Malware analysis

An overview and some key challenges

Marco Cova, Lastline
marco@lastline.com

About me

- Senior security researcher and a member of the founding team of Lastline, Inc
  - Based in the Old Street, London, office
- Previously, lecturer in Computer Security at the University of Birmingham, UK
- Research interests:
  - Malware analysis
  - Vulnerability analysis

Oracles, Filters, Seeders, Anti Evasions

A PIPELINE FOR SCALABLE AND PRECISE ANALYSIS OF MALWARE

One problem, two dimensions

Precision

- Can we *detect* malware?
- Adversarial setting: modern malware uses a number of techniques to *evade* detection
- Often, detection tools are publicly available/publicly described → testable by malware authors

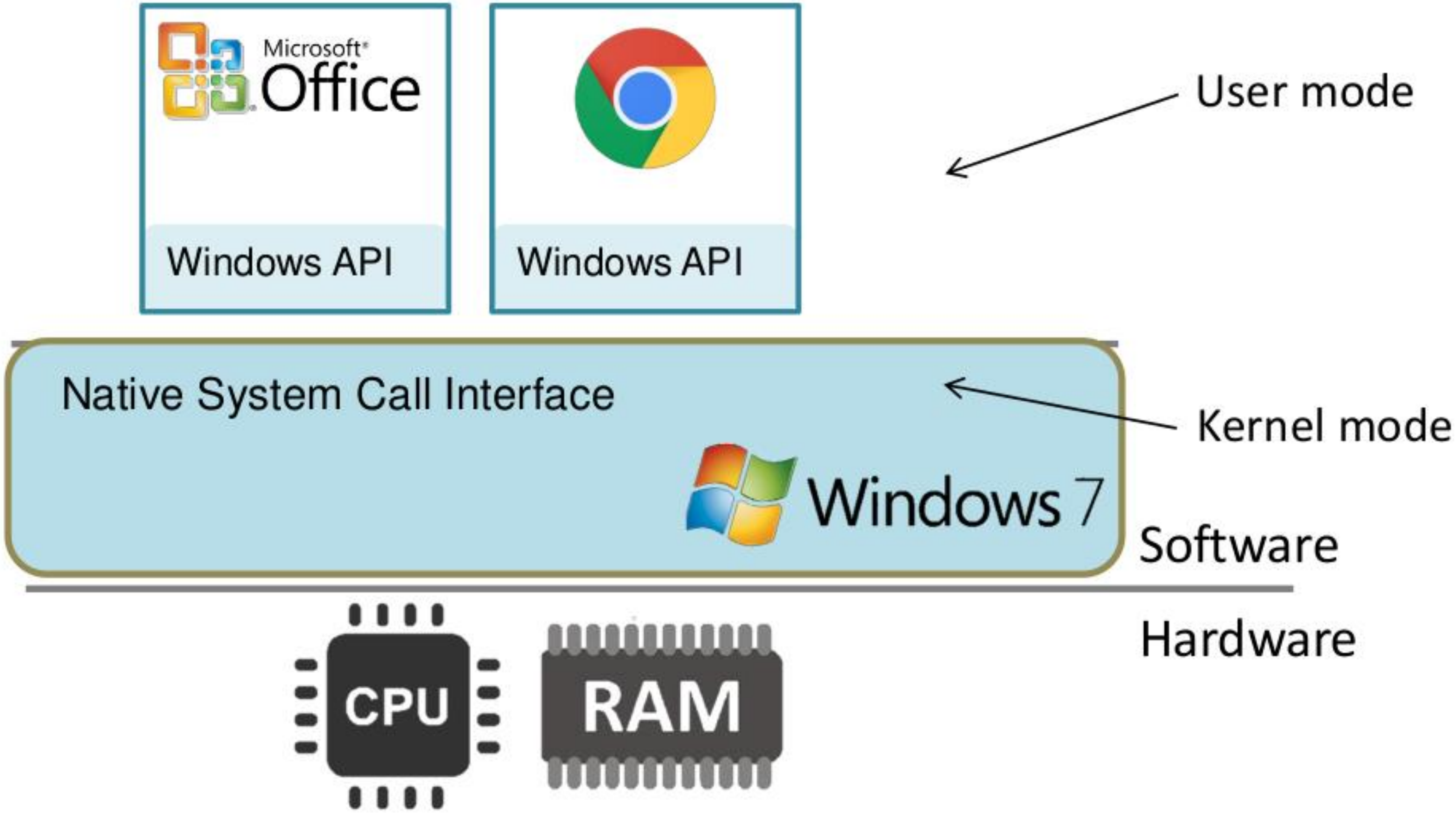
Scalability

- Can we *scale* the detection?
- Challenge: analyze 4+ new pieces of malware *per second*
- Cost, time, infrastructure constraints

Oracle

- Essentially, a classification algorithm for artifacts (web pages, executables, office documents, Android apps, etc.)
  - Input: web page, .exe, .pdf, .apk, …
  - Output: classification (malicious or benign)
- In practice, it is useful to extract and provide users with *evidence* to support classification
  - Exploit detection
  - Deobfuscation results
  - Anything that helps forensics, really

Oracle approaches



Wepawet

- **Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code**
  Marco Cova, Christopher Kruegel, Giovanni Vigna in *Proceedings of the World Wide Web Conference (WWW),* Raleigh, NC, April 2010
- http://wepawet.cs.ucsb.edu
- By the numbers:
  - Number of unique IPs that submitted to Wepawet: 141,463
  - Number of pages visited and analyzed by Wepawet: 67,424,459
  - Number of malicious pages identified as malicious: 2,239,335

Wepawet Features

- Exploit preparation
  - Number of bytes allocated (heap spraying)
  - Number of likely shellcode strings
- Exploit attempt
  - Number of instantiated plugins and ActiveX controls
  - Values of attributes and parameters in method calls
  - Sequences of method calls
- Redirections and cloaking
  - Number and target of redirections
  - Browser personality- and history-based differences
- Obfuscation
  - String definitions/uses
  - Number of dynamic code executions
  - Length of dynamically-executed code

Filter

- If everything goes well, after a while we will have more samples/pages than we can analyze in-depth with your oracle
- Analysis time ranges from a few seconds to a couple of minutes
  - Oracle actually runs the sample
  - Sometimes multiple times (anti-evasion techniques)
  - We may get creative and add sophisticated (= slower/more expensive) analyses (e.g., taint analysis, multi-path execution)
- Do we really need to do this for *every* sample?

