WEB Application Security

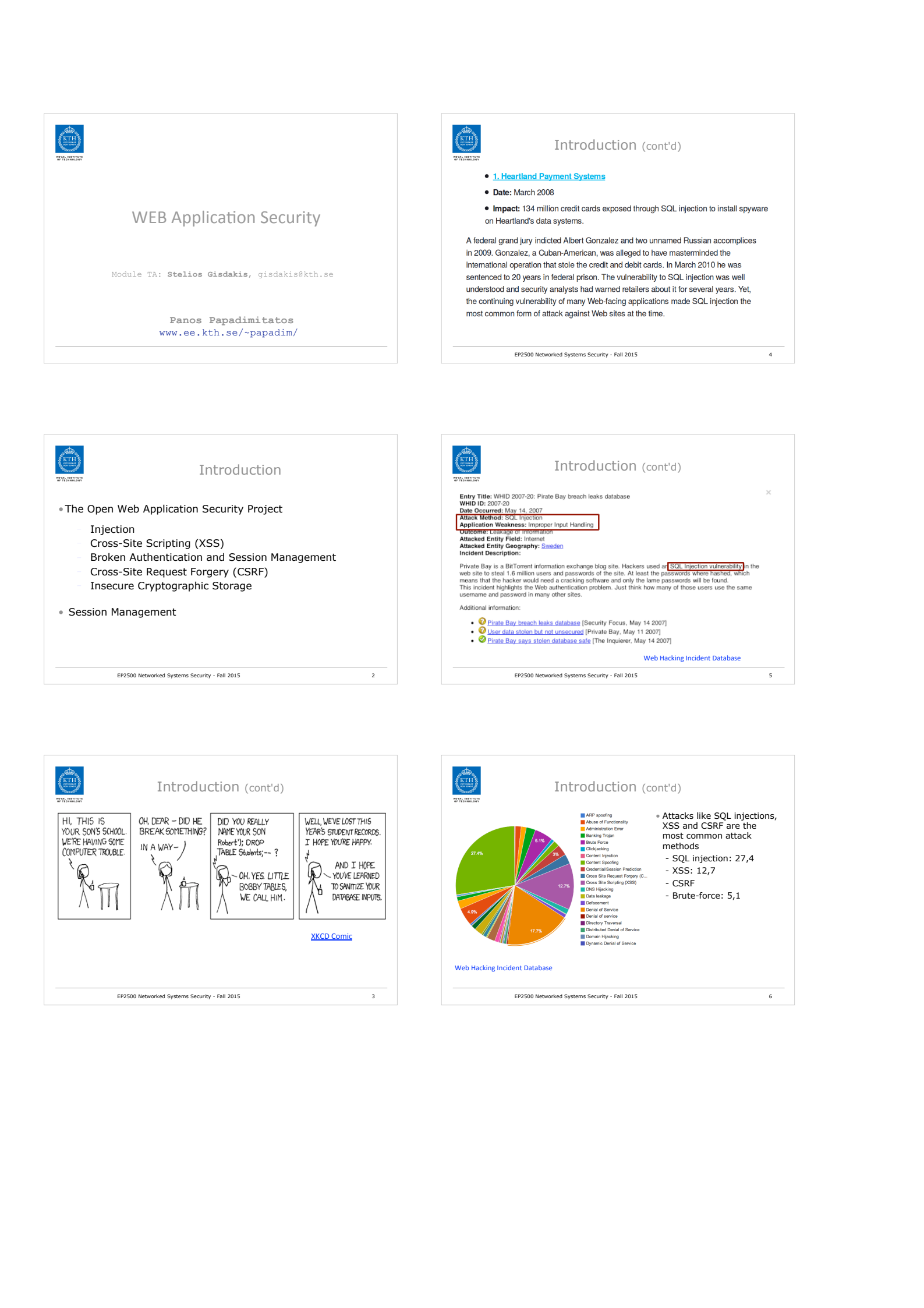
Module TA: **Stelios Gisdakis**, gisdakis@kth.se

Panos Papadimitatos
www.ee.kth.se/~papadim/

Introduction

- The Open Web Application Security Project
  - Injection
  - Cross-Site Scripting (XSS)
  - Broken Authentication and Session Management
  - Cross-Site Request Forgery (CSRF)
  - Insecure Cryptographic Storage
- Session Management

Introduction (cont'd)



XKCD Comic

Introduction (cont'd)

- [1. Heartland Payment Systems]
- **Date:** March 2008
- **Impact:** 134 million credit cards exposed through SQL injection to install spyware on Heartland's data systems.

