Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications

Marco Balduzzi, Carmen Torrano Gimenez,
Davide Balzarotti, and Engin Kirda

8th Feb 2011

NDSS 2011

The Web as We Know It

- Has evolved from a collection of simple and static pages to fully dynamic applications
  - Applications are more complex than they used to be
  - Many complex systems have web interfaces
- As a consequence:
  - Web security has increased in importance (e.g. OWASP)
  - Attack against web apps constitute 60% of attacks on the Internet
  - Application being targeted for hosting drive-by-download content or C&C servers

Increased Importance of Web Security

- A lot of work done to detect injection type flaws:
  - SQL Injection
  - Cross Site Scripting
  - Command Injection
- Injection vulnerabilities have been well-studied, and tools exist
  - Stored procedures
  - Sanitization routines in languages (e.g., PHP)
  - Static code analysis (e.g., Pixy)
  - Dynamic techniques (e.g., Huang et al.)

HTTP Parameter Pollution (HPP)

- A new class of Injection Vulnerability called *HTTP Parameter Pollution (HPP)* is less known
  - Has not received much attention
  - First presented by *di Paola* and *Carettoni* at OWASP 2009

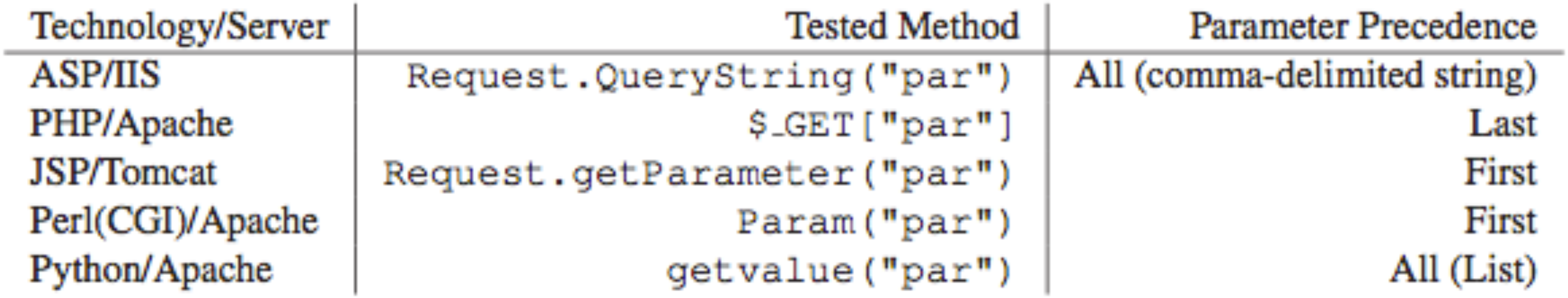
- Attack consists of injecting encoded query string delimiters into existing HTTP parameters (e.g. GET/POST)
  - If application does not sanitize its inputs, HPP can be used to launch client-side or server-side attacks
  - Attacker may be able to override existing parameter values and exploit variables out of a direct reach

Research Objectives

- To create the first automated approach for detecting HPP flaws
  - Blackbox approach, consists of a set of tests and heuristics
- To find out how prevalent HPP problems were on the web
  - Is the problem being exaggerated?
  - Is this problem known by developers?
  - Does this problem occur more in smaller sites than larger sites?
  - What is the significance of the problem?

