



# Nozzle:

## A Defense Against Heap-spraying Code Injection Attacks

Paruj Ratanaworabhan, Cornell University  
Ben Livshits and Ben Zorn, Microsoft Research  
(Redmond, WA)

# Heap Spraying is a Problem

Thursday

When PDFs

The Shadowserve Acrobat affecting exploited. We are sample last week clear that we did others are aware Reader 8.1.0, 8.1. confirmed via tes will also affect it a

...

However, it would you **DISABLE JAVA** functionality and should be an easy

Disabling JavaScript  
Click: Edit -> Pref

RITY WEBLOG

## FireEye Malware Intelligence Lab

Threat research, analysis, and mitigation | [www.fireeye.com](http://www.fireeye.com)

[Home](#) | [Archives](#) | [Subscribe](#)

2009

He

W

Int

As

how

### Background Summary

Most of the Acrobat exploits over the last several months use the, now common, [heap spraying technique](#), implemented in [Javascript/ECMAScript](#), a [Turing complete](#) language that Adobe thought would go well with static documents. (Cause that [went so well for Postscript](#)) (Ironically, PDF has now come full circle back to having the [features of Postscript](#) that it was [trying to get away from](#).) The exploit could be made far far *less* reliable, by [disabling Javascript in your Adobe Acrobat Reader](#).

But apparently there's no easy way to disable Flash through the UI. [US-CERT](#) recommends renaming the %ProgramFiles%\Adobe\Reader 9.0\Reader\authplay.dll and %ProgramFiles%\Adobe\Reader 9.0\Reader\rt3d.dll files. [Edit: Actually the source for this advice is the [Adobe Product Security Incident Response Team \(PSIRT\)](#).]

Anyway, here's why... Flash has it's own version of ECMAScript called [Actionscript](#), and whoever wrote this new 0-day, finally did something new by implementing the heap-spray routine with Actionscript inside of Flash.

[Details](#)

[http://blog.fireeye.com/research/2009/07/actionscript\\_heap\\_spray.html](http://blog.fireeye.com/research/2009/07/actionscript_heap_spray.html)

[html](#)

GOOGLE SEARCH

Google™ Custom Search

Search

BLOG ARCHIVE

▼ 2009 (140)

► August (11)

▼ July (33)

LuckySploit \*\*

\*.8866.org,podzone.o  
ulaiba.net...

