Web-Based Malware

Jason Ganzhorn

5-12-2010

Background

- A large number of transactions take place over the Internet
  - Shopping
  - Communication
  - Browse News
- It's likely that you perform some of these transactions as well.

Scenario (Setup)

- You like to read articles about the latest developments in gadgetry.
- Many blogs allows readers to comment on articles, and typically these comments will load along with the page when you load up the article.
- You also use IE 6 because you don’t believe in newfangled browsers (but you still like gadgets).

Scenario (Security Breaches)

- A fairly popular tech blog known as tech.gadget is mostly funded by advertising and pulls ads from a common online advertising agency.
- A hacker pays the agency to distribute an ad containing malicious code to test a package of exploits on viewers' computers.

Scenario (Download Stage)

- It so happens that your IE 6 has one of the targeted vulnerabilities.
- You search for tech news, find a cool article at tech.gadget, and then read it, ignoring the hacker's ad.
- However, one of the ad's exploits kicks in and a piece of malware is transferred to your machine.

Scenario (Finale)

- Now there is a piece of malicious software running on your computer.
- This could be:
  - a keylogger
  - a bot
  - a browsing history tracker
- Obviously, this is not good.

So What?

- None of us are more than a few minor version numbers behind (and almost certainly not running IE 6), so why is this a big deal?
- Many people don’t understand why security updates are important. Why update if I don’t see something broken?

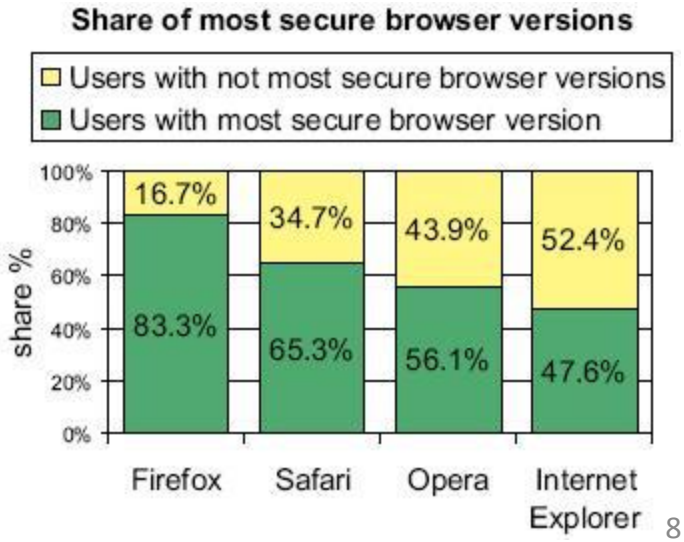
The Update Problem

As you can see from these charts, some people simply do not update their browsers.

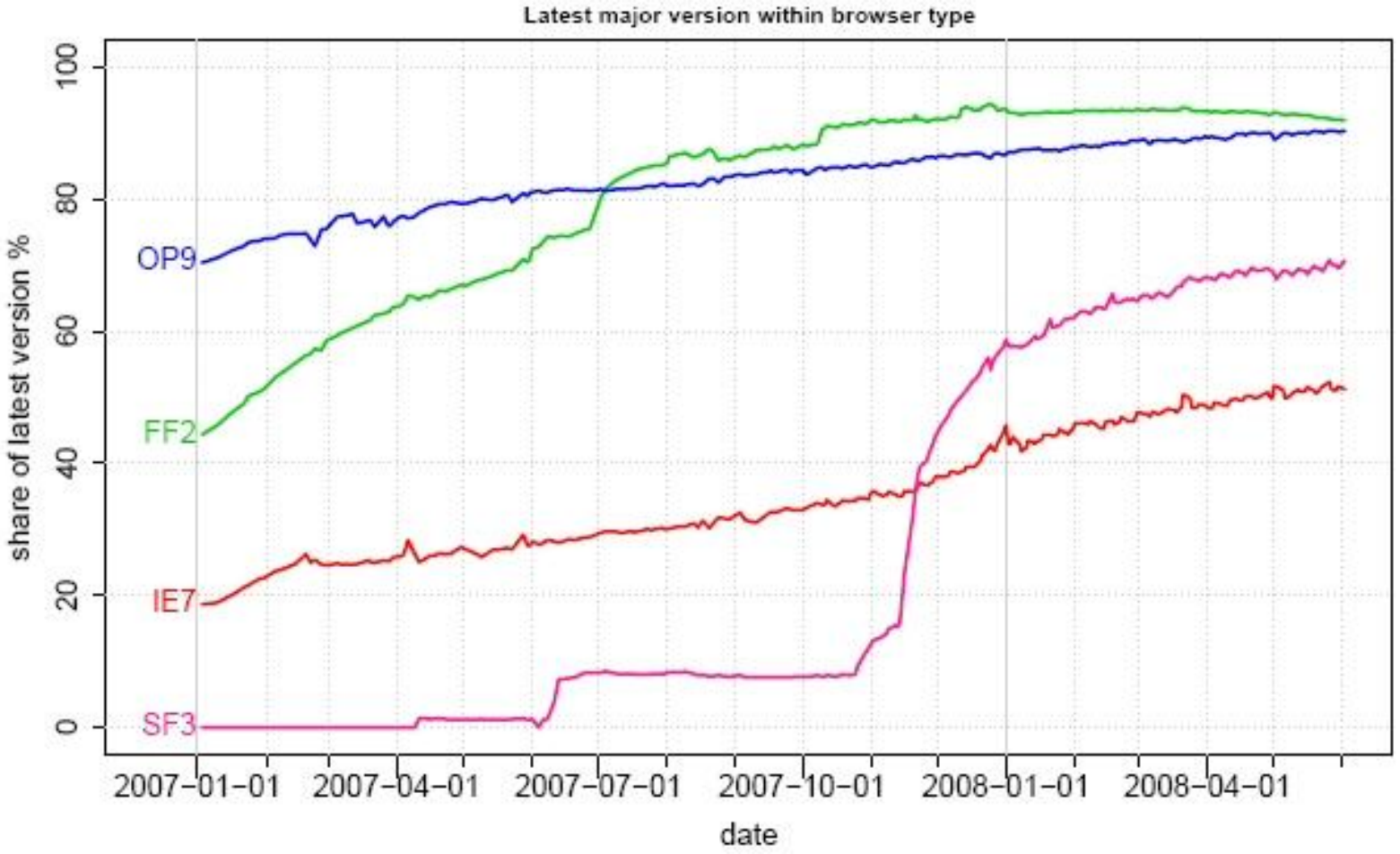
Why not?