HTTP Parameter Handling

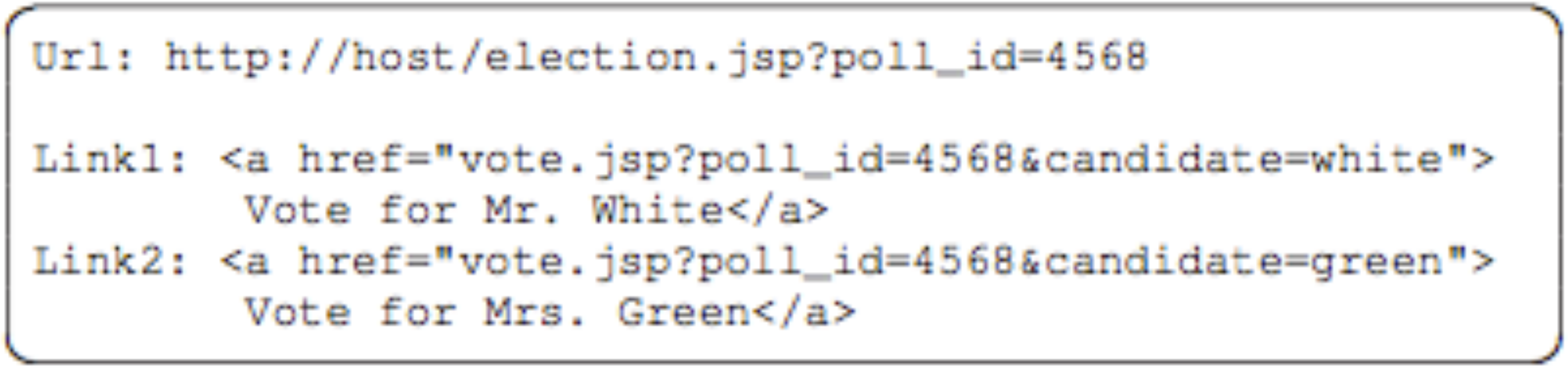
- During interaction with web application, client provides parameters via different channels (GET or POST)
  - http://www.site.com/login?login=alice
- What happens when the same parameter is provided twice?
  - http://www.site.com/login?login=alice&login=bob
  - If parameter is provided twice, language determines which is returned, e.g.:



Technology/Server	Tested Method	Parameter Precedence
ASP/IIS	`Request.QueryString("par")`	All (comma-delimited string)
PHP/Apache	`$_GET["par"]`	Last
JSP/Tomcat	`Request.getParameter("par")`	First
Perl(CGI)/Apache	`Param("par")`	First
Python/Apache	`getvalue("par")`	All (List)

HTTP Parameter Pollution

- An HTTP Parameter Pollution (HPP) attack occurs
  - When a malicious parameter P_{inj}, preceded by an encoded query string delimiter `(e.g. %26 for &)`, is injected into an existing parameter P_{host}
- Typical client-side scenario:
  - Web application for election and two candidates



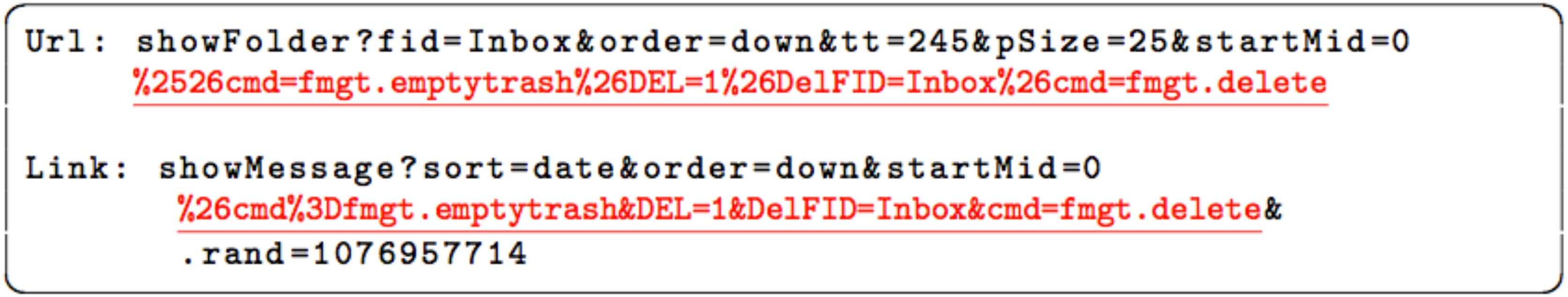
```
Url: http://host/election.jsp?poll_id=4568

Link1: <a href="vote.jsp?poll_id=4568&candidate=white">
        Vote for Mr. White</a>
Link2: <a href="vote.jsp?poll_id=4568&candidate=green">
        Vote for Mrs. Green</a>
```

HTTP Parameter Pollution

- *pool_id* is vulnerable and Attacker creates URL:
  - `http://host/election.jsp?poll_id=4568`**`%26candidate%3Dgreen`**
- The resulting page now contains two "polluted" links:
  - `<a href=vote.jsp?pool_id=4568`**`&candidate=green`**`&candidate=white>` `Vote for Mr. White </a>`
  - `<a href=vote.jsp?pool_id=4568`**`&candidate=green`**`&candidate=green>` `Vote for Mrs. Green </a>`
- If the developer expects to receive a single value
  - JSP's `Request.getParameter("candidate")` returns the 1st value
  - The parameter precedence is consistent…
  - Candidate Mrs. Green is always voted!