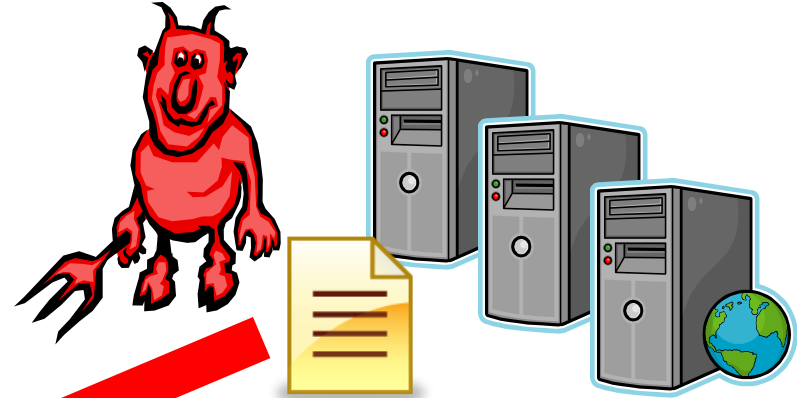
Spam \*\* 29 July

LuckySploit \*\* siyou.org

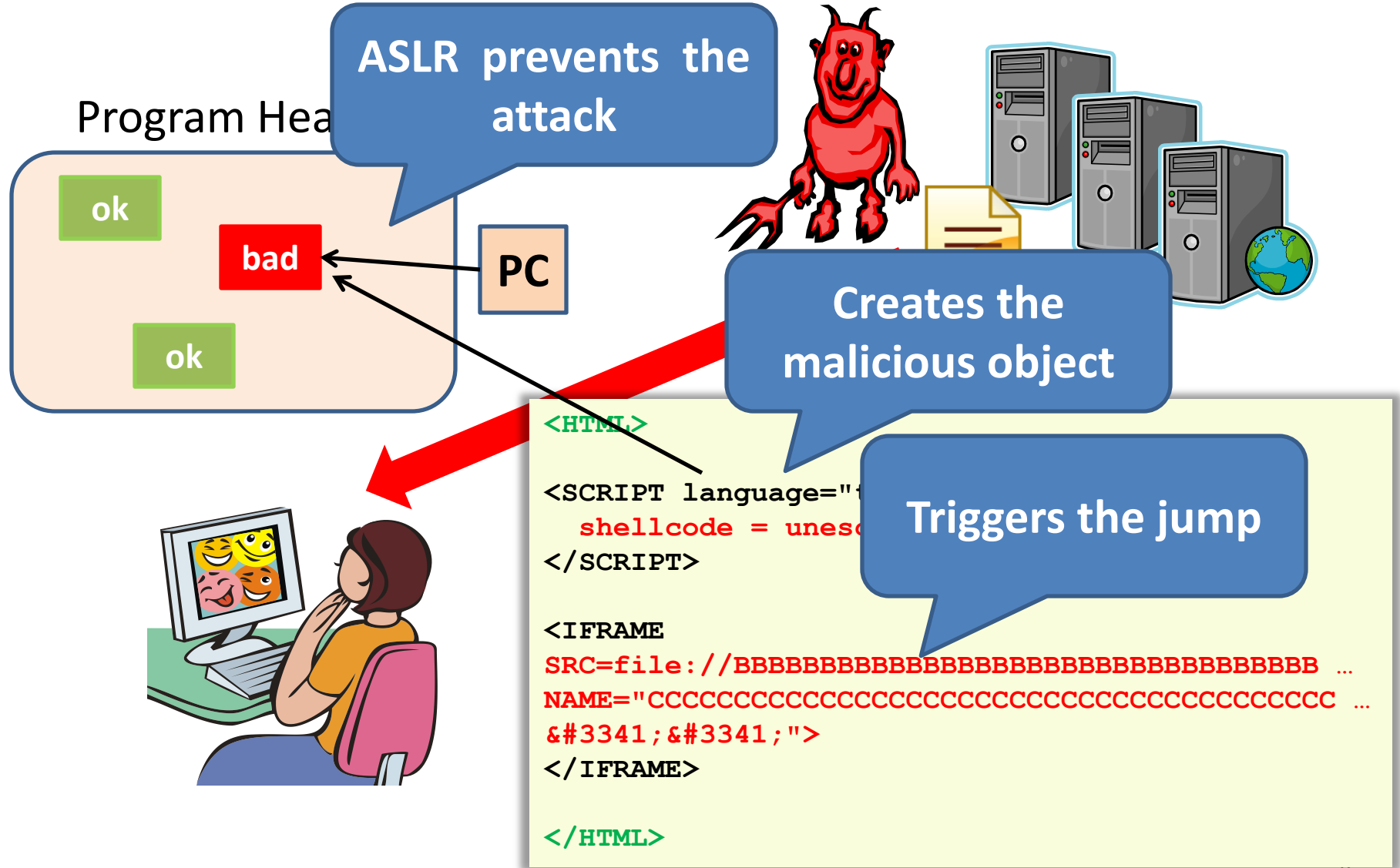
Spam \*\* 27 July

LuckySploit \*\*