- Ignorance – not everyone is tech savvy enough to know that security patches are a good thing.
- Difficulty – for a large company, a major version upgrade can be quite the IT hassle.

Latest Major Version	IE7	FF2	SF3	OP9	Total
Release date of latest major version	2006-10-18	2006-10-24	2007-10-26	2006-06-20	
Share of latest major version within browser type	52.5%	92.2%	70.2%	90.1%	59.1%
Number of latest major version in million (worldwide)	579	209	34	10	**832**



Slow Update Adoption



Charts on last two slides from: http://www.techzoom.net/publications/insecurity-iceberg/

What to Do?

- There’s a lot of people out there with potentially vulnerable browsers.
- It would be nice if there were a way to identify sites that could potentially infect you with malicious software before you just blindly click on them from search results, without having to update your browser.
- It turns out that some people from Google have been looking into just that.

Analysis of Web-based Malware

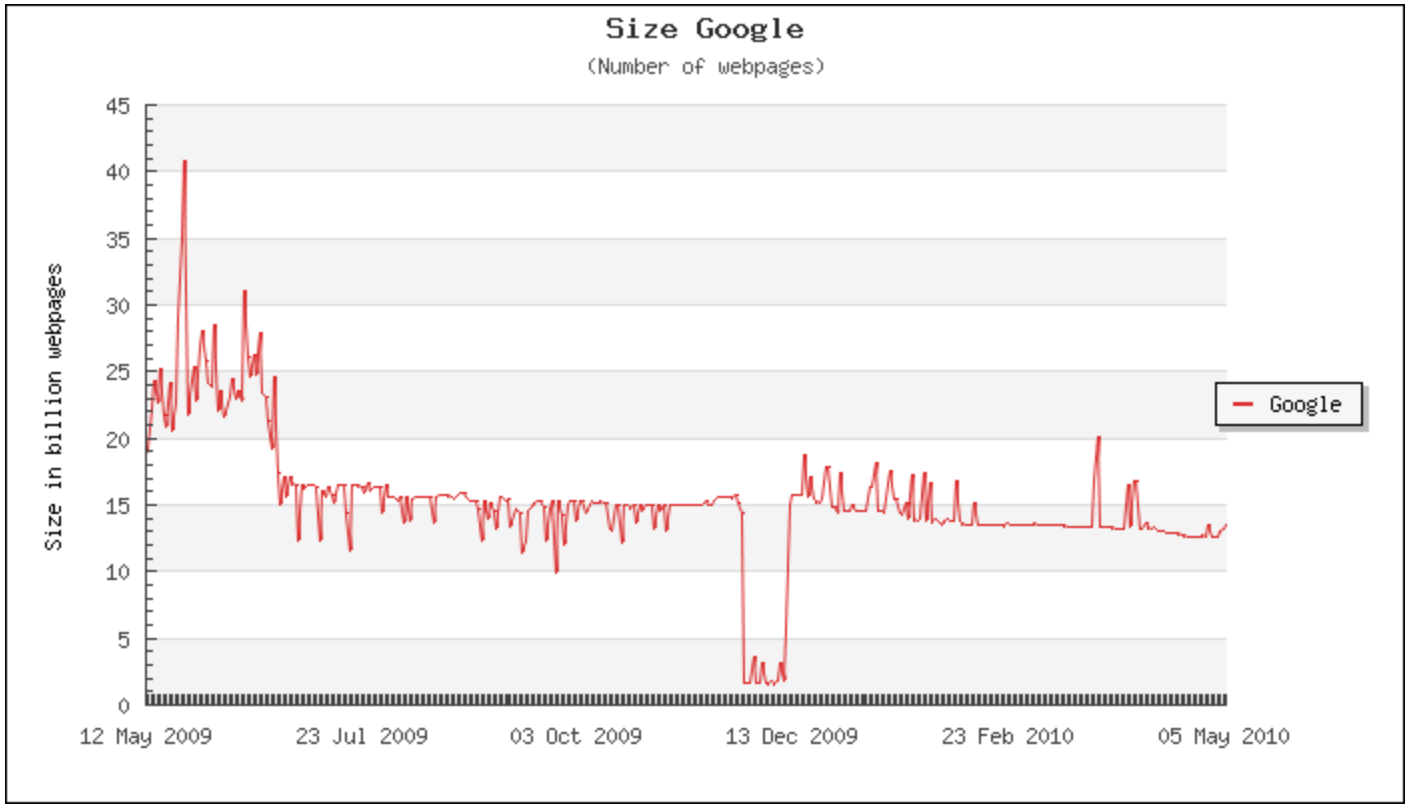
THE GHOST IN THE BROWSER

Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu

The Goal

- Google has an extensive repository of pages on the web.
- Utilizing these resources, the researchers are trying to identify which of those pages could potentially be malicious.