Parameter Pollution – More uses

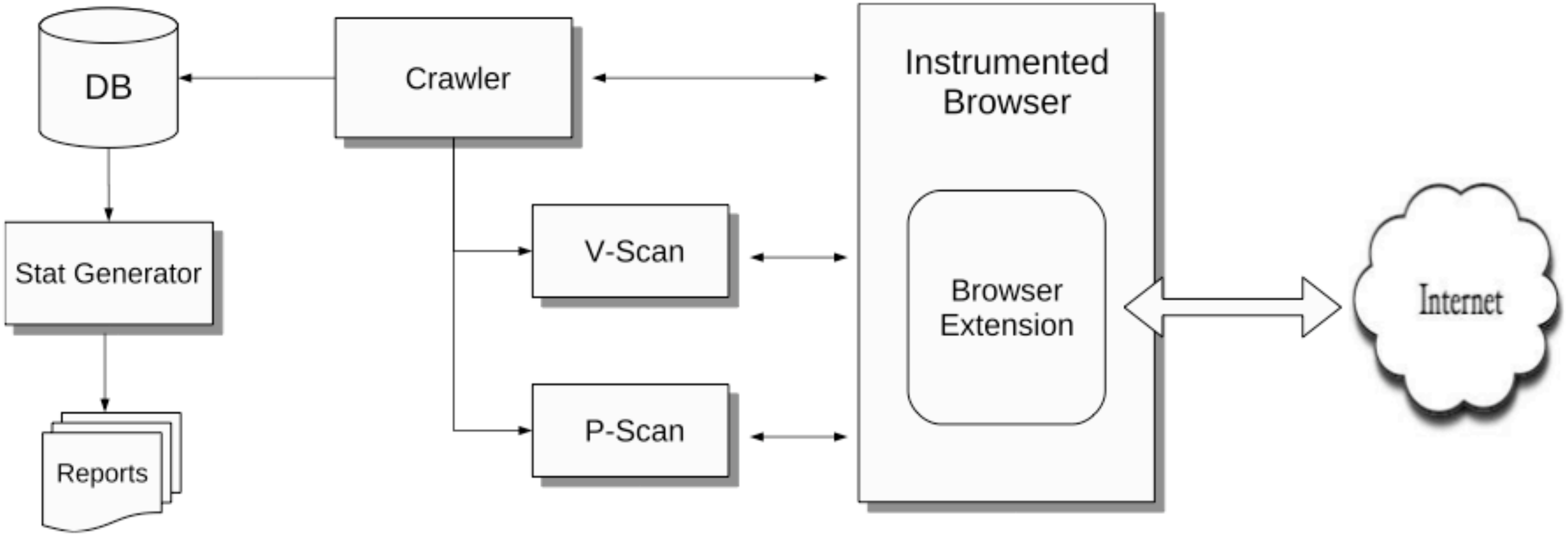
- Cross-channel pollution
  - HPP attacks can also be used to override parameters between different input channels (GET/POST/Cookie)
  - Good security practice: accept parameters only from where they are supposed to be supplied
- HPP to bypass CSRF tokens
  - E-mail deletion attack against Yahoo Mail



```
Url:  showFolder?fid=Inbox&order=down&tt=245&pSize=25&startMid=0
      %2526cmd=fmgt.emptytrash%26DEL=1%26DelFID=Inbox%26cmd=fmgt.delete

Link:  showMessage?sort=date&order=down&startMid=0
        %26cmd%3Dfmgt.emptytrash&DEL=1&DelFID=Inbox&cmd=fmgt.delete&
        .rand=1076957714
```

System for HPP Detection

- Main components: browser, crawler, two scanners



Main Components

① Instrumented browser fetches the webpages and renders their content

- Full support for client-side scripts (e.g. Javascript) and external resources (e.g. <embed>)
- Extracts all links and forms

② Crawler communicates with browser, determines URLs to visit and forms to submit. Passes the information to two scanners:

  ③ P-Scan: Determines page behavior when two parameters with the same name are injected