A federal grand jury indicted Albert Gonzalez and two unnamed Russian accomplices in 2009. Gonzalez, a Cuban-American, was alleged to have masterminded the international operation that stole the credit and debit cards. In March 2010 he was sentenced to 20 years in federal prison. The vulnerability to SQL injection was well understood and security analysts had warned retailers about it for several years. Yet, the continuing vulnerability of many Web-facing applications made SQL injection the most common form of attack against Web sites at the time.

Introduction (cont'd)

Entry Title: WHID 2007-20: Pirate Bay breach leaks database
WHID ID: 2007-20
Date Occurred: May 14, 2007
Attack Method: SQL Injection
Application Weakness: Improper Input Handling
Outcome: Leakage of Information
Attacked Entity Field: Internet
Attacked Entity Geography: Sweden
Incident Description:

Private Bay is a BitTorrent information exchange blog site. Hackers used an SQL injection vulnerability in the web site to steal 1.6 million users and passwords of the site. At least the passwords where hashed, which means that the hacker would need a cracking software and only the lame passwords will be found.
This incident highlights the Web authentication problem. Just think how many of those users use the same username and password in many other sites.

Additional information:

- Pirate Bay breach leaks database [Security Focus, May 14 2007]
- User data stolen but not unsecured [Private Bay, May 11 2007]
- Pirate Bay says stolen database safe [The Inquierer, May 14 2007]

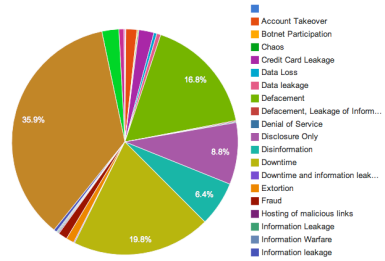
Web Hacking Incident Database

Introduction (cont'd)

- Attacks like SQL injections, XSS and CSRF are the most common attack methods
  - SQL injection: 27,4
  - XSS: 12,7
  - CSRF
  - Brute-force: 5,1

Web Hacking Incident Database

Introduction (cont'd)



Web Hacking Incident Database

- Top 3 outcomes:
  - Information Leakage (36%)
  - Downtime (20%)
  - Defacement (17%)
  - ...

Introduction (cont'd)

#Sownage (Sony + Ownage) Phase 1 will begin within the next day. We may have a pre-game show for you folks though. Stay tuned.

The group of hackers who compromised **passwords** and posted a fake news story to the PBS Frontline website on Sunday has taken responsibility for the Sony hack, claiming in a press release that a single SQL injection was all it took to access the data.

Introduction (cont'd)

SQL injection

XSS

?

CSRF

Un-validated Redirects and Forwards

Broken Authentication

?

?

Click-Jacking

Insufficient Transport Layer Protection

The Need for Web Application Security

The OSI model

- Many popular applications nowadays are web-based
  - Variety of protocols
  - Broad gamut of threats
  - Numerous attacks
- Defense in depth: Security mechanisms at multiple layers

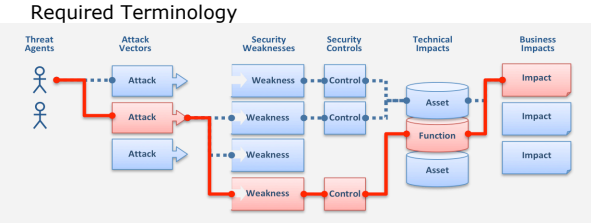
Application Security Risks

Terminology

- **Threat Agent**: an entity that tries to compromise the system or specific assets
- **Attack Vector**: a methodology to launch an attack
- **Weakness**: a software problem that could lead to vulnerabilities
- **Vulnerability**: A likely to exploit (by a threat agent) existing weakness
- **Security Control**: Countermeasures we apply to reduce or eliminate vulnerabilities
- **Impact**: The effect of an exploited vulnerability, both in terms of the system functionality (technical impact) and of the business process (business impact)