Size of the Google Index



Go here for more information on how the index size was estimated. Note that Google does not appear to officially release the size of their search index any more.

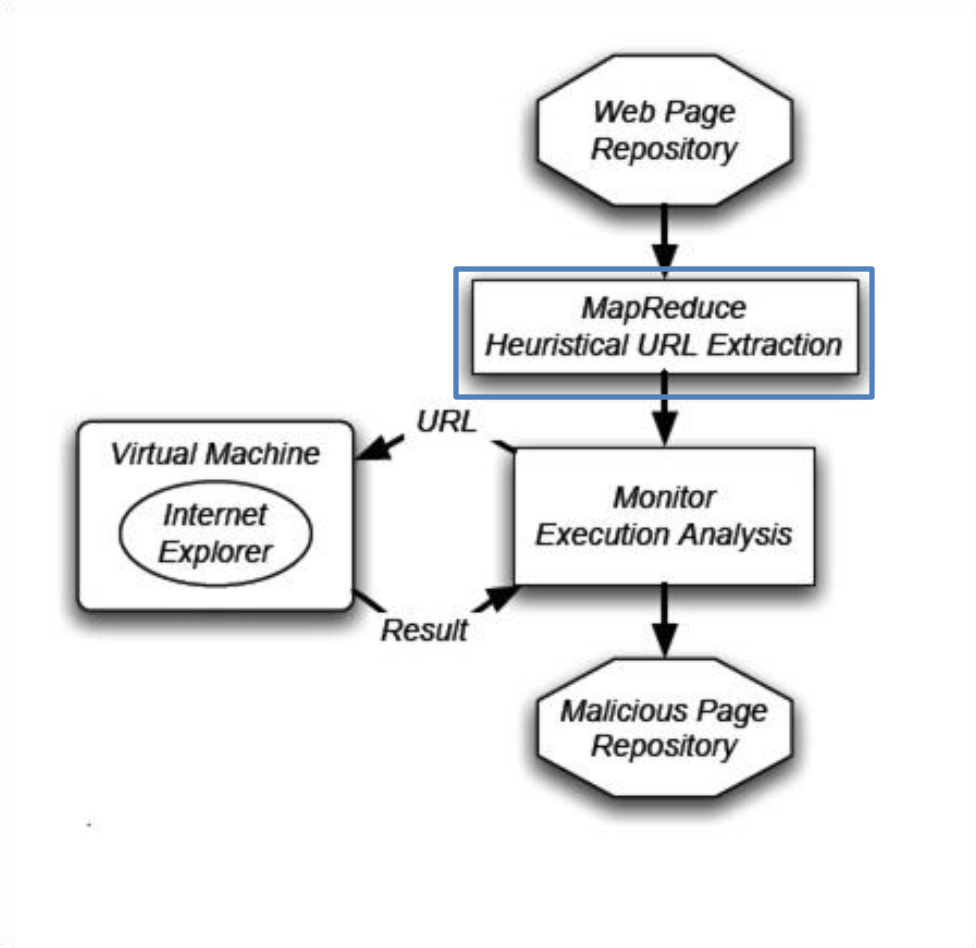
Potential Problems

- Impact of false positives – if a legitimate website is marked as a potential distributor of malware, that could be bad for its business.
- Sheer number of pages on the web.
  - Netcraft reports in their April 2010 survey that they received responses from 205,368,103 sites.
  - The Google index has somewhere around 15 billion pages.

Dealing with the Web

- There are a lot of pages on the web. How can this number be pared down to something that is reasonable to examine?
- The Google researchers apply simple heuristics to each page to determine whether a page attempts to exploit a web browser.
- Pages that test positive under these heuristics are then examined more closely.