  ④ V-Scan: Tests and attempts to verify that site is vulnerable to HPP

P-Scan: Analysis of the Parameter Precedence

- P-Scan
  - Analyzes a page to determine the precedence of parameters when multiple occurrences of the same parameter are submitted
  - Take parameter *par1=val1*, generate a similar value *par1=new_val*
    - `Page0 (original): app.php?par1=val1`
    - `Page1 (test 1)   : app.php?par1=new_val`
    - `Page2 (test 2)   : app.php?par1=val1&par1=new_val`
  - How do we determine precedence? Naïve approach:
    - Page0==Page2 -> precedence on First parameter
    - Page1==Page2 -> precedence on Second parameter

P-Scan: Problem with the naïve approach

- In practice, naïve technique does not work well
  - Applications are complex, much dynamic content (publicity banners, RSS feeds, ads, etc.)
  - Hence, we perform pre-filtering to eliminate dynamic components (embedded content, applets, css stylesheets, etc.)
  - Remove all self-referencing URLs (as these change when parameters are inserted)
  - We then perform 4 different tests to determine similarity

P-Scan: Other Tests

- Identity test
  - Is the tested parameter considered by the application?
    - `Page0=Page1=Page2`
- Base test
  - Test assumes that the pre-filtering works perfectly (seldom the case)
- Join test
  - Are the 2 values combined somehow together?
- Fuzzy test
  - It is designed to cope with dynamic pages
  - Similarity between pages
  - Based on the `Gestalt Pattern Matching` algorithm

V-Scan: Testing for HPP vulnerabilities

- For every page, URL-encoded parameter is injected
  - E.g., `“%26foo%3Dbar”`
  - Then check if the `“&foo=bar”` string is included inside the URLs of links or forms in the answer page

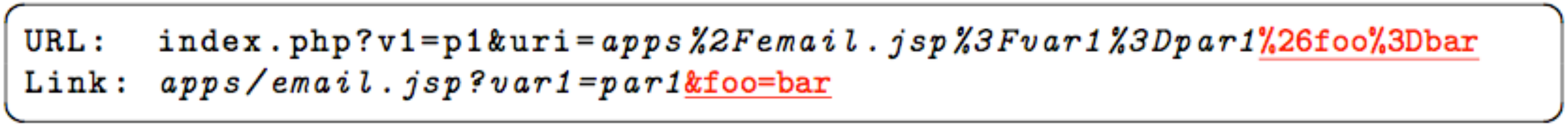
- V-Scan starts by extracting the list $P_{URL}=[P_{U1},P_{U2},\ldots P_{Un}]$ of the parameters that are present in the page URL, and the list $P_{body}=[P_{B1},P_{B2},\ldots P_{Um}]$ of the parameters that are present in links or forms contained in the page body

V-Scan: Testing for HPP vulnerabilities

- $P_A = P_{URL} \cap P_{Body}$: set of parameters that appear unmodified in the URL and in the page content (links, forms)
- $P_B = p \mid p \in P_{URL} \wedge p /\in P_{Body}$: URL parameters that do not appear in the page. Some of these parameters may appear in the page under a different name
- $P_C = p \mid p /\in P_{URL} \wedge p \in P_{Body}$: set of parameters that appear somewhere in the page, but that are not present in the URL

V-Scan: Special Cases

- E.g., one of the URL parameters (or part of it) is used as the entire target of a link



```
URL:   index.php?v1=p1&uri=apps%2Femail.jsp%3Fvar1%3Dpar1%26foo%3Dbar
Link:  apps/email.jsp?var1=par1&foo=bar
```

- Similar issues with printing, sharing functionalities

- To reduce false positives, we use heuristics
  - E.g., the injected parameter does not start with http://
  - Injection without URL-encoding

Implementation – The PAPAS tool

- PAPAS: Parameter Pollution Analysis System
  - http://papas.iseclab.org
- The components communicate via TCP/IP sockets
  - The browser component has been implemented as a Firefox extension
  - Advantage: We can see exactly how pages are rendered (cope with client-side scripts)
- PAPAS is fully customizable:
  - Three modes are supported
    - Fast mode, extensive mode, assisted mode
  - E.g., scanning depth, number of performed injections, page loading timeouts, etc.