Static filtering

- *Quick* identification of samples that can be *safely* discarded
  - For every sample, determine if it is
    likely benign → discard, or
    likely malicious → send to Oracle,
    (can't say → send to Oracle)
- Basis for the classification is typically a set of static features
- Necessarily more imprecise than oracle
  - We only worry about not having false negatives
  - Very tolerant with false positives (consequence: more work for our oracle)

Prophiler

- Filter for malicious web pages
- **Prophiler: a Fast Filter for the Large-Scale Detection of Malicious Web Pages**, Davide Canali, Marco Cova, Christopher Kruegel, Giovanni Vigna in *Proceedings of the International World Wide Web Conference (WWW)*, 2011

Static features

- We define three classes of features (77 in total)
  - HTML (19)
    - source: web page content
  - JavaScript (25)
    - source: web page content
  - URL and host-based (33)
    - source: page URL and URLs included in the content
- One machine learning model for each feature class

Example features

HTML features

- iframe tags, hidden elements, elements with a small area, script elements, embed and object tags, scripts with a wrong filename extension, out-of-place elements, included URLs, scripting content percentage, whitespace percentage, meta refresh tags, double HTML documents, …

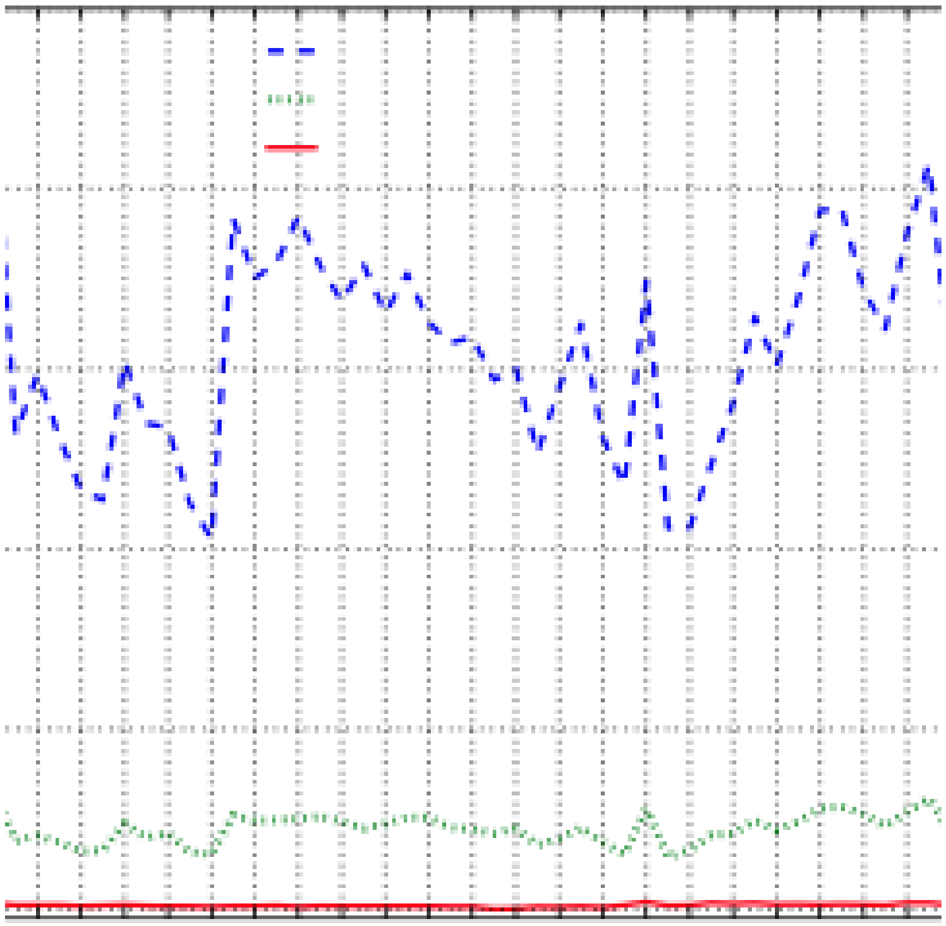
Matches

```
<div style="display:none">
 <iframe src="http://biozavr.ru:8080/index.php" width=104 height=251 >
</iframe></div>
```

```
<body><div id="DivID">
  <script src='a2.jpg'></script>
  <script src='b.jpg'></script>
  <script src='url.jpg'></script>
  <script src='c.jpg'></script>
  <script src='d.jpg'></script>
  <script src='e.jpg'></script>
  <script src='f.jpg'></script>
</body>
```

Evaluation

- Large-scale evaluation of Prophiler
- 60 days of crawling + analysis
- 18,939,908 unlabeled pages
- 14.3% of pages flagged as suspicious and submitted to Wepawet (13.7% FP)
- 85.7% load reduction on Wepawet = saving more than 400 days of analysis!



Seeder

- Great, we now have some spare capacity: we'll process more samples!
- But how do we actually seed our oracle + filter?
  - Public sources (forums, private mailing list, twitter feeds)
  - Users ("crowdsourcing")
  - Sharing agreements
- How do we actually build our own feed?