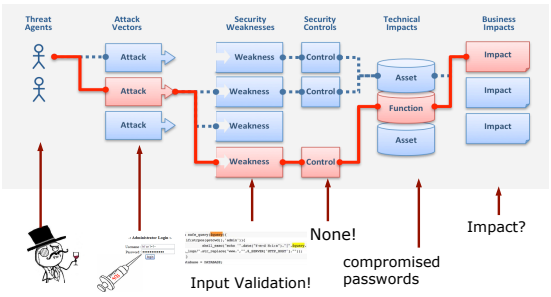
Application Security Risks (cont'd)

Required Terminology



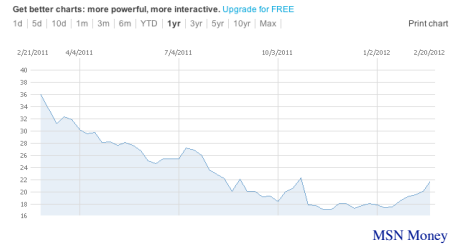
OWASP Top 10

Application Security Risks (cont'd)



Application Security Risks (cont'd)

Analysis of the *SONY* incident



The Open Web Application Security Project (Top 10-2010)

OWASP

- OWASP is a non-profit organization that focuses on Application Security
- The OWASP Top 10 project is an effort to identify the 10 most common Application Security Risks (updated every 3 years)

The Open Web Application Security Project (Top 10-2013 new list expected in 2016)

- A1 Injection
- A2 Broken Authentication and Session Management (XSS)
- A3 Cross Site Scripting (XSS)
- A4 Insecure Direct Object References
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Missing Function Level Access Control
- A8 Cross Site Request Forgery (CSRF)
- A9 Using Components with Known Vulnerabilities
- A10 Unvalidated Redirects and Forwards

frequency

Cross-Site Scripting (XSS)

Problem Description:

- **Type**: injection problem
- **Attack Method**: XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.
- **Weakness**: Flaws that allow these attacks are widespread, notably when a web application uses input from a user in the output it generates without validating or encoding it.

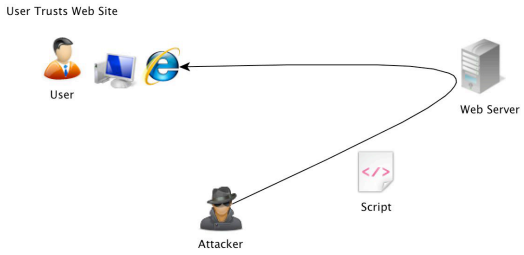
XSS OWASP

Reflected XSS

Two Main types of XSS

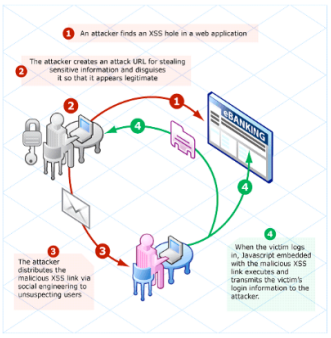
- **Reflected XSS**
  - Non persistent type of XSS (malicious script is not stored)
  - Attack Method: The modus operandi includes a design step. The attacker creates and tests an offending URI, a social engineering step; she convinces her victims to load this URI on their browsers, and the eventual execution of the offending code — using the victim's credentials.