Limitations

- PAPAS does not support the crawling of links embedded in active content
  - E.g., flash
- PAPAS currently only focuses on client-side exploits where user needs to click on a link
  - HPP is also possible on the server side – but this is more difficult to detect
  - Analogous to detecting stored XSS

Ethical Considerations

- Only client-side attacks. The server-side have the potential to cause harm
- We provided the applications with innocuous parameters (&foo=bar). No malicious code.
- Limited scan time (15min) and activity
- We immediately informed, when possible, the security engineers of the affected applications
  - Thankful feedback

Evaluation – the Fun Part ;)

- Two sets of experiments:

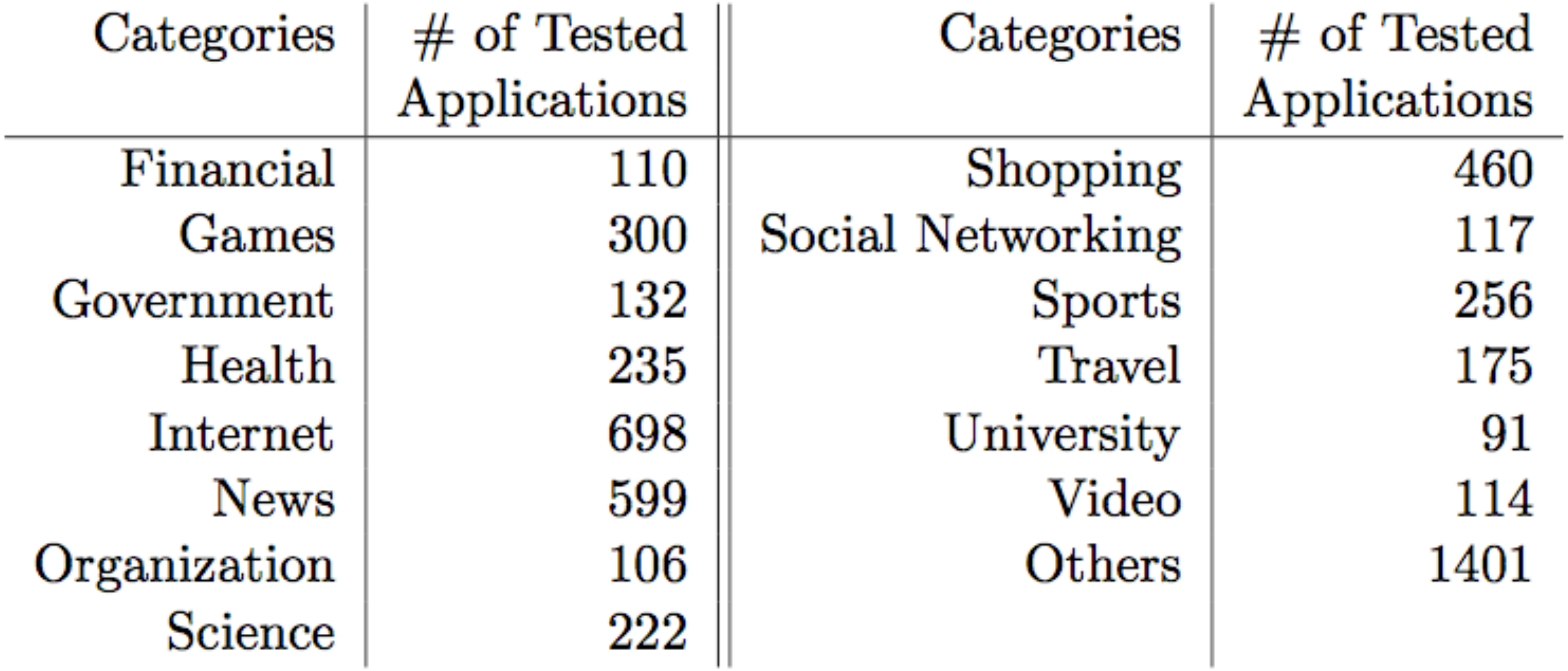
① We used PAPAS to scan a set of popular websites (Alexa TOP 5000)

- The aim: To quickly scan as many websites as possible and to see how common HPP flaws are
- In 13 days, we scanned 5016 websites, more than 149,000 unique web pages

② We then analyzed some of the sites we identified to be HPP vulnerable in more detail

