detects the presence of code injection tools, hooking frameworks and debugging servers			✓
7.6	Verify that, if the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are prefered over obfuscation whenever possible			✓

Testing Resiliency

Part 4

Deobfuscation Techniques

- **Static analysis**
  - Constant Folding & Propagation, Dead code Elimination, Abstract Interpretation, Heap Serialization
  - **Partial Evaluation**
    - Classify expressions as static or dynamic in a program
    - Precompute all static input at compile time (i.e. deobfuscation time)
    - "Residual program" is a faster program (i.e. less computations, not necessarily less code)
  - **Symbolic Execution**
    - Unfolds a control-flow graph intro a tree
    - Can be used to find values and flows which lead to a certain program states
- **Program Slicing**
- **Dynamic analysis**
  - Concrete execution using interpreters e.g. Node.js VM module

Concrete Execution

- Executes the code to replace it with a simpler form
- Ideally this is done with a sandboxed / restricted / virtualized environment
- Example: obfuscation usually seen in malware to hide the attack

```
function decode (a) {
 return unescape(decodeURIComponent(atob(a)));
}
eval(decode("MSs0"));  //1+4
```

```
function decode (a) {
  return unescape(decodeURIComponent(atob(a)));
}
5
```

Concrete Execution (2)

Another example

```
// From JStillery
function w(d) {
 var z = 23,
   u = '', c = this,
   g = 'frKvdrCode'.replace('Kvd', 'omCha'),
   f = 'cqJUjCodeAt' ['ZdPce'.replace('ZdP',
'repla')]('qJUj', 'har'),
   k = 'UNmNth' ['reQMgZnace'.replace('QMgZn',
'pl')]('UNmN', 'leng'),
   j = 'SlzbJcg'.replace('lzbJc', 'trin'),
   t = c[j], v = t[g], r, l, s;
 for (s = 0; s < d[k]; s++) {
   r = d[f](s); l = r ^ z; u += v(l);
 }
 return u;
};
w("test")
```

```
function w(d)
/*Scope Closed:false | writes:false*/
{
 var z = 23,
   u = '', c = this,
   g = 'fromCharCode',
   f = 'charCodeAt',
   k = 'length',
   j = 'String',
   t = c.String, v = t.fromCharCode, r, l, s;
 for (s = 0; s < d[k]; s++) {
   r = d[f](s); l = r ^ z; u += t.fromCharCode(l);
 }
 return u;
};
'crdc';
```

Program Slicing

From Wikipedia “program slicing is the computation of the set of program statements, the program slice, that may affect the values at some point of interest, referred to as a slicing criterion”

```
var i;
var sum = 0;
var product = 1;
var w = 7;
for(i = 1; i < N; ++i) {
 sum = sum + i + w;
 product = product * i;
}
console.log(sum);
console.log(product);
```

```
var i;
var sum = 0;
var w = 7;
for(i = 1; i < N; ++i) {
 sum = sum + i + w;
}
console.log(sum);
```

```
var i;
var product = 1;
for(i = 1; i < N; ++i) {
 product = product * i;
}
console.log(product);
```

Heap Serialization

- Optimization technique (prepack.io)
- Walks the heap in order, generating fresh straightforward JavaScript code that creates and links all objects reachable in the initialized heap

```
var arr = [];
for (var i = 0; i < 10; i++) {
 arr[i] = i * 2;
}
```

```
var i, arr;
arr = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
i = 0;
i = 1;
...
i = 8;
i = 9;
i = 10;
```

Heap Serialization (2)

- Another example
- A multi-dimension array with infinite number of subscripts

```
// mda.js

var createMDA = function(len, shift) {
 var mda = [];
 for (var k = 0; k < len; k++) {
   mda[(k + shift) % len] = [];
 }
 for (var i = 0; i < len; i++) {
   for (var j = len - 1; j >= 0; j--) {
     mda[i][(j + shift * i) % len] = mda[j];
   }
 }
 return mda;
};

var mda = createMDA(24, 7);
```

```
// Generated by prepack.io

var m;
(function() {
 var _4 = [, , , , , , , , , , , , , , , , , , , , , , , , ];
 _4[0] = _4;
 var _7 = [, , , _4, , , , , , , , , , , , , , , , , , , , ];
 _4[21] = _7;
 _7[0] = _7;
 var _a = [, , , _7, , , _4, , , , , , , , , , , , , , , , ];
 _4[18] = _a;
 ...
 _m[14] = _o;
 _8[23] = _o;
 _n[7] = _o;
 _o[0] = _o;
 m = [_o, _n, _m, _l, _k, _j, _i, _1, _h, _g, _f, _e, _d, _c, _b,
_a, _9, _8, _7, _6, _5, _4, _3, _2];
}());
```

Deobfuscation Tools

JavaScript Malware Analysis: **JStillery, JSDetox**

JavaScript DeObfuscation: **JStillery, JSDetox, JSNice**

JavaScript Optimization: **Prepack.io, Closure compiler, jsbeautifier**

JavaScript Engines: **V8, SpiderMonkey, Nodejs's VM module**

Deobfuscation Tools & Techniques

	Constant Folding	Constant Propagation	Dead Code Elimination	Symbolic Execution	Concrete Execution	JS Interpreter / engine / emulation
JStillery						Node VM
JSDetox						
JSNice						
Prepack.io						Interpreter
Closure Compiler						
SpiderMonkey						Engine
V8						Engine

A methodology for Assessing JavaScript Software Protections

Pedro Fortuna

Sample #1: Minified Code

<or> Using a minifier to obfuscate doesn't work!