Reflected XSS (cont'd)



Reflected XSS (cont'd)

- Launching a Stored XSS Attack requires methods for the distribution of attack vectors to the victims



HP Communities

Reflected XSS (cont'd)

- **Execution Steps:**
  - Detection of a vulnerable to XSS site
  - Preparation of XSS attack vector
  - (URL+ malicious script)
  - Distribution of XSS attack vector (SPAM is useful here)
  - The victim issues the request to the server
  - The server responds
  - The script is executed

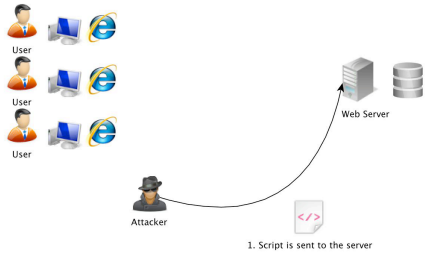
Stored XSS

Two Main types of XSS

- **Stored XSS**
  - Persistent type of XSS (malicious script IS **stored**)
  - Attack Method: Stored Cross Site Scripting (XSS) is the most dangerous type of Cross Site Scripting. Web applications that allow users to store data are potentially exposed to this type of attack.

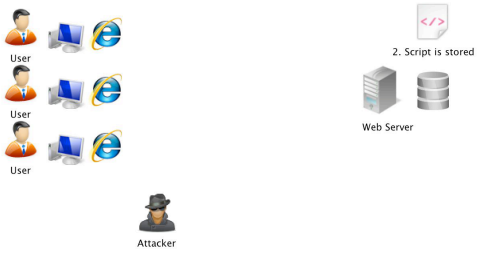
Stored XSS (cont'd)

Stored XSS



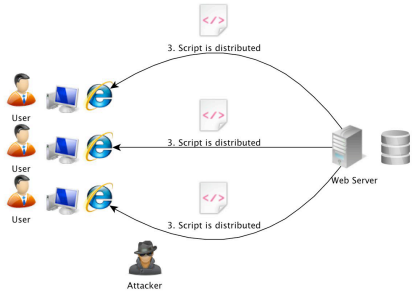
Stored XSS (cont'd)

Stored XSS



Stored XSS (cont'd)

Stored XSS



The script is sent to any user accesses the website

Reflected VS Stored XSS

Video on XSS

Consequences of XSS

- Disclosure of Cookies
- Disclosure of user files
- Malware installation
- Page redirection
- Account takeover

Consequences of XSS

- **Disclosure of Cookies**
  - A malicious script can disclose only specific cookies
  - JavaScript scripts are allowed to interact with pages according to the **Same Origin Policy**

Same Origin Policy

- **Same Origin Policy**
  - Scripts are allowed to interact with a page if and only if:
    - They use the same **protocol**
    - They come from the same **domain**
    - They come from the same **host**

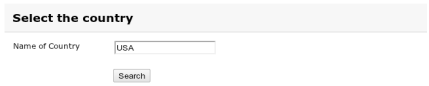
SQL injection

Problem Description:

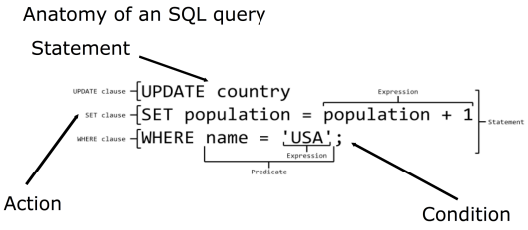
- **Type**: injection problem
- **Attack Method**: A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application.
- **Weakness**: SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

Injection OWASP

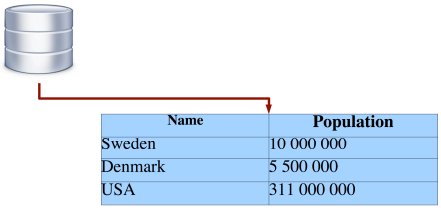
Anatomy of an SQL query



Anatomy of an SQL query (cont'd)

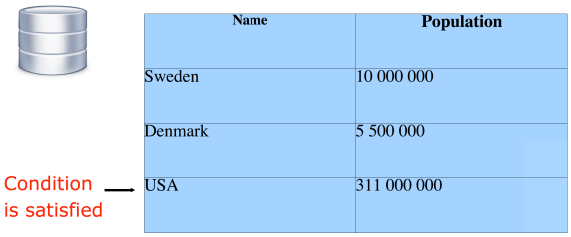


Anatomy of an SQL query (cont'd)