# Drive-By Heap Spraying



# Drive-By Heap Spraying (2)

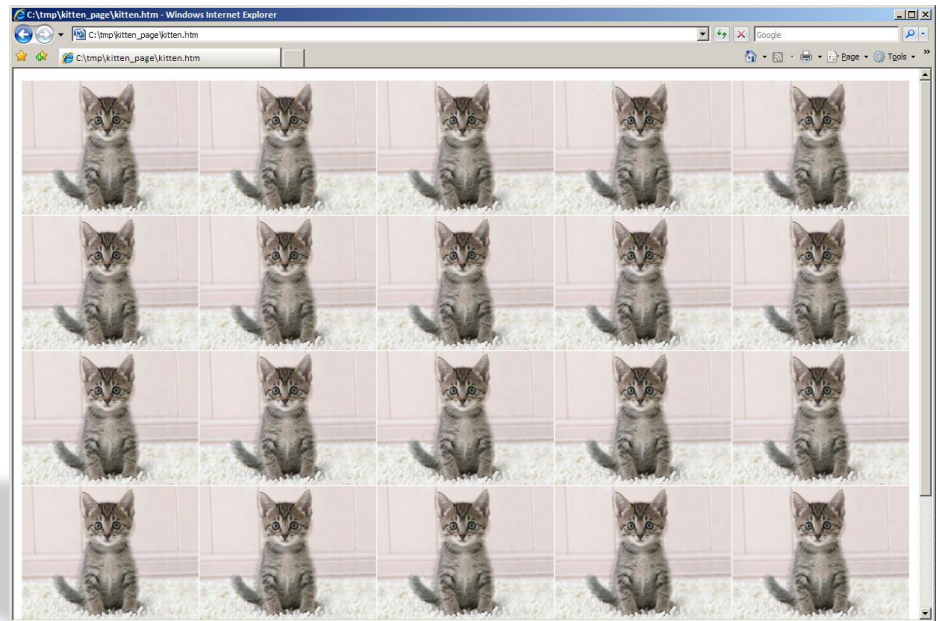




# Kittens of Doom

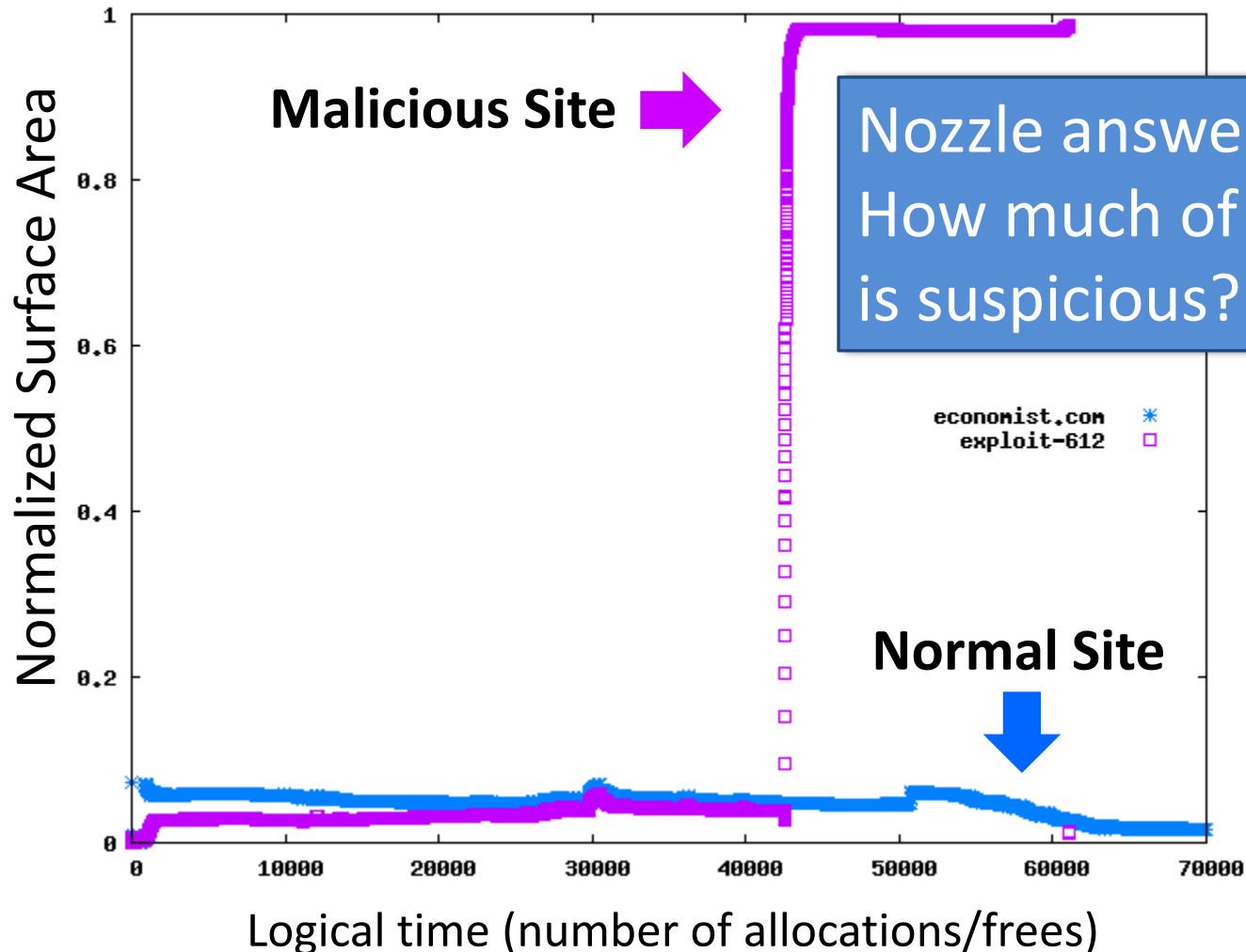
## What data can you trust?