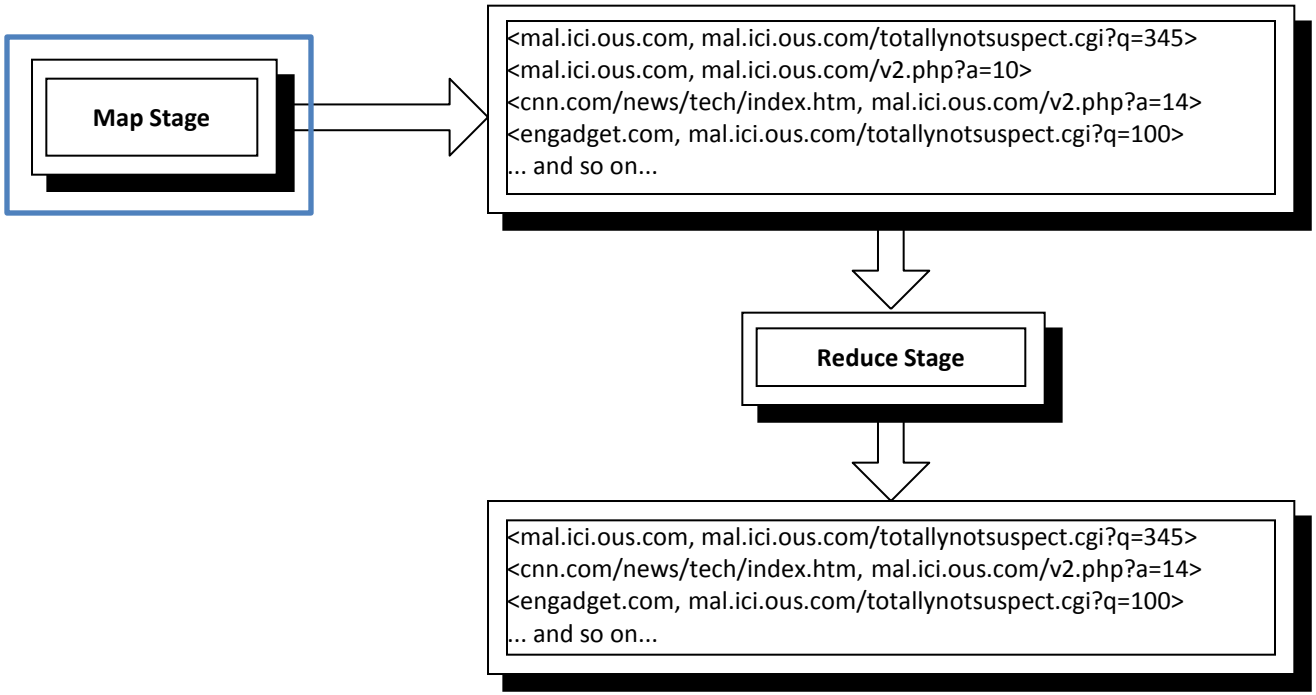
Detection Architecture Diagram



MapReduce Description

- A programming model that operates in two stages.
  - *Map* stage: a sequence of key-value pairs is read as input and a sequence of intermediate key-value pairs is output
  - *Reduce* stage: All intermediate values associated with the same intermediate key are merged and output as a final sequence of key-value pairs.

MapReduce Process in this Paper



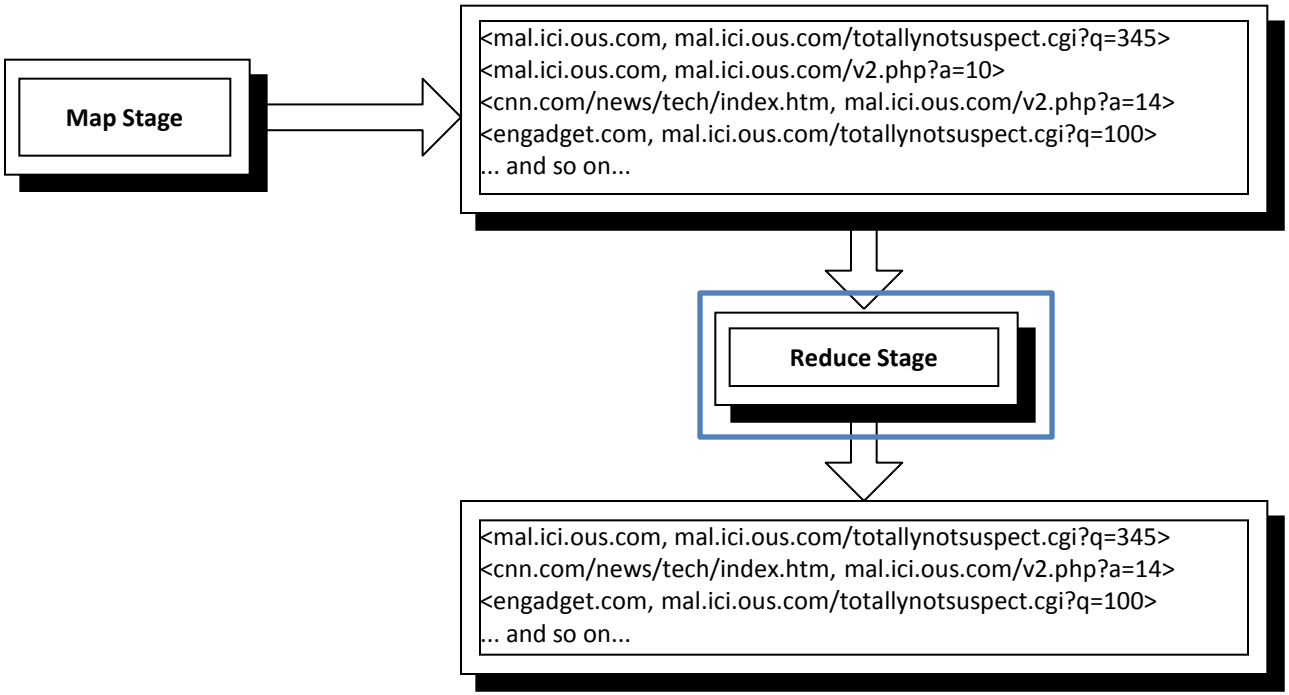
Map Stage

- The *Map* stage is run on all crawled web pages.
- The URL of each analyzed web page is a key.
- The HTML in each page is parsed; links in known suspicious elements such as iframes pointing to malware-distributing hosts are stored as values.

Map Stage (cont'd.)

- Another heuristic used to identify suspicious links at this stage relies on detection of abnormalities such as heavy obfuscation.
- On completion, this stage yields an intermediate list of URLs as keys and all links from that page to possibly malicious URLs as values.