Evaluation – The Dataset

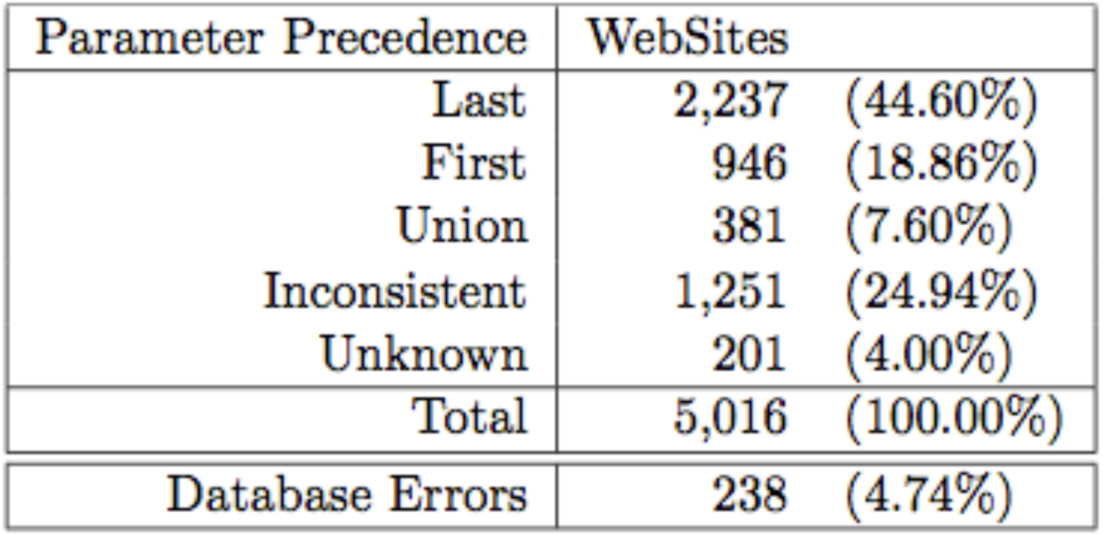
- Tested categories



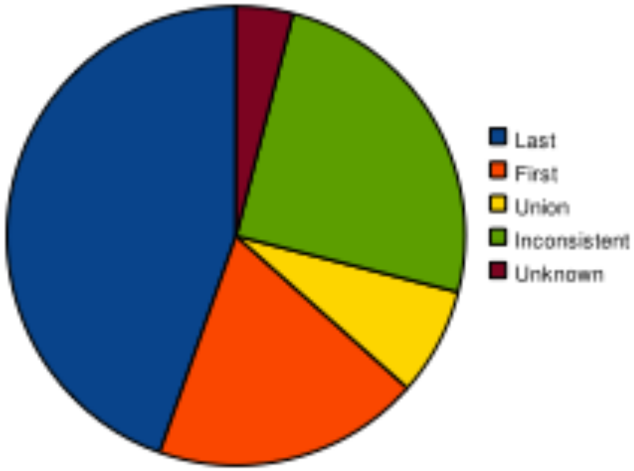
Categories	# of Tested Applications	Categories	# of Tested Applications
Financial	110	Shopping	460
Games	300	Social Networking	117
Government	132	Sports	256
Health	235	Travel	175
Internet	698	University	91
News	599	Video	114
Organization	106	Others	1401
Science	222		

Evaluation – Parameter Precedence

- *Inconsistent*: the website has been developed using a combination of heterogeneous technologies (e.g. PHP and Perl)
- This is perfectly safe if the developer is aware of the HPP threat… this is not always the case

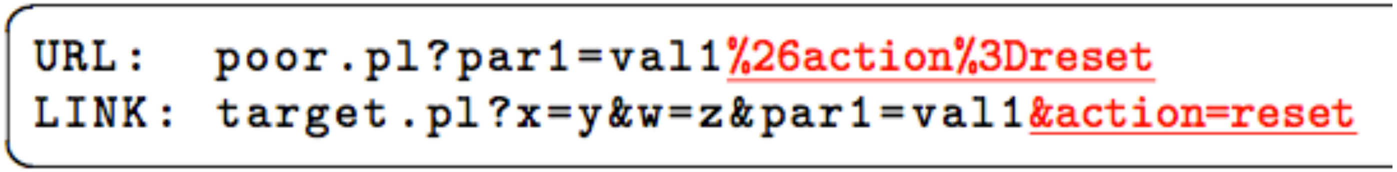


Parameter Precedence	WebSites	
Last	2,237	(44.60%)
First	946	(18.86%)
Union	381	(7.60%)
Inconsistent	1,251	(24.94%)
Unknown	201	(4.00%)
Total	5,016	(100.00%)
Database Errors	238	(4.74%)