- Heap spraying is quite general, easy to implement
- Many applications allow scripts in type safe languages
  - JavaScript, ActionScript
  - Java, C#
- Many applications accept data from untrusted sources
  - Embed malicious code in images, documents, DLLs, etc.
- [Sotirov & Dowd BH'08]



# Nozzle – Runtime Heap Spraying Detection

Application: Web Browser



# Outline

- Nozzle design & implementation
- Evaluation
  - False positives
  - False negatives
  - New threats (Adobe Reader)
- Summary



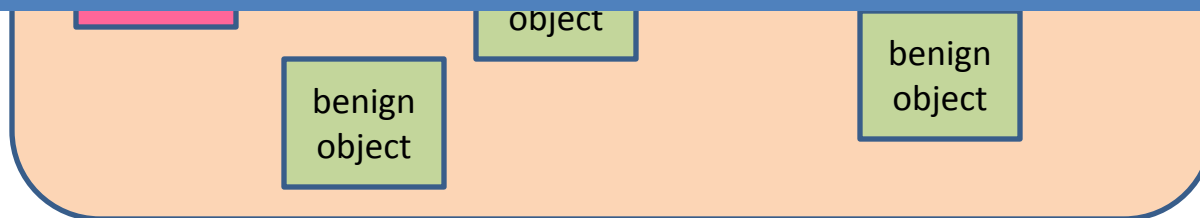
# Nozzle Design

Application Threads

Nozzle Threads

## Advantages

- Just need to hook standard APIs – malloc, free, HeapAlloc, HeapFree, etc.
- Monitor new applications using Detours
- Can be applied to existing binaries