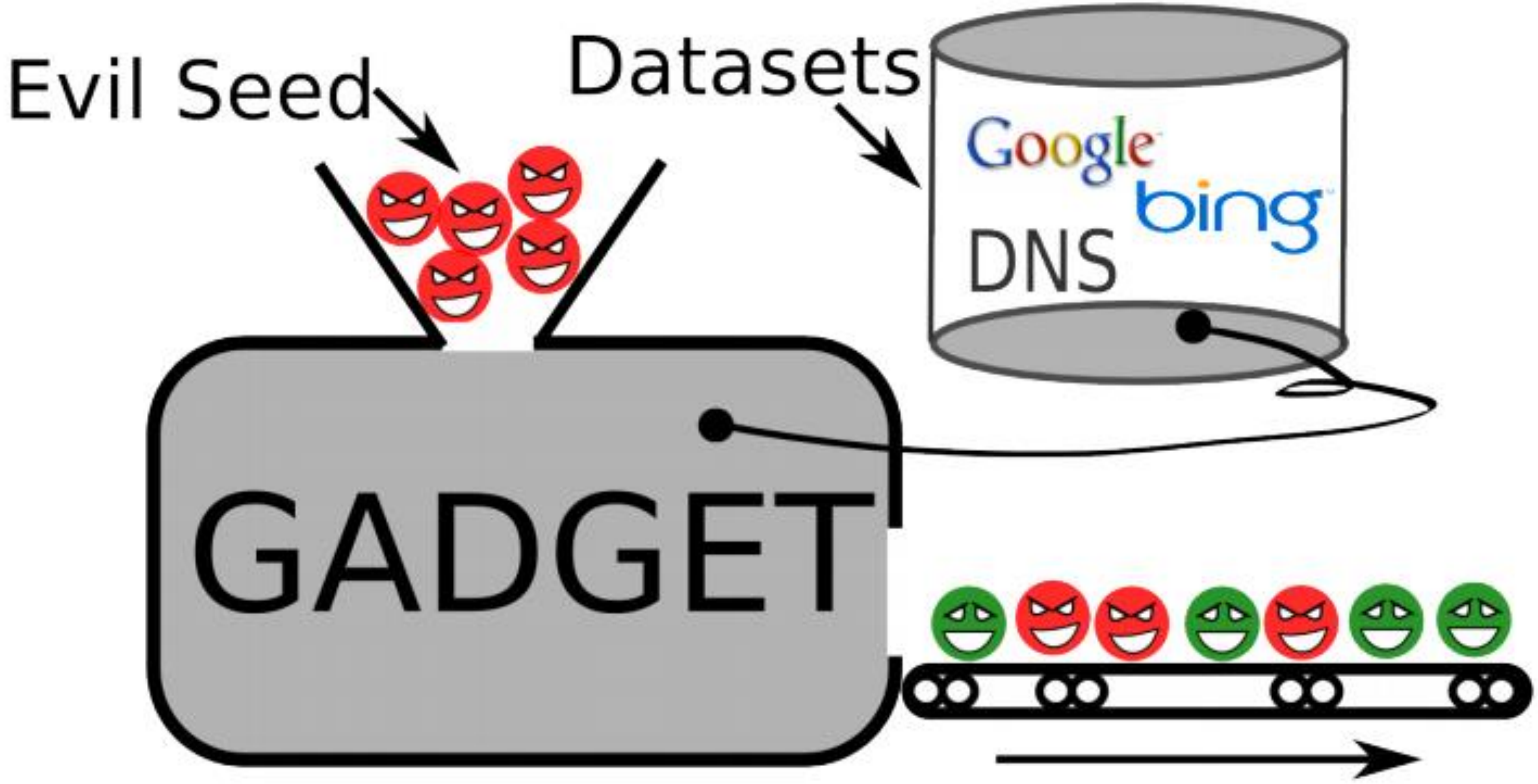
Crawling

- Obvious idea: crawling
  - Crawl the web looking for malicious web pages
  - Detect the exploit and grab the executable being installed on the target machine
  - Analyze the executable
- After filling up a few disks, we realize we actually throw away most of the pages we look at (benign):
  - Problem: *toxicity* of regular crawling is pretty low
  - Observation: crawling only as good as the initial seeds
- Challenge: can we find “better” seeds?
  - Crawl parts of the web that are more likely to contain malicious content

EvilSeed

- Guided search approach to increase toxicity of pages that are crawled
- Inputs: malicious web pages found in the past
- Output: set of (more likely malicious) web pages
- **EVILSEED: A Guided Approach to Finding Malicious Web Pages**, Luca Invernizzi, Stefano Benvenuti, Paolo Milani, Marco Cova, Christopher Kruegel, Giovanni Vigna, in *Proceedings of the IEEE Symposium on Security and Privacy*, 2012

Gadgets



Gadgets

All gadgets share the same structure:

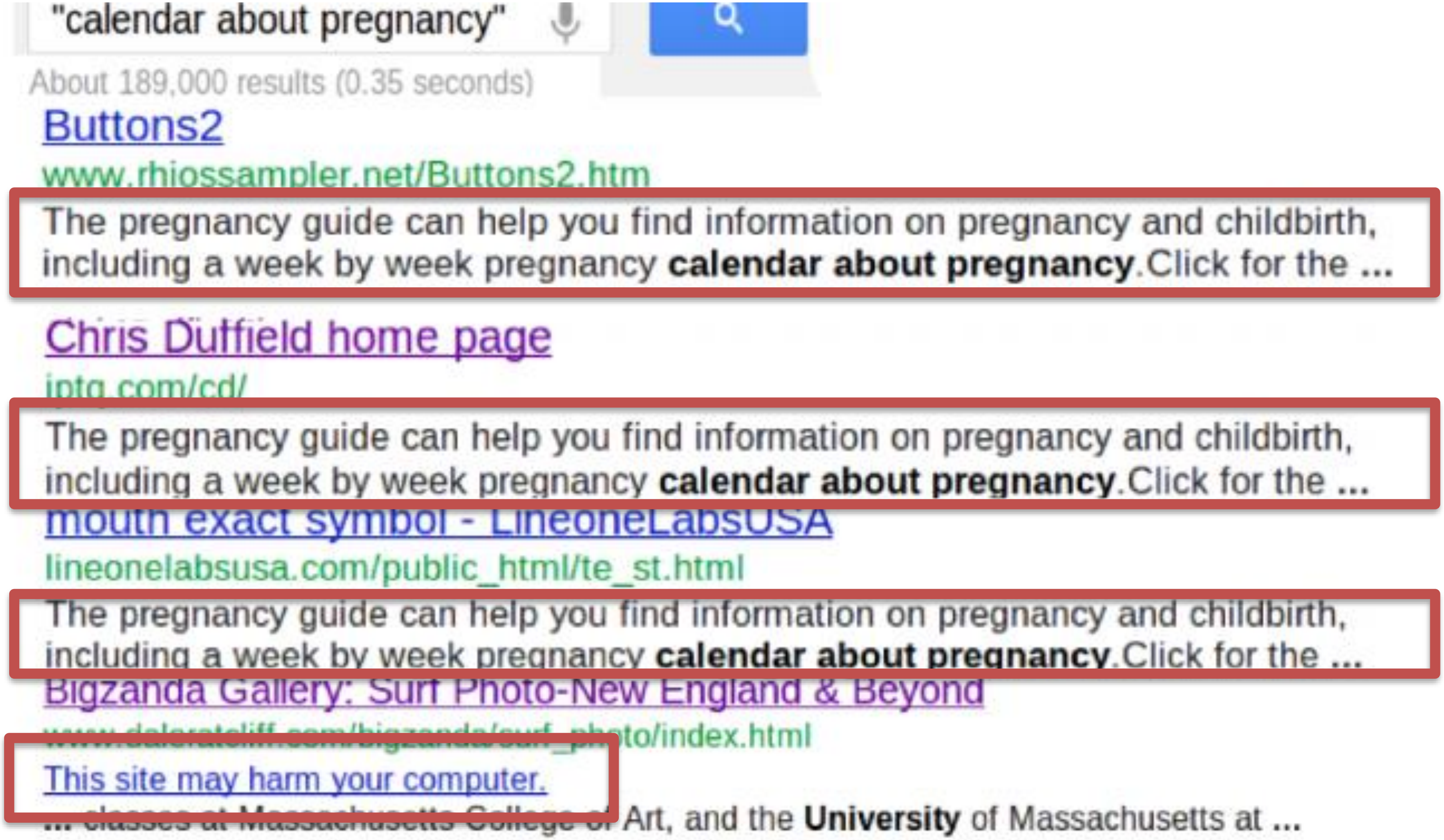
- Method to extract features from a sample set
- Method to search for similar samples leveraging some third-party dataset