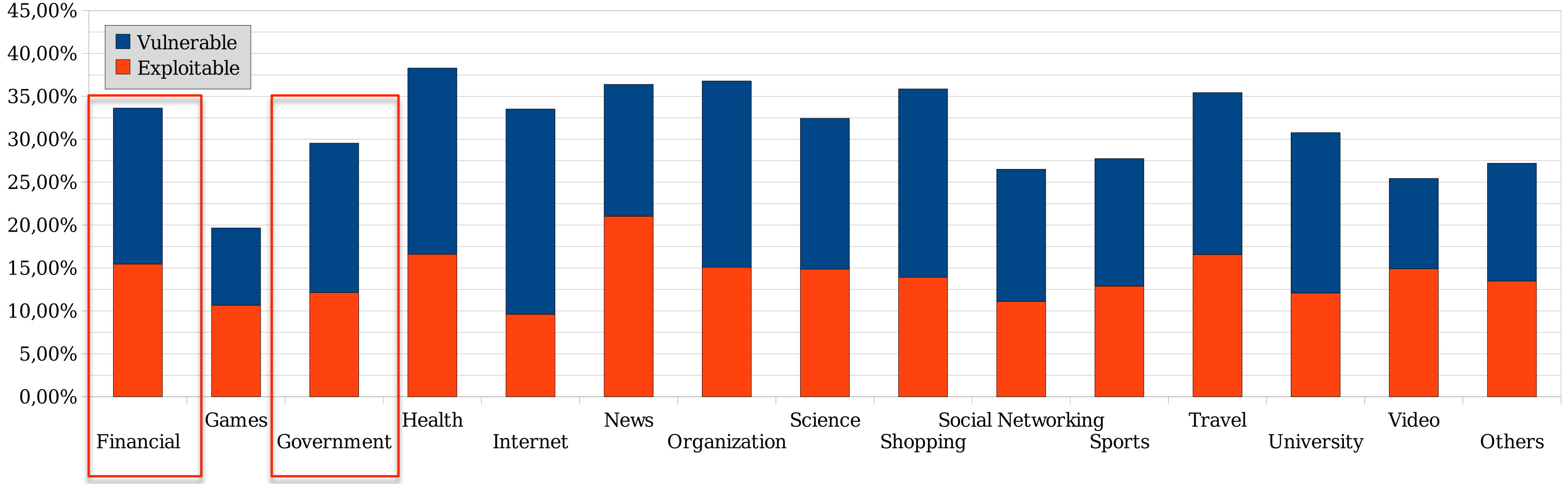
Evaluation – HPP Vulnerabilities

- PAPAS discovered that about 1500 (30%) websites contained at least one page vulnerable to HTTP Parameter Injection
  - The tool was able to inject an encoded parameter
- Vulnerable != Exploitable
  - Is the parameter precedence consistent?
- 702 applications are exploitable
  - The injected parameter either overrides the value of an existing one or is accepted as "new parameter"



```
URL:   poor.pl?par1=val1%26action%3Dreset
LINK:  target.pl?x=y&w=z&par1=val1&action=reset
```

Evaluation



- False positives: 10 applications (1.12%) use the injected parameter as entire target for one link
  - Variation of the special case we saw in slide 18 (V-Scan: special cases)

Some Case Studies

- We investigated some of the websites in more detail
  - Facebook, Google, Symantec, Microsoft, PayPal…
  - We notified security officers and some of the problems were fixed
  - Several shopping cart applications could be manipulated to change the price of an item
  - Some banks were vulnerable and we could play around with parameters
  - Facebook: share component
  - Google: search engine results could be manipulated

Conclusion

① We presented the first technique and system to detect HPP vulnerabilities in web applications.

- We call it PAPAS, http://papas.iseclab.org

② We conducted a large-scale study of the Internet

- 5,000 webapps

③ Our results suggest that Parameter Pollution is a largely unknown, and wide-spread problem

We hope our work will help raise awareness about HPP!

Questions?

I love you too, pollution!