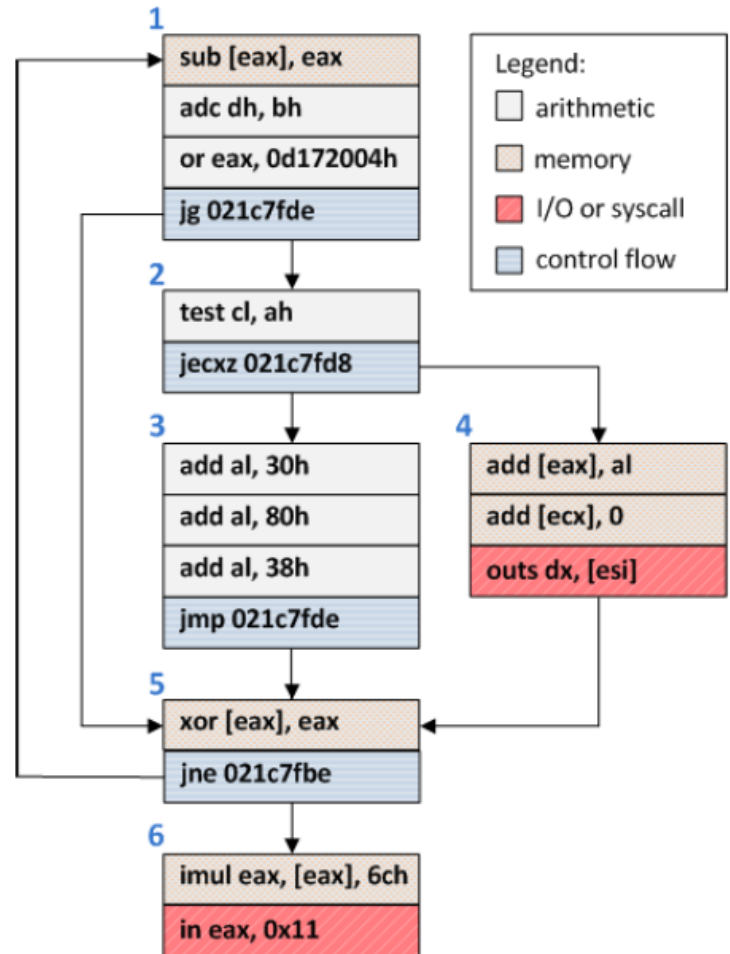


Application Heap



# Object Surface Area Calculation (1)

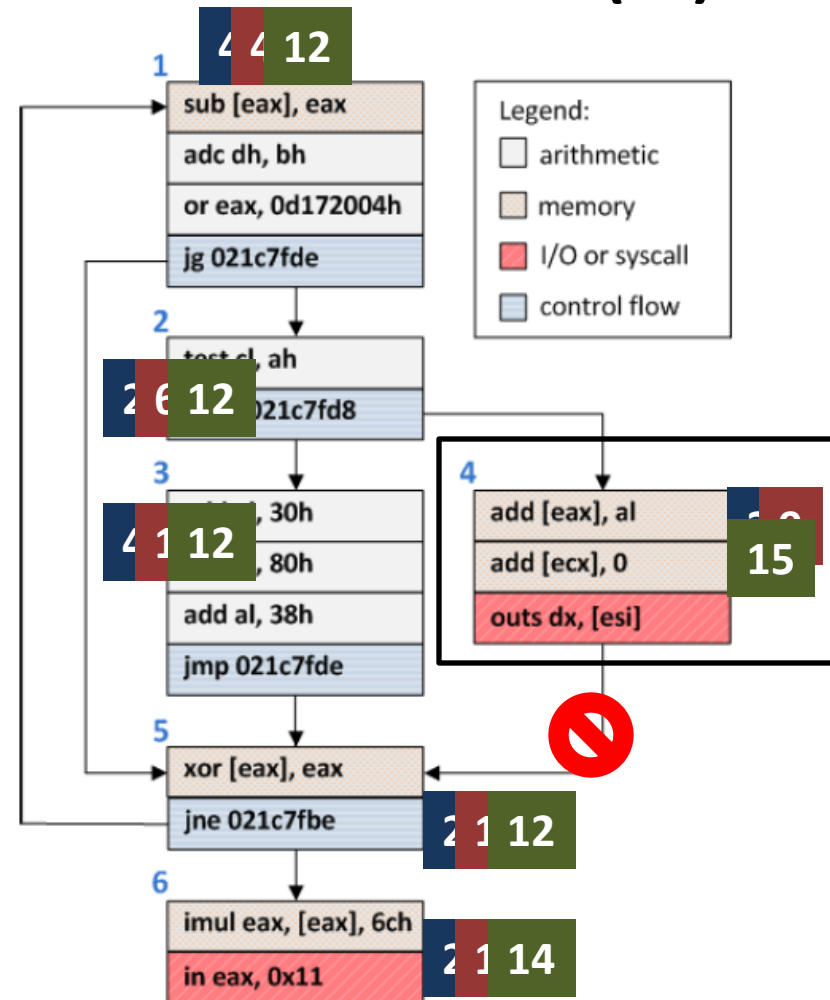
- Assume: attacker wants to reach shell code from jump to any point in object
- Goal: find blocks that are likely to be reached via control flow
- Strategy: use dataflow analysis to compute “surface area” of each block