MapReduce Process in this Paper



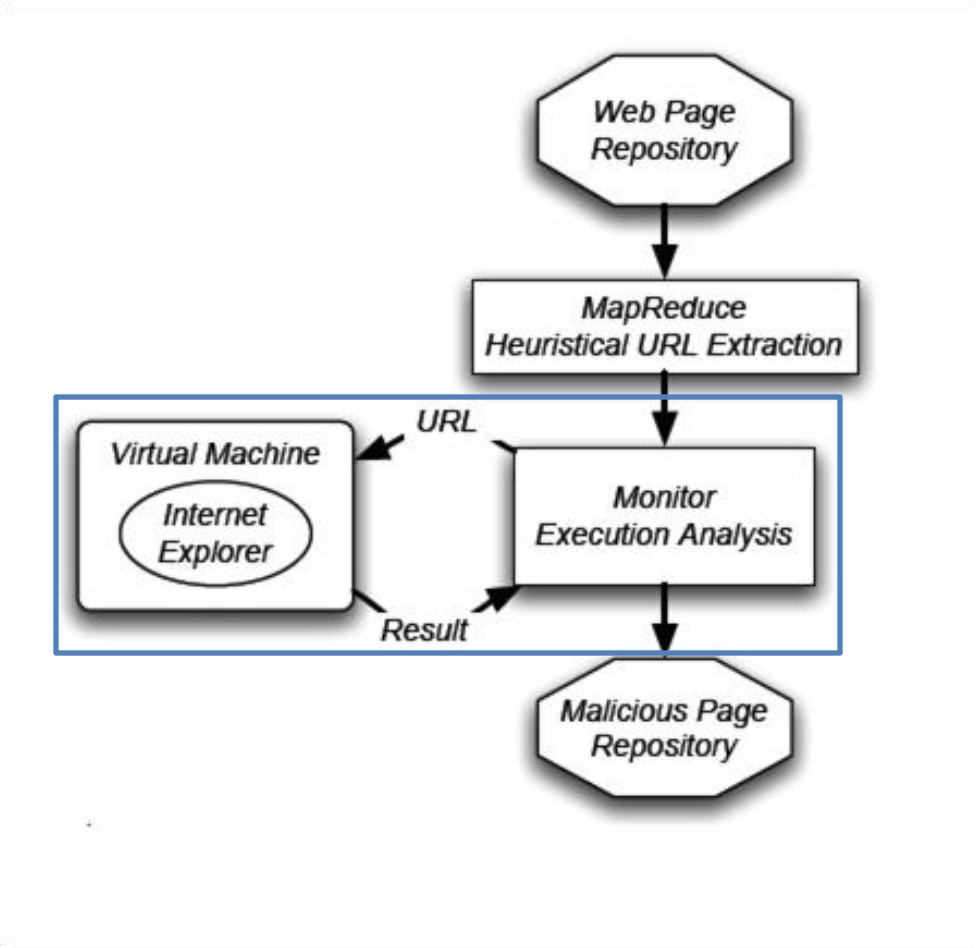
Reduce Stage

- The *Reduce* stage is run on all the intermediate key-value pairs.
- It is very simple – all but the first intermediate value is discarded for each intermediate key.
- On completion, this stage yields a list of potentially malicious URLs and an example of a possibly suspect link from each of them.

MapReduce Results

- This MapReduce process pares the number of web pages to process from several billion to a few million.
- The number of web pages to process can in fact be reduced farther, using a second MapReduce step to sample by site instead of by page.

Detection Architecture Diagram



Exploit Confirmation

- Even after the MapReduce step, there are still several million pages with possible links to exploits.
- How to confirm whether these pages actually cause a web browser exploit?