- Links gadget (malware hub)
- Content dorks gadget
- SEO gadget
- Domain registration gadget
- DNS queries gadget

Content dork gadget

- Creates “dorks” (signatures) from the content of landing pages (malicious)
  - Assumption: pages that are similar are also likely to be landing pages
- Two methods:
  - n-gram extraction
  - term-extraction (e.g., cnn.com yields: Eurozone recession, gay wedding, Facebook attack, graphic, content)
- We’ll use these signatures to find other pages that are similar

Content dork gadget



"calendar about pregnancy"

About 189,000 results (0.35 seconds)

Buttons2

www.rhiossampler.net/Buttons2.htm

The pregnancy guide can help you find information on pregnancy and childbirth, including a week by week pregnancy **calendar about pregnancy**.Click for the ...

Chris Duffield home page

iptq.com/cd/

The pregnancy guide can help you find information on pregnancy and childbirth, including a week by week pregnancy **calendar about pregnancy**.Click for the ...

mouth exact symbol - LineoneLabsUSA

lineonelabsusa.com/public_html/te_st.html

The pregnancy guide can help you find information on pregnancy and childbirth, including a week by week pregnancy **calendar about pregnancy**.Click for the ...

Bigzanda Gallery: Surf Photo-New England & Beyond

www.daleratcliff.com/bigzanda/surf_photo/index.html

This site may harm your computer.

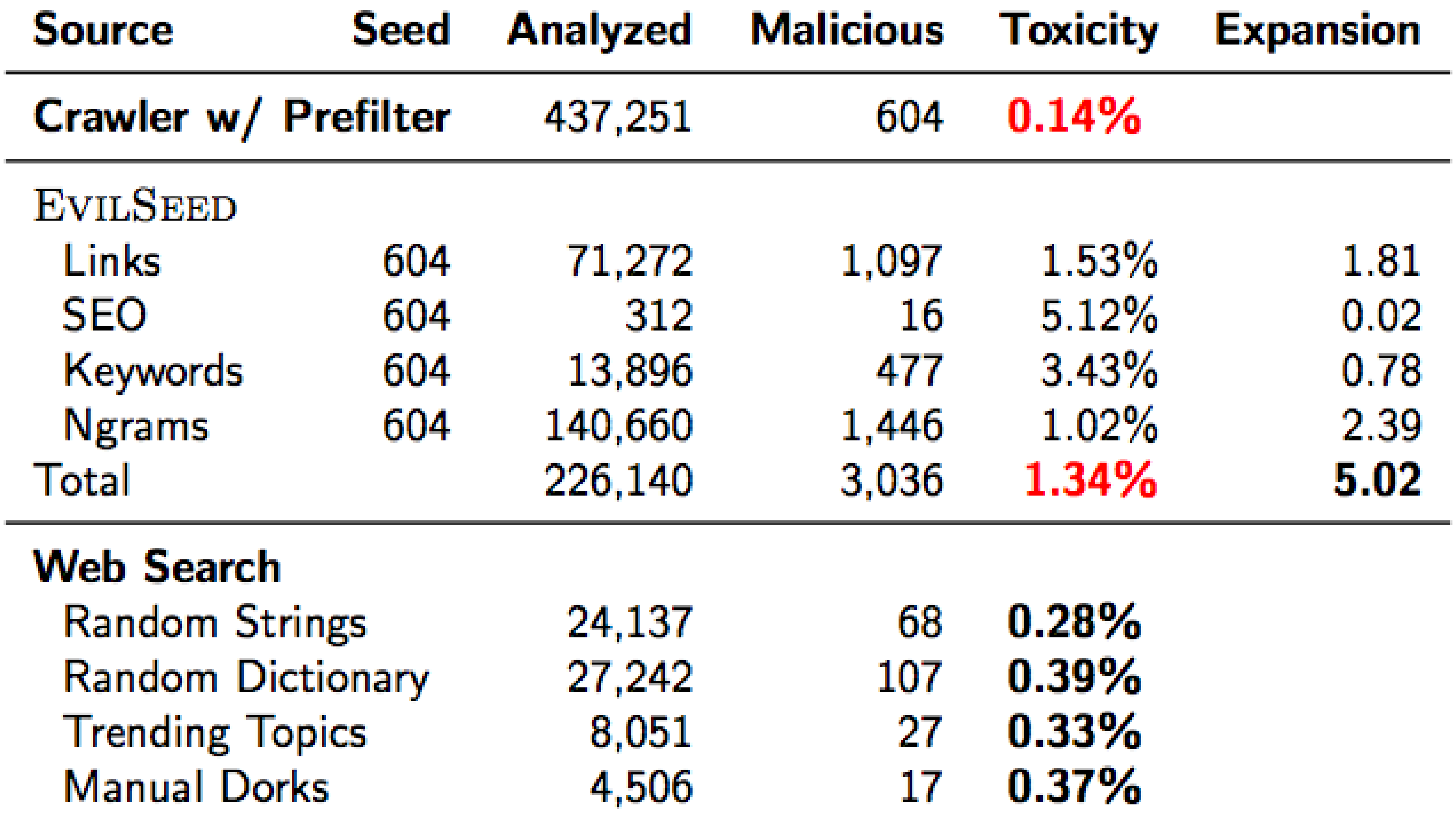
... classes at Massachusetts College of Art, and the **University** of Massachusetts at ...