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)

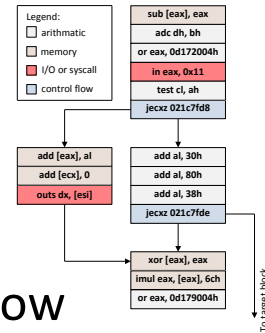
$NSA(H)$

$obj$



build CFG

$B_i$



dataflow

Compute threat  
of entire heap

$SA(H)$

Compute threat of  
single block

$SA(B_i)$

Compute threat of  
single object

$SA(o)$

# 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

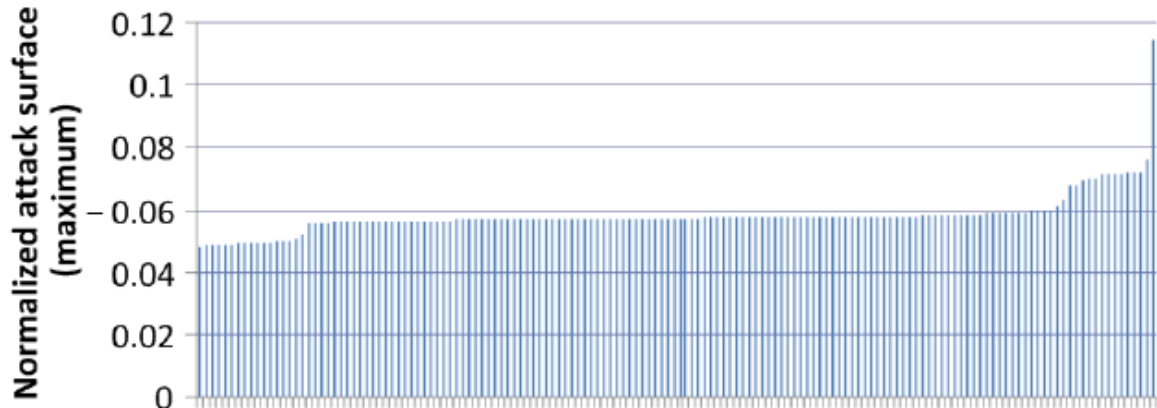
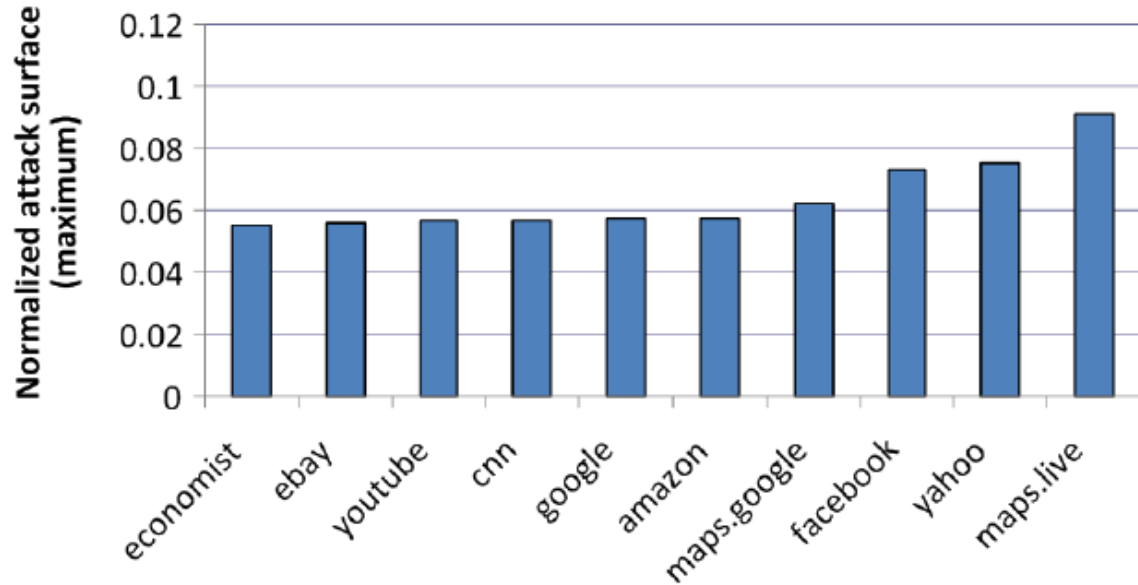