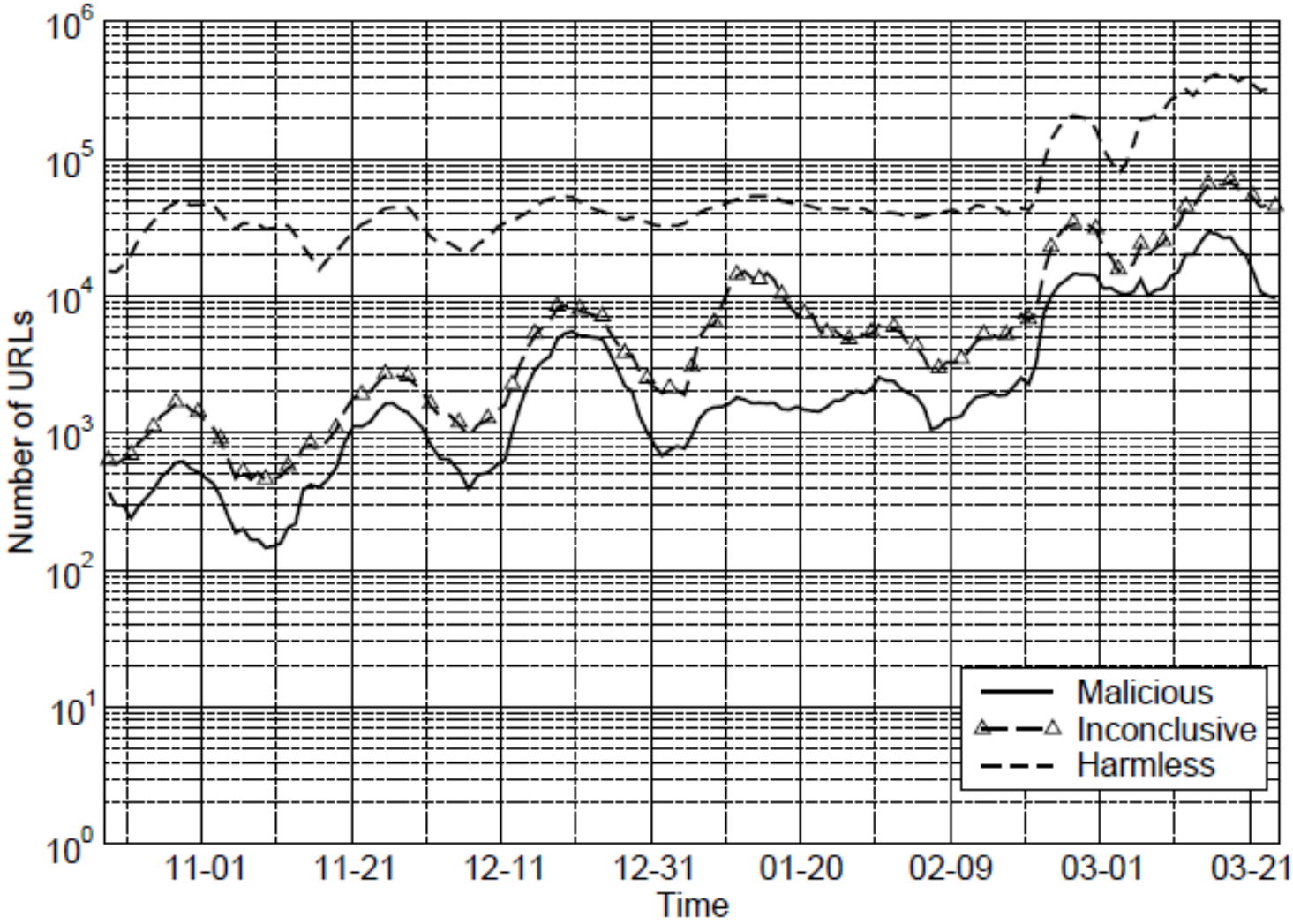
Exploit Testing

- Each URL is fed to a copy of Internet Explorer running in a virtual machine.
- All HTTP fetches and state changes in the VM can be tracked. These state changes include:
  - New process startup
  - Registry changes
  - File system changes

Potential Exploit Scoring

- Each recorded component is scored to provide an overall score.
  - Example: each HTTP fetch is classified using a number of different anti-virus engines.
- Each individual score is then summed up to form an overall score for the analysis
- If the majority of URLs on a site are malicious, some or all of the site might be labeled as harmful when shown as a search result.

Evaluation – Throughput



Evaluation - Throughput

- This analysis originally processed 50,000 unique URLs per day.
  - Optimizations increased this rate to 300,000 per day.
- In-depth analysis of 4.5 million URLs at the time of writing.
  - 450,000 engaged in drive-by-downloads.
  - 700,000 more seemed malicious but with lower confidence.

Content Control and Dependencies

- How can you lose control of content on your page?
  - Web server insecurity – if an adversary can take control of the server, they can modify its content, such as its templating system
  - User-contributed content – poor sanitization of input can lead to injected code. The researchers discovered several bulletin boards that allowed the insertion of arbitrary HTML.

Content Control and Dependencies

- Advertising – “sub-letting” trust issues
  - Here‘s a real-life example:
  - A banner advertisement from a large American advertising company was delivered in the form of JS that generated more JS.
  - This new JS chained through another large American agency and then a smaller one that apparently used geo-targeting.
  - The geo-targeted ad resulted in an iframe pointing to a Russian advertising company.
  - That iframe requested encrypted JS from an IP address that attempted several exploits, some of which were successful.
  - Trust is not transitive.

Content Control and Dependencies

- Web masters sometimes include external JS or iframes to provide additional functionality.
  - The paper gives an example of a page that linked to a free statistics counter.
  - The counter worked benignly for about four years.
  - Then the linked JS was changed to instead try to exploit every visitor to pages linking to the supposed counter.
  - Another trust issue – even if you trust the original content provider, you have no control over what happens to the external code you link to.

Exploitation Mechanisms

- Sometimes malicious JS targets specific vulnerabilities.
  - Microsoft Data Access Component vulnerability (required only about 20 lines of JS to reliably launch an arbitrary binary on a vulnerable installation)
  - Microsoft WebViewFolderIcon vulnerability exploited using JS heap spraying techniques

Exploitation Mechanisms

- Exploiting one vulnerability is limiting. Multi-exploit kits are typically more effective.
- An example is the MPack kit produced by Russian crackers.
  - Commercial software ($500 - $1000)
  - Technical support
  - Software vulnerability updates
  - Customized attacks to victim browsers, including IE, Firefox, and Opera.
  - More fascinating details here.

Exploitation Mechanisms

- What if the user has no discoverable exploitable vulnerabilities?
- Fall back on good old social engineering and promise the user content they might find intriguing.
- Example: Offer the user copyrighted video content for “free” and then claim a “codec” is needed to correctly play the video.