Evaluation metrics

$$Toxicity = \frac{\text{URLs classified as malicious}}{\text{URLs submitted to the Oracle}}$$

$$Expansion = \frac{\text{malicious URLs found by EvilSeed}}{\text{seed size}}$$

EvilSeed results



Source	Seed	Analyzed	Malicious	Toxicity	Expansion
Crawler w/ Prefilter		437,251	604	**0.14%**	
EVILSEED					
Links	604	71,272	1,097	1.53%	1.81
SEO	604	312	16	5.12%	0.02
Keywords	604	13,896	477	3.43%	0.78
Ngrams	604	140,660	1,446	1.02%	2.39
Total		226,140	3,036	**1.34%**	**5.02**
Web Search					
Random Strings		24,137	68	**0.28%**	
Random Dictionary		27,242	107	**0.39%**	
Trending Topics		8,051	27	**0.33%**	
Manual Dorks		4,506	17	**0.37%**	

Anti evasion

- All is going great: we are processing tons of malicious samples.
- At this point of the story, the bad guys will actively try to evade your system
- Lots of effort in designing evasion techniques
  - Analysis environment detection
  - User detection
  - Stalling
- Challenge: how do we bypass evasion attempts or at least detect if we are being evaded?

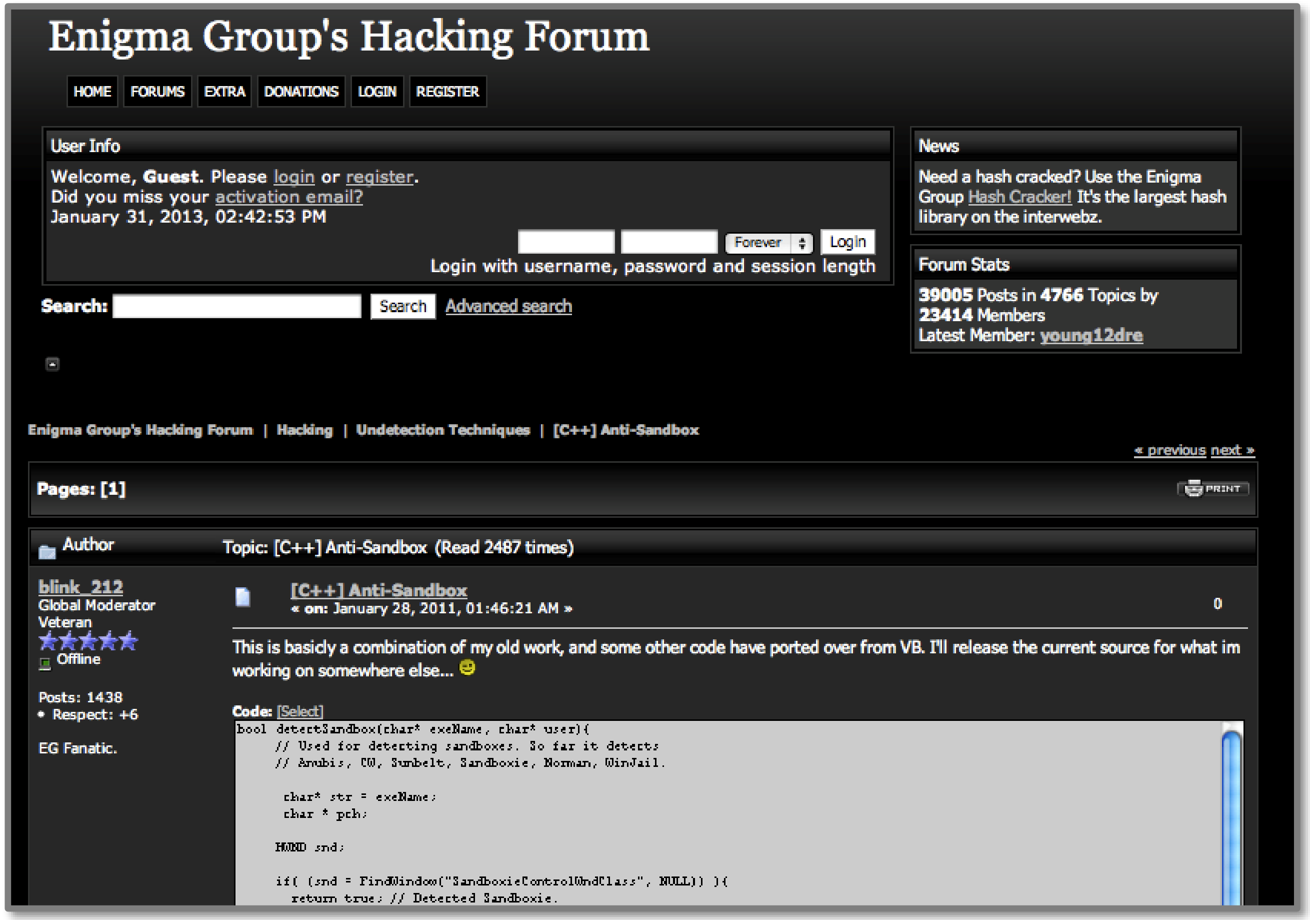
Evasions

Evasion #1: environment check

Is there anything in the environment that makes it unusual or unexpected?

- Unexpected DLLs or applications
- Recurring product IDs/serial numbers
  - HKLM\SYSTEM\CURRENTCONTROLSET\SERVICES\DISK\ENUM
- Hardware configs
  - GlobalMemoryStatus
  - DeviceIoControl (IOCTL_STORAGE_QUERY_PROPERTY)
  - NtOpenKey (Hardware\Description\System\CentralProcessor\0)

Evasion #1: environment check



Enigma Group's Hacking Forum

User Info

Welcome, **Guest**. Please login or register.
Did you miss your activation email?
January 31, 2013, 02:42:53 PM

Login with username, password and session length

Search:

News

Need a hash cracked? Use the Enigma Group Hash Cracker! It's the largest hash library on the interwebz.

Forum Stats

39005 Posts in **4766** Topics by **23414** Members
Latest Member: young12dre

Enigma Group's Hacking Forum | Hacking | Undetection Techniques | [C++] Anti-Sandbox

Pages: [1]

Author | Topic: [C++] Anti-Sandbox (Read 2487 times)

blink_212
Global Moderator
Veteran
★★★★★
Offline

Posts: 1438
• Respect: +6

EG Fanatic.