## Runtime Overhead

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

# Nozzle on Benign Sites

- Benign sites have low Nozzle NSA
- Max NSA always less than 12%
- Thresholds can be set much higher for detection (50% or more)



Alexa top 150 sites

# Nozzle with Known Heap Sprays

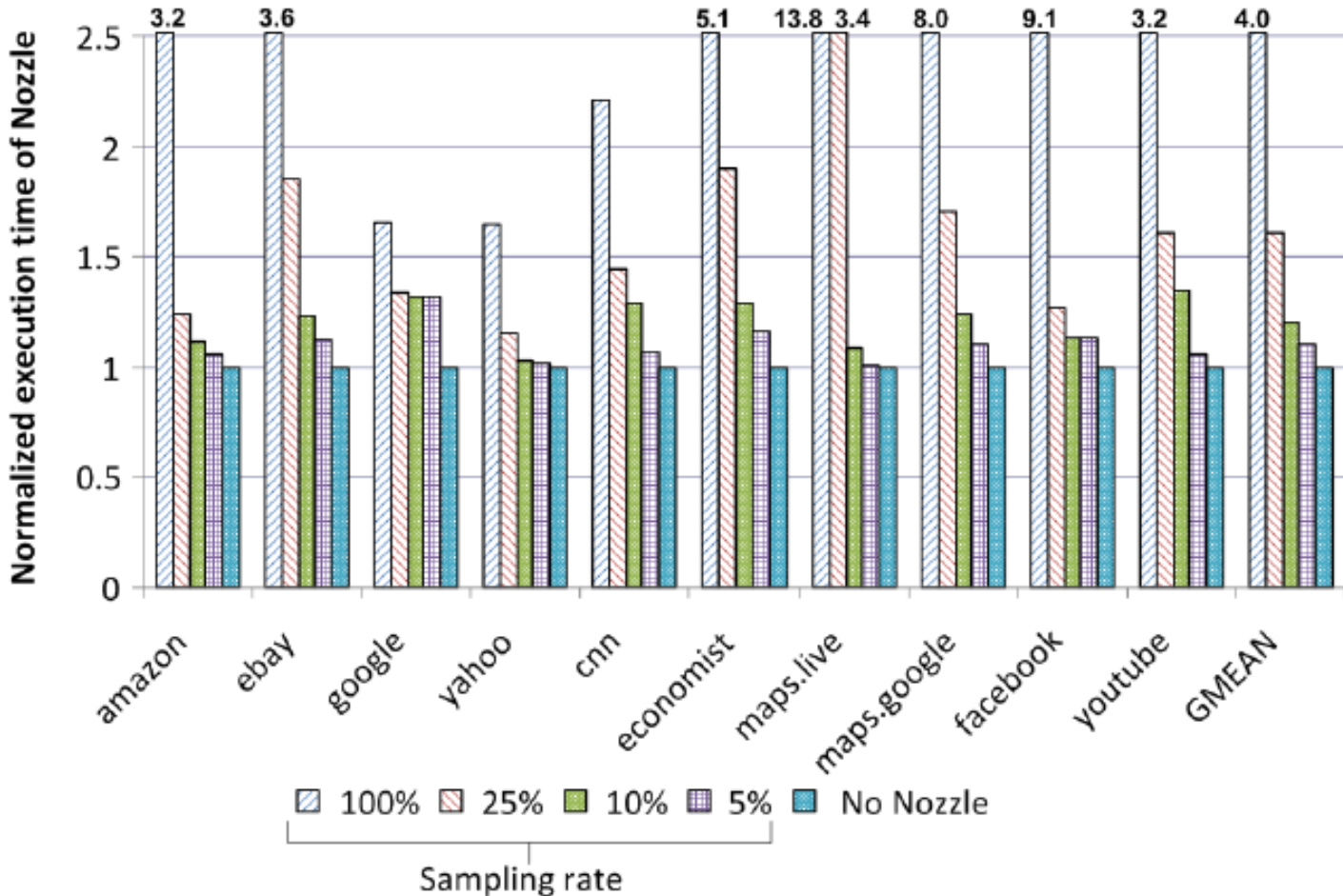
- 12 published heap spray pages in multiple browsers
- 2,000 synthetic heap spray pages using MetaSploit
  - advanced NOP engine
  - shellcode database