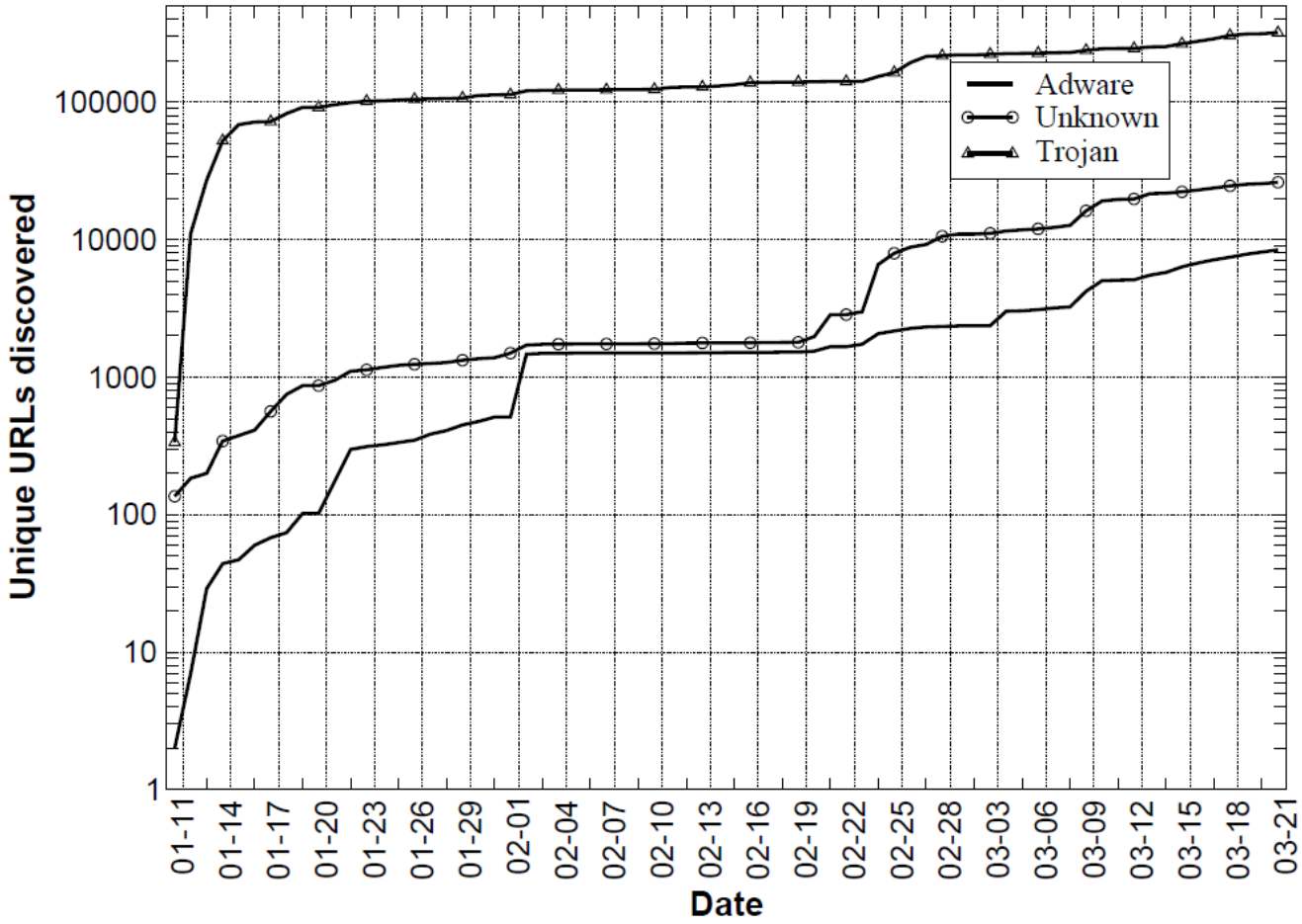
Trends in Malware

- In the arms race between malware generation and detection, some trends have emerged.
- Exploit code is often obfuscated
  - The paper gives an example of a VBScript exploit that was escaped twice using JavaScript escaping.
  - Even some reputable web pages serve obfuscated JavaScript.

Trends in Malware

- The authors also attempted to classify the different types of malware that use the web to deploy. The stated goal was to discover whether web-based malware was being used to construct botnets.
- Their automated analysis seems very rough, with only “Trojan,” “Adware,” and “Unknown/Obfuscated” categories.

Trends in Malware



Trends in Malware

- Most of the examined exploits were hosted on third-party servers and not on the compromised web site.
- Occasionally, all requests to the legitimate site were redirected to a malicious site.
- Many exploits are hosted on multiple servers as well, to minimize the chance of failure.
- A few of the malicious URLs pointed to rapidly changing malware binaries.

Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code

Marco Cova, Christopher Kruegel, Giovanni Vigna

Summary

- The problem is that malicious JavaScript code can be very difficult to identify due to the dynamic nature of JavaScript.
- The paper presents a solution built around anomaly detection with emulation.
- Machine-learning techniques are used to establish profiles of “normal” JavaScript code.

Detection Technique

- Detection is based on finding anomalies which include:
  - Redirection chains.
  - Differences in served JS based on reported browser version.
  - Differences in the JS served to the same IP for consecutive identical requests.
  - Environment preparation (including heap spraying)
  - Exploitation patterns (such as odd plugin-loading behavior)
  - Extensive deobfuscation (one feature to look for is an abnormal amount of dynamic code execution)

Simple Code Obfuscation Example

- This is a very basic example of code obfuscation.
- Real obfuscated code is often polymorphic and dynamically generated.

```
var a="Hello World!";
function MsgBox(msg)
{
    alert(msg+"\n"+a);
}
MsgBox("OK");
```

```
eval(function(p,a,c,k,e,d){e=functio
n(c){return
c};if(!''.replace(/^/,String)){while
(c--
){d[c]=k[c]||c}k=[function(e){return
d[e]}];e=function(){return'\\w+'};c=
1};while(c--
){if(k[c]){p=p.replace(new
RegExp('\\b'+e(c)+'\\b','g'),k[c])}}
return p}('4 0="3 5!";9
2(1){6(1+"\\7"+0)}2("8");',10,10,'a|
msg|MsgBox|Hello|var|World|alert|n|O
K|function'.split('|'),0,{}))
```

Use of Models

- Models are constructs designed to assign a probability score to a feature value, given some established model of “normality.”
  - Example: suppose there are 70 instantiated plugins/ActiveX controls on a page. The “normal” number has been established to be roughly 4-5, so the model assigns a very low probability that 70 is a normal value.