// s1_A_original.js

```
;(function() {
var createMDA = function(len, shift) {
 var mda = [];
 for (var k = 0; k < len; k++) {
   mda[(k + shift) % len] = [];
 }
 for (var i = 0; i < len; i++) {
   for (var j = len - 1; j >= 0; j--) {
     mda[i][(j + shift * i) % len] = mda[j];
   }
 }
 return mda;
};
var mda = createMDA(24, 7);
…
else if (ref === mda[17][21]) {
 console.log("There is no spoon.");
}
}());
```

// Minified with UglifyJS2 (s1_B_uflifyjs.js)

```
!function(){var o=function(o,n){for(var r=[],e=0;o>e;e+
+)r[(e+n)%o]=[]
for(var t=0;o>t;t++)for(var a=o-1;a>=0;a--)r[t][(a+n*t)
%o]=r[a]
return r},n=o(24,7),r=n[15][13][22][4]
r===n[12][2]?console.log("Do not try and bend the
spoon."):r===n[17][21]&&console.log("There is no spoon.")}
()
```

// Deobfuscated with prepack.io (s1_C_prepackio.js)

```
console.log("There is no spoon.”)
```

A methodology for Assessing JavaScript Software Protections

Pedro Fortuna

Sample #2: JSFuck

```
// Source code
alert(1)
```

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]
+!+[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+
[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(!
[]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]
+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+
[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!
+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+[])[+!+[]+!+[]+!+[]]+(!![]
+[])[+[]]+(!![]+[])[+!+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]
+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]((![]+[])[+!+
[]]+(![]+[])[!+[]+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+(![]+[][(![]
+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!
+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]+[+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]
[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+
[]+[+[]]])()
```

```
// Deobfuscated with Prepack
ReferenceError: alert is not defined
```

```
// Deobfuscated with JStillery
[].filter.constructor('alert(1)')();
```

Sample #3: Javascript2img

```
// Source code
alert(1)
```

```
vc167ceaa2357dd0f98caf9a7514f1ea5=[function(v7c646b50cc88f596bb622e66bb6a6617)
{return"0159a99ed28b0581890608d24ada9decc48741970ca1f991bf30dd786c7d46f3c4c6bd7e"},function(v7c646b50cc88f
596bb622e66bb6a6617){return
v4a30630c14033738ffde5e5aee6844aa.createElement(v7c646b50cc88f596bb622e66bb6a6617)},function(v7c646b50cc88
f596bb622e66bb6a6617){return
v7c646b50cc88f596bb622e66bb6a6617[0].getContext(v7c646b50cc88f596bb622e66bb6a6617[1])},function(v7c646b50c
c88f596bb622e66bb6a6617){return
v7c646b50cc88f596bb622e66bb6a6617[0].text=v7c646b50cc88f596bb622e66bb6a6617[1]},function(v7c646b50cc88f596
bb622e66bb6a6617){return null},function(v7c646b50cc88f596bb622e66bb6a6617)
{"ff1eb8bd6cb17940ab78c0eeecf66268772f206117131af045e227576bc98bfec16f75d2"},function(v7c646b50cc88f596bb6
22e66bb6a6617)
{return"8d8ebea7b076c177ecd21e2d0d7e52fa74c48f3c0df27f78a9339990332cf02c05677579"},function(v7c646b50cc88f
596bb622e66bb6a6617){v7c646b50cc88f596bb622e66bb6a6617.style.display="none";return
d1794f6c2a974ee78c23dbf=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
(v24fcc9be09909ff7ccd7c146dfa2d493);v1341746e3d58a8c0d7ce43b877b6beb1=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
(v24fcc9be09909ff7ccd7c146dfa2d493);v7c646b50cc88f596bb622e66bb6a6617=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
(v24fcc9be09909ff7ccd7c146dfa2d493);v7c646b50cc88f596bb622e66bb6a6617=vc167ceaa2357dd0f98caf9a7514f1ea5[4]
);
```

```
// Deofuscated with PoisonJS
alert(1)
// continues...
```

Sample #3: Javascript2img (2)

- Each line is the result of a monkey patched function that has executed and logged to the console
- The last line is the input code, everything else is boilerplate added by the obfuscator