Name	Population
Sweden	10 000 000
Denmark	5 500 000
USA	311 000 000

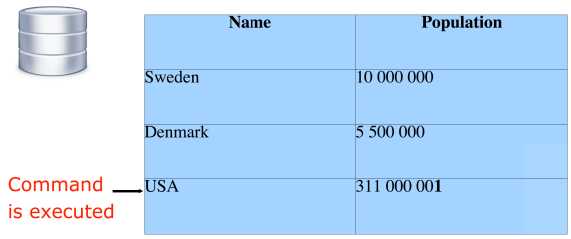
Anatomy of an SQL query (cont'd)



Name	Population
Sweden	10 000 000
Denmark	5 500 000
USA	311 000 000

Condition is satisfied

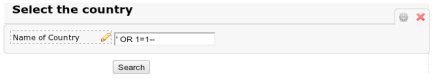
Anatomy of an SQL query (cont'd)



Name	Population
Sweden	10 000 000
Denmark	5 500 000
USA	311 000 001

Command is executed

An SQL injection



An SQL injection (cont'd)

Update country

SET population = population + 1
WHERE name = '' OR 1=1 --

Always TRUE!

An SQL injection (cont'd)

Anatomy of an SQL query (cont)

Name	Population
Sweden	10 000 000
Denmark	5 500 000
USA	311 000 000

Condition is satisfied

An SQL injection (cont'd)

Anatomy of an SQL query (cont)

Name	Population
Sweden	10 000 001
Denmark	5 500 001
USA	311 000 001

Command is executed

SQLi and XSS Countermeasures

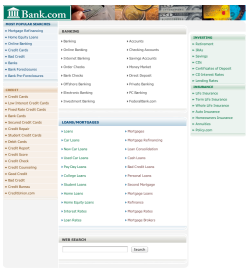
- **RULE #0** - Never Insert Untrusted Data Except in Allowed Locations (XSS)
- **RULE #1** - HTML Escape Before Inserting Untrusted Data into HTML Element Content (XSS)
- **RULE #2** - Attribute Escape Before Inserting Untrusted Data into HTML Common Attributes (XSS)
- **RULE #3** - JavaScript Escape Before Inserting Untrusted Data into JavaScript Data Values (XSS)
- **RULE #4** - CSS Escape And Strictly Validate Before Inserting Untrusted Data into HTML Style Property Values (XSS)
- **RULE #5** - URL Escape Before Inserting Untrusted Data into HTML URL Parameter Values (XSS)
- **Sanitize SQL inputs**(SQLi)
  - Use WHITELISTING (what inputs should we accept?)
- Use **stored procedures** and **prepared statements** (SQLi)

OWASP XSS Prevention

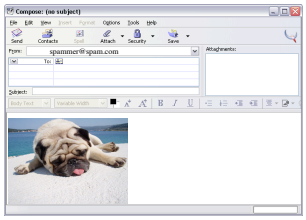
SQLi and XSS Countermeasures (cont'd)

- Why together?
  - Both risks take advantage of the same weaknesses
    - XSS: Why should a web form accept **<script>**
    - SQLi: Why should a web form accept characters such as : ', –, …
  - Typical examples of lack of **Input Validation**
    - User Input finds is trusted without being filtered!

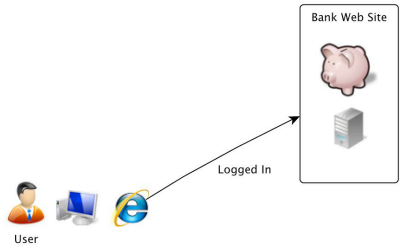
Cross Site Request Forgery (CSRF)



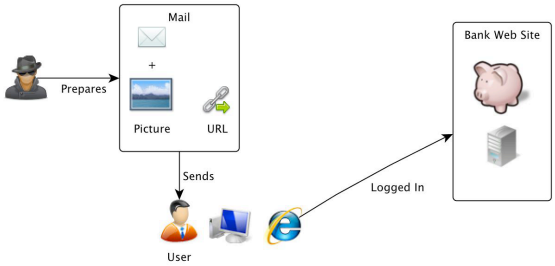
Vs



Cross Site Request Forgery (cont'd)



Cross Site Request Forgery (cont'd)



Cross Site Request Forgery (cont'd)

- What will happen?