[C++] Anti-Sandbox
« **on:** January 28, 2011, 01:46:21 AM »

0

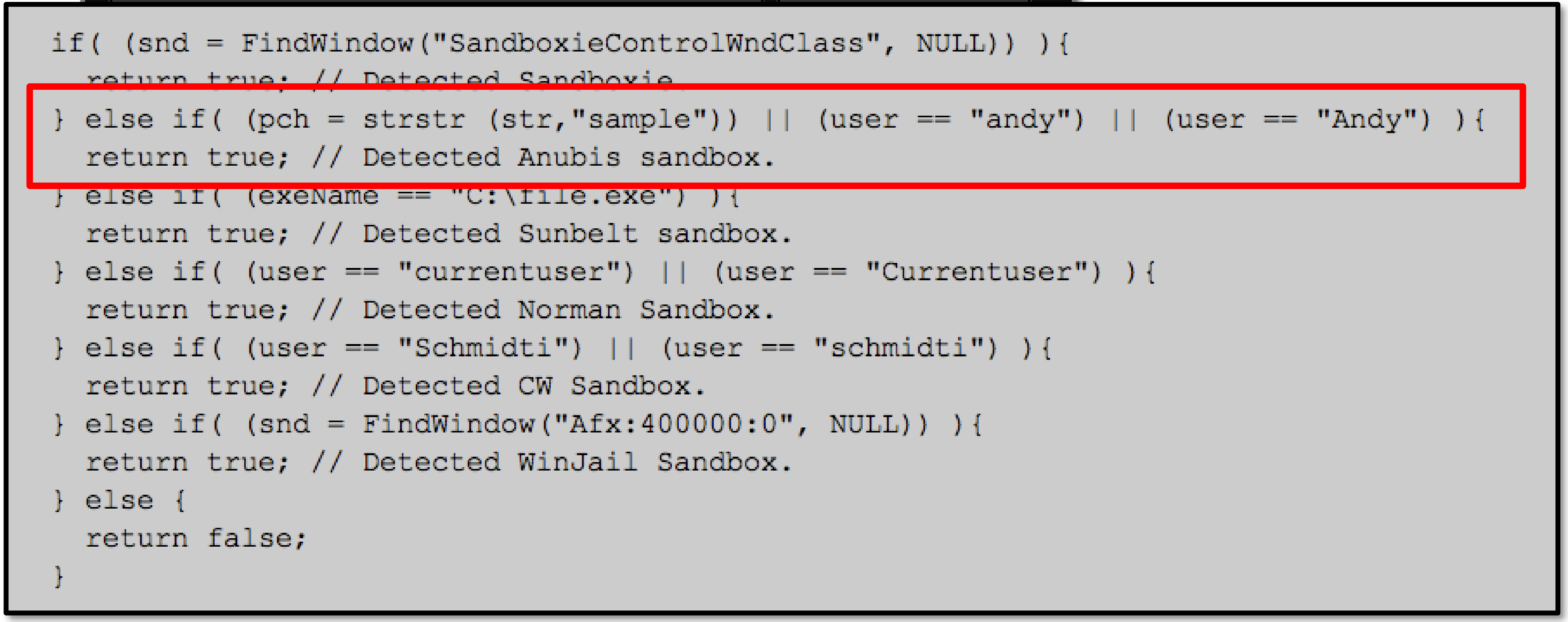
This is basicly a combination of my old work, and some other code have ported over from VB. I'll release the current source for what im working on somewhere else... ☺

Code: [Select]

```
bool detectSandbox(char* exeName, char* user){
     // Used for detecting sandboxes. So far it detects
     // Anubis, CW, Sunbelt, Sandboxie, Norman, WinJail.

      char* str = exeName;
      char * pch;

     HWND snd;

     if( (snd = FindWindow("SandboxieControlWndClass", NULL)) ){
       return true; // Detected Sandboxie.
```

Evasion #1: environment check

Enigma Group's Hacking Forum


```
if( (snd = FindWindow("SandboxieControlWndClass", NULL)) ){
  return true; // Detected Sandboxie.
} else if( (pch = strstr (str,"sample")) || (user == "andy") || (user == "Andy") ){
  return true; // Detected Anubis sandbox.
} else if( (exeName == "C:\file.exe") ){
  return true; // Detected Sunbelt sandbox.
} else if( (user == "currentuser") || (user == "Currentuser") ){
  return true; // Detected Norman Sandbox.
} else if( (user == "Schmidti") || (user == "schmidti") ){
  return true; // Detected CW Sandbox.
} else if( (snd = FindWindow("Afx:400000:0", NULL)) ){
  return true; // Detected WinJail Sandbox.
} else {
  return false;
}
```

Evasion #2: stalling and hiding

Make the execution slow so that the actual malicious behavior occurs after the analysis has (likely) terminated

- In practice, stall the analysis for a few minutes
- Naive implementation

```
push 2000000h
call Sleep
```