Date	Browser	Description	milw0rm
11/2004	IE	IFRAME Tag BO	612
04/2005	IE	DHTML Objects Corruption	930
01/2005	IE	.ANI Remote Stack BO	753
07/2005	IE	javaprxy.dll COM Object	1079
03/2006	IE	createTextRang RE	1606
09/2006	IE	VML Remote BO	2408
03/2007	IE	ADODB Double Free	3577
09/2006	IE	WebViewFolderIcon setSlice	2448
09/2005	FF	0xAD Remote Heap BO	1224
12/2005	FF	compareTo() RE	1369
07/2006	FF	Navigator Object RE	2082
07/2008	Safari	Quicktime Content-Type BO	6013

Result: max NSA between 76% and 96%  
Nozzle detects real spraying attacks



# Nozzle Runtime Overhead



# Using Nozzle in Adobe Reader

AcroRd32.exe

det-  
AcroRd32.exe

## Demo

### Results

- Detected (NSA > 75%)
- Runtime overhead was 8% on average
- NSA of normal document < 10%

# Summary

- Heap spraying attacks are
  - Easy to implement, easy to retarget
  - In widespread use
- Existing detection methods fail to classify malicious objects on x86 architecture
- Nozzle
  - Effectively detects published attacks (known and new)
  - Has acceptable runtime overhead
  - Can be used both online and offline

# Questions?

Paruj Ratanaworabhan ([paruj.r@gmail.com](mailto:paruj.r@gmail.com))

Ben Livshits ([livshits@microsoft.com](mailto:livshits@microsoft.com))

Ben Zorn ([zorn@microsoft.com](mailto:zorn@microsoft.com))

Nozzle heap spraying



See us on Channel 9:

<http://channel9.msdn.com/posts/Peli/>

[Heap-Spraying-Attack-Detection-with-Nozzle/](http://channel9.msdn.com/posts/Peli/Heap-Spraying-Attack-Detection-with-Nozzle/)

# Backup

# Attacks on Nozzle

- Injecting junk into start of object
  - Where does the exploit code begin?
- TOCTTOU – When do you scan the object?
- Attacks on surface area calculation
  - Jumps outside of objects
  - Multiple instances of shellcode inside an object
- Hiding the code itself
  - Code that rewrites heap at last minute

# What about Data Execution Prevention?

- DEP / NX bit = hardware to prevent code execution on the heap
- DEP is great , but isn't used everywhere
  - Issues with app compatibility
  - DEP can be circumvented
  - JIT compilers complicate the story
- Nozzle augments DEP for defense in depth

# Normalized Surface Area Locally