It depends on the content of the URL the attacker hid within the picture

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
```

- This would transfer "some" money to another account!

Cross Site Request Forgery Countermeasures

- **Verification** of all critical actions
- Session identifiers for all actions
- CAPTCHA codes

- Awareness

Broken Authentication and Session Management (Insecure Cryptographic Storage)

Example 1:

Name	Password
Alice	catwoman
Bob	BObRocks
Eve	eve1234

In case the system is compromised, the attacker has access to every single Account!

Insecure Cryptographic Storage (cont'd)

Example 2:

Name	Password
Alice	84fe2a70870a8759cba0e77df1b6fc29fc79f44b
Bob	67e5ddf35cfde1597be941214990a3cc4f90db44
Eve	cd0a4ac00802af6458ce5f8153c971a82ca0aefd

Just hashing passwords does not provide security against **unauthorized disclosure**

Insecure Cryptographic Storage (cont'd)

A simple Hash is not enough to protect sensitive data (Rainbow Tables)

- Salting of Hashes
- Encryption

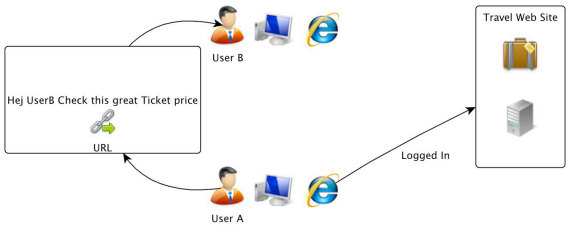
Session Management

- HTTP is STATELESS
- State is kept with the use of session Ids.

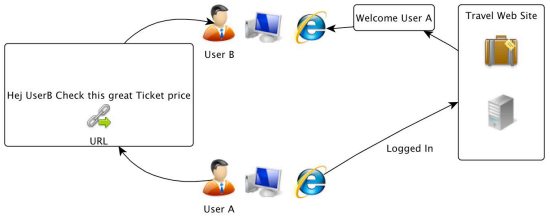
Session Management (cont'd)

- HTTP is STATELESS
- State is kept with the use of session Ids
  - Session IDs are kept inside URLs
  - or Cookies

Broken Authentication and Session Management (Improper Session Management)

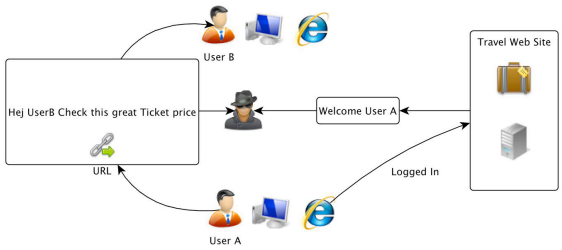


Session Management (cont'd)



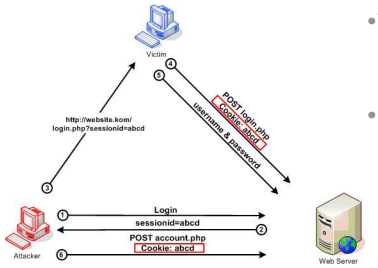
Authentication and Session management is done with a session ID inside the URL!

Session Management (cont'd)



Authentication and Session management is done with a session ID inside the URL!

Session Management (Session Fixation)

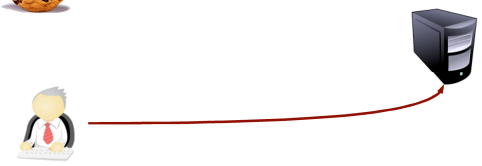


- The attacker creates a session with a known session Id
- No need to guess the session Id of the victim

OWASP Session Fixation