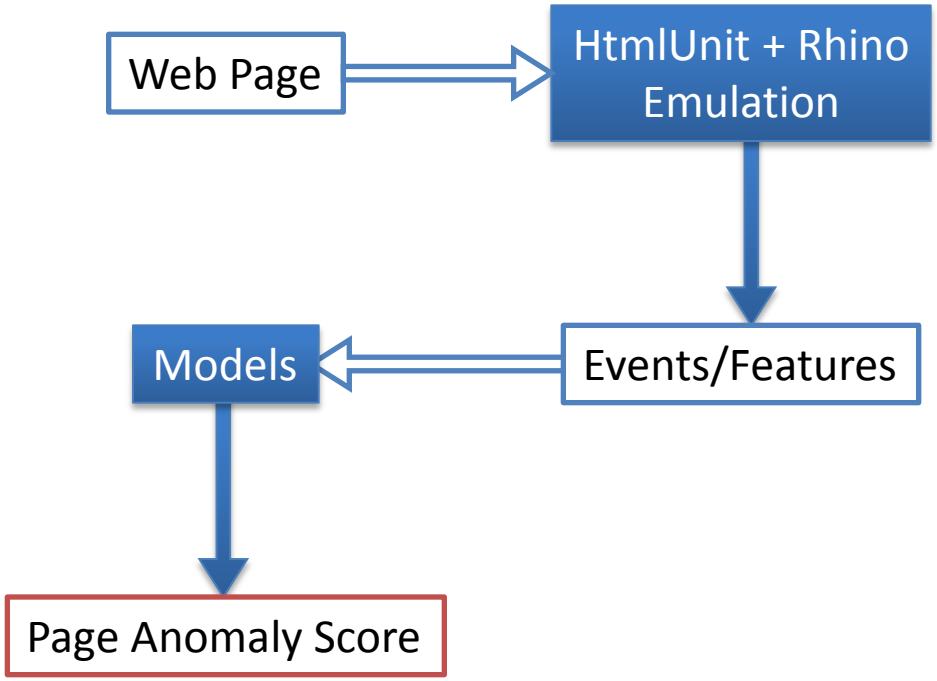
Use of Models

- Models operate in either:
  - Detection mode.
  - Training mode.
- There are a few different types of models utilized – paper has much more detailed information.
- Overall anomaly score assigned as weighted sum of all model scores.

Use of Emulation

- Emulation is designed to reveal the true behavior of the JavaScript code.
- HtmlUnit – Java-based framework for testing web applications.
  - Implements standard browser functionality (except visual page rendering).
  - Models HTML documents.
  - Supports JavaScript using Mozilla Rhino interpreter.
- HtmlUnit and Rhino were instrumented to extract the features used in the anomaly detection.

Diagram of Analysis Method



Evaluation of JSAND System

- Proposal was implemented as a system called JSAND.
- System can classify exploits used by a malicious page.
- System can use classification information to generate exploit signatures for other tools.

Evaluation - Datasets

- Known-good dataset – used to train models, determine anomaly thresholds, and compute false positives.
- Known-bad dataset – components are described in paper.
- Uncategorized datasets – no ground truth about the maliciousness of contained pages is available.

Evaluation – False Positive Rates

	# URLs Tested	# Reported Malicious URLs	# False Positives
Known-Good Subset	3,508	0	0
Crawling Set	115,706	137	15

The majority of the false positives uncovered on the crawling set were due to more different ActiveX controls being used on a benign page than had been seen in the training session, according to the authors.

Evaluation – False Negative Rates

The authors compared JSAND's false negative rate to that of three other tools that utilize different detection approaches.

Dataset	Samples (#)	JSAND FN	ClamAV FN	PhoneyC FN	Capture-HPC FN
Spam Trap	257	1 (0.3%)	243 (94.5%)	225 (87.5%)	0 (0.0%)
SQL Injection	23	0 (0.0%)	19 (82.6%)	17 (73.9%)	-
Malware Forum	202	1 (0.4%)	152 (75.2%)	85 (42.1%)	-
Wepawet-bad	341	0 (0.0%)	250 (73.3%)	248 (72.7%)	31 (9.1%)
Total	823	2 (0.2%)	664 (80.6%)	575 (69.9%)	31 (5.2%)

Capture-HPC was not used for the SQL injection and Malware forum datasets because the exploit binaries were hosted at sites that are no longer reachable.

Evaluation

- Capture-HPC and JSAND were run side-by-side on the 16,894 URLs in the Wepawet-uncat dataset.
- Capture-HPC found 285 confirmed malicious URLs, of which JSAND missed 25.
- JSAND flagged 8,714 URLs as anomalous (identifying 1 or more exploit for 762 of those URLs). Capture-HPC did not flag 8,454 of those.

Performance

- JSAND analyzed the Wepawet-bad dataset (341 samples) in 2:22 hours vs. Capture-HPC's time of 2:59 hours.
- Parallelization to three computers reduced the time to 1 hour.
- This still seems pretty uncomfortably long for 341 samples.

Conclusion

- It's quite likely that the Google Safe Browsing API is based on a blacklist of suspected phishing and malware pages built using the technology described in the first paper.
- Anomalous behavior discovery from the second paper could be used or borrowed from in the heuristical analysis from the first paper.
- Ultimately these techniques seem promising, if somewhat difficult to operate quickly.