Evasion #2: stalling and hiding

Anti-sleep-acceleration

– introduce a race condition that involves sleeping

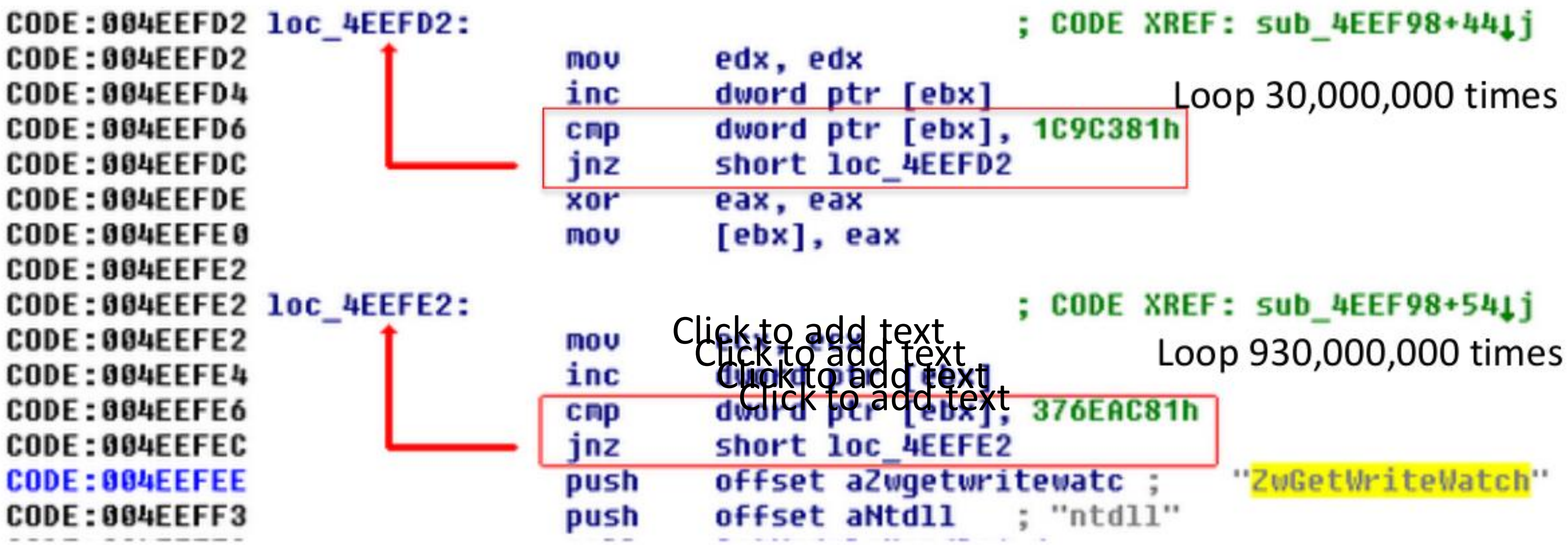
- Sample creates two threads

1. Sleep() + NtTerminateProcess()
2. decrypts and runs payload

- Another variation

1. Sleep() + DeleteFileW(<name>.bat)
2. start <name>.bat file

Evasion #2: stalling and hiding



```
CODE:004EEFD2 loc_4EEFD2:                              ; CODE XREF: sub_4EEF98+44↓j
CODE:004EEFD2                 mov     edx, edx
CODE:004EEFD4                 inc     dword ptr [ebx]
CODE:004EEFD6                 cmp     dword ptr [ebx], 1C9C381h
CODE:004EEFDC                 jnz     short loc_4EEFD2
CODE:004EEFDE                 xor     eax, eax
CODE:004EEFE0                 mov     [ebx], eax
CODE:004EEFE2
CODE:004EEFE2 loc_4EEFE2:                              ; CODE XREF: sub_4EEF98+54↓j
CODE:004EEFE2                 mov     edx, edx
CODE:004EEFE4                 inc     dword ptr [ebx]
CODE:004EEFE6                 cmp     dword ptr [ebx], 376EAC81h
CODE:004EEFEC                 jnz     short loc_4EEFE2
CODE:004EEFEE                 push    offset aZwgetwritewatc ;    "ZwGetWriteWatch"
CODE:004EEFF3                 push    offset aNtdll    ; "ntdll"
```

Loop 30,000,000 times

Loop 930,000,000 times

Click to add text

Click to add text
Click to add text
Click to add text

Stalling like Rombertik

More at http://labs.lastline.com/exposing-rombertik-turning- the-tables-on-evasive-malware

Evasion #3: human detection

- Is there a human behind the keyboard?

```
var X=this.mouseX;
var Y=this.mouseY;
for (;;) {
 if ((this.mouseX!=X)||
     (this.mouseY!=Y)){
        break;
 }
}
do_evil_stuff();
```

Evasion #3: human detection

- And is she not an analyst/reverser?

```
if
(!!window._IE_DEVTOOLBAR_CONSOLE_COMMAND_LINE)
  return; /* don’t run the exploit */
```

HASTEN

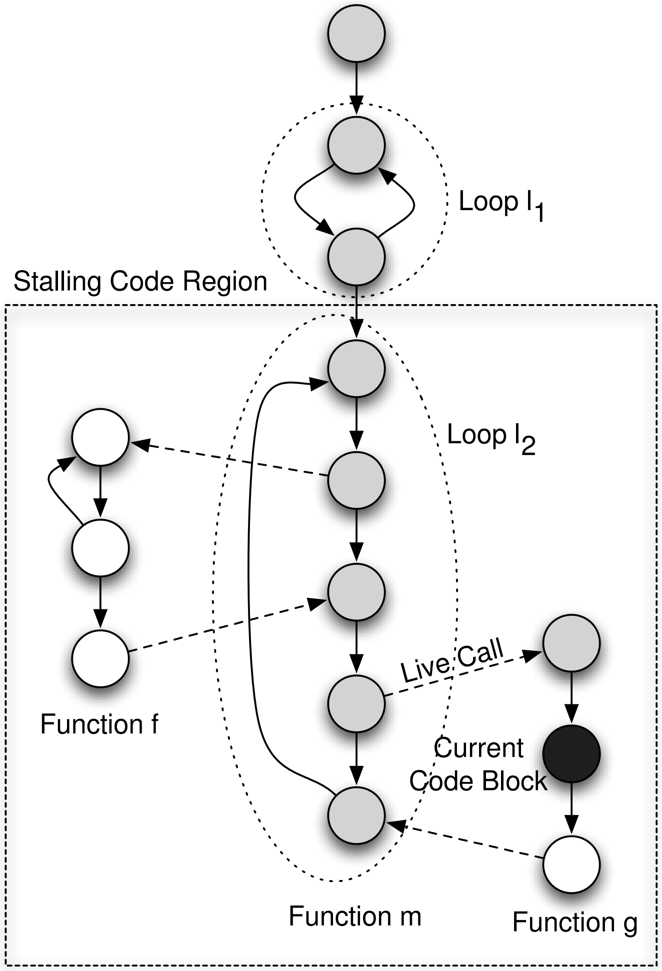
- Approach to detect and mitigate malicious stalling code
- **The power of procrastination: detection and mitigation of execution-stalling malicious code**, Clemens Kolbitsch, Engin Kirda, Christopher Kruegel, Giovanni Vigna, in *Proceedings of the ACM conference on Computer and Communications Security*, 2011

Bypass stalling

- Mitigate stalling loops
  1. Detect that program does not make progress
  2. Passive mode
     - Find loop that is currently executing
     - Reduce logging for this loop (until exit)
  3. Active mode
     - When reduced logging is not sufficient
     - Actively interrupt loop
- Progress checks
  – Based on system calls:
    too many failures, too few, always the same, …