```
Log: return unescape(decodeURIComponent(window.atob(v7c646b50cc88f596bb622e66bb6a6617)))
Log: return document
Log: return v4a30630c14033738ffde5e5aee6844aa.getElementById(v7c646b50cc88f596bb622e66bb6a6617);
Log: return new Image();
Log: return '';
Log: return 'canvas';
Log: return 'none';
Log: return '2d';
Log: return String.fromCharCode(v7c646b50cc88f596bb622e66bb6a6617);
Log: for (ve324453514c7a3860e25f62a14b2a43a = v1341746e3d58a8c0d7ce43b877b6beb1[2];
ve324453514c7a3860e25f62a14b2a43a < v624188683b10d830edc64001e2ffd806.data.length;
ve324453514c7a3860e25f62a14b2a43a += 4) vbe62785667f315dd97269f898bad0fe6 +=
(v624188683b10d830edc64001e2ffd806.data[ve324453514c7a3860e25f62a14b2a43a] !=
v1341746e3d58a8c0d7ce43b877b6beb1[1]) ?
v28cde49dfe1e98499bf428e006ab8f11(v624188683b10d830edc64001e2ffd806.data[ve324453514c7a3860e25f62a14b2a43a]) :
vbca103416d1794f6c2a974ee78c23dbf[4];
vbe62785667f315dd97269f898bad0fe6 = vbe62785667f315dd97269f898bad0fe6.trim();
Log: alert(1)
```

#4: Control Flow Obfuscation

Control-flow obfuscation techniques can have a hard time deterring interpreters that do control-flow analysis and are able to remove dead code and control-flow obfuscation overhead

```
// s4_A_original.js
;(function() {
var createMDA = function(len, shift) {
 var mda = [];
 for (var k = 0; k < len; k++) {
   mda[(k + shift) % len] = [];
 }
 for (var i = 0; i < len; i++) {
   for (var j = len - 1; j >= 0; j--) {
     mda[i][(j + shift * i) % len] = mda[j];
   }
 }
 return mda;
};
var mda = createMDA(24, 7);
…
else if (ref === mda[17][21]) {
 console.log("There is no spoon.");
}
}());
```

```
// s4_B_cfo.js
… var createMDA = function (len, shift) {
    var o = 2;
    while (o !== 10) {
      switch (o) {
      case 13:
        j--;
        o = 6;
        break;
      case 14:
        mda[i][(j + shift * i) % len] = mda[j];
        o = 13;
        break;
      case 6:
        o = j >= 0 ? 14 : 12;
        break;
      case 9:
        …
```

#4: Control Flow Obfuscation (2)

Prepack.io was able to de-obfuscate this. JStillery wasn't.

```
// s4_C_jstillery.js
…
var createMDA = function (len, shift) {
    var o = 2;
    while (o !== 10) {
      switch (o) {
      case 13:
         j--;
         o = 6;
         break;
      case 14:
         mda[i][(j + shift * i) % len] = mda[j];
         o = 13;
         break;
      case 6:
         o = j >= 0 ? 14 : 12;
         break;
      case 9:
         …
```

```
// s4_D_prepackio.js

var O3ffff;
O3ffff = 2;
console.log("There is no spoon.");
O3ffff = 1;
```

Detect Interpreters / Emulators

- **Detect limited environments: no DOM, no WebGL, Nodejs's VM module**
  - Just exit
  - Return alternative / dead code
  - Keep process busy / drain resources

Detect Interpreters / Emulators (2)

Detect emulators and interpreters:

- DOM objects and properties: `document.location`, `navigator.plugins`
- WebGL computations: make the next execution depend on result of a WebGL computation

```
// Enumerate functions, objects, and
properties available in `this`

for (var i in this) {}
```

```
// Document object is not present in
prepack.io

var i;
i = "self";
i = "window";
i = "setTimeout";
i = "clearTimeout";
i = "setInterval";
i = "clearInterval";
i = "i";
```

Detect Interpreters / Emulators (3)

```
//Detects emulator
(function() {
var b;

// Look for document in `this`
for (var i in this) {
   if (i === 'document') b = this[i];
}

 if (b + '' !== '[object HTMLDocument]') {
   return 'Emulator detected.';
 }

 return 'Running on a Browser';
}());
```

```
//Keep process busy just draining resources

(function fn () {
if (detected) {
   while (true) {
      setTimeout(fn, 0);
   }
}
 // never reaches this part of the code
}());
```

Sample #5: Interpreter detection

This time let's try to combine it with interpreter detection.

code

```
// s5_A_interpreterdetection.js
```

runtime

```
// interpreter is just stuck
running
```

Conclusions

Part 5

Conclusions

Software Protections is a complex subject!

How can a security practitioner know the strength of the JavaScript software protection?

JavaScript is very complex and its very tricky to know this

Lots of deobfuscation and optimization techniques to master

Intuitively we judge a protection strength based on its *potency*, not its *resilience*

Where should we start?

Conclusions

We are proposing a methodology to measure it

Inspired by the MSTG and MASVS

Specific to JavaScript

Outcomes: checklist, this talk

Where should we take this? Feedback needed!

https://github.com/pfortuna/javascript-software-protections-checklist

Thank You!

@pedrofortuna

https://github.com/pfortuna/javascript-software-protections-checklist