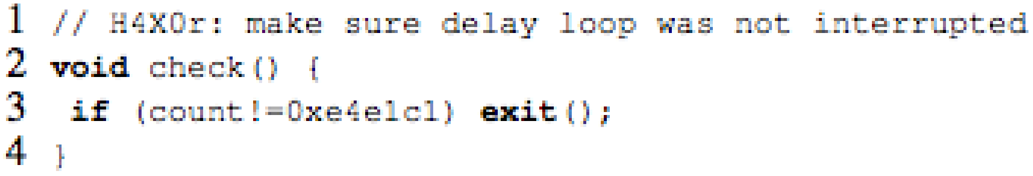
Passive Mode

- Finding code blocks (white list) for which logging should be reduced
  - Build dynamic control flow graph
  - Run loop detection algorithm
  - Identify live blocks and call edges
  - Identify first (closest) *active* loop (loop still in progress)
  - Mark all regions reachable from this loop



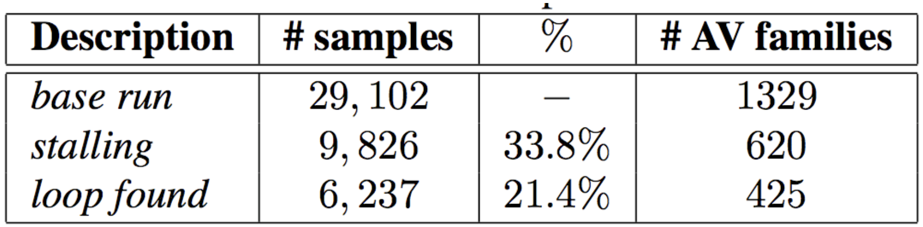
Active Mode

- Interrupt loop
  - Find conditional jump that leads out of white-listed region
  - Simply invert it the next time control flow passes by
- Problem
  - Program might later use variables that were written by loop but that do not have the proper value and fail



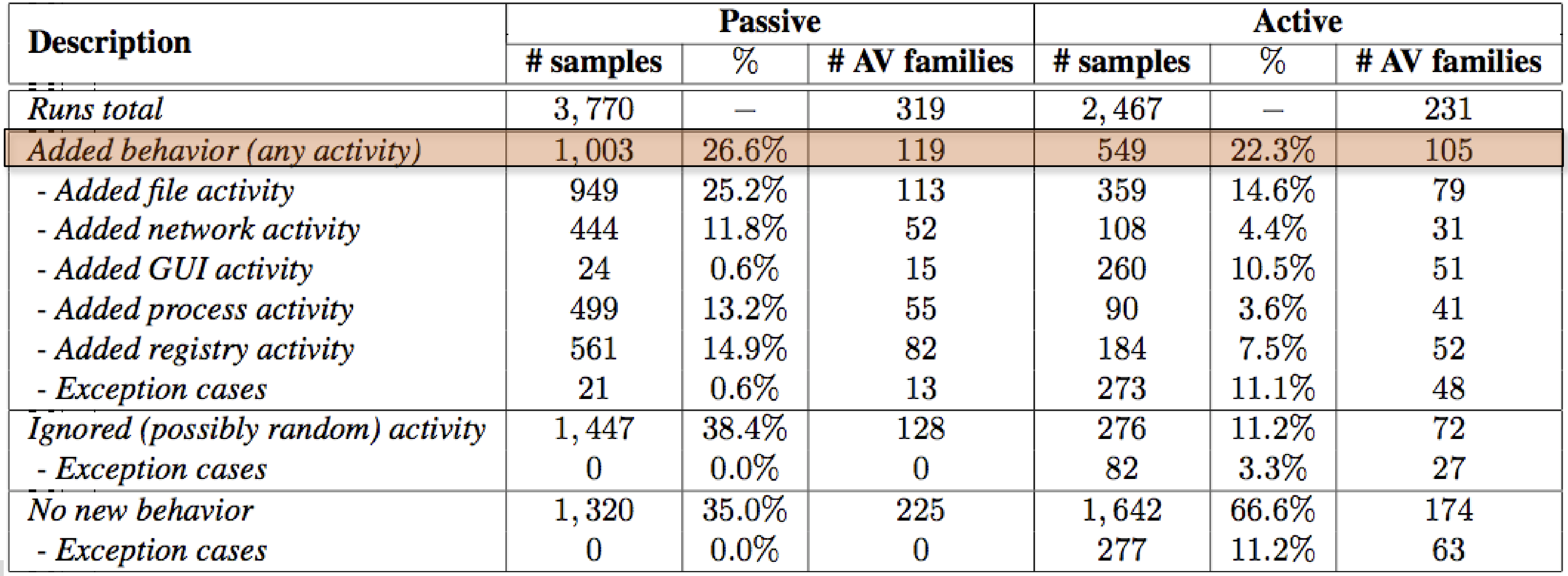
```
// H4X0r: make sure delay loop was not interrupted
void check() {
 if (count!=0xe4e1c1) exit();
}
```

Experimental Results



Description	# samples	%	# AV families
base run	29, 102	–	1329
stalling	9, 826	33.8%	620
loop found	6, 237	21.4%	425

- 1,552 / 6,237 stalling samples reveal additional behavior
- At least 543 had obvious signs of malicious (deliberate) stalling



Description	Passive			Active		
	# samples	%	# AV families	# samples	%	# AV families
Runs total	3, 770	–	319	2, 467	–	231
Added behavior (any activity)	1, 003	26.6%	119	549	22.3%	105
- Added file activity	949	25.2%	113	359	14.6%	79
- Added network activity	444	11.8%	52	108	4.4%	31
- Added GUI activity	24	0.6%	15	260	10.5%	51
- Added process activity	499	13.2%	55	90	3.6%	41
- Added registry activity	561	14.9%	82	184	7.5%	52
- Exception cases	21	0.6%	13	273	11.1%	48
Ignored (possibly random) activity	1, 447	38.4%	128	276	11.2%	72
- Exception cases	0	0.0%	0	82	3.3%	27
No new behavior	1, 320	35.0%	225	1, 642	66.6%	174
- Exception cases	0	0.0%	0	277	11.2%	63

Conclusions

- Malware is key component in many security threats on the Internet
- Automated analysis of malicious code faces a number of challenges
  - Evasion is one critical challenge
  - Scalability of the analysis
- Pipeline of techniques to achieve scalability and precision
  - Different approaches, methods at each step

marco_cova
marco@lastline.com

QUESTIONS?

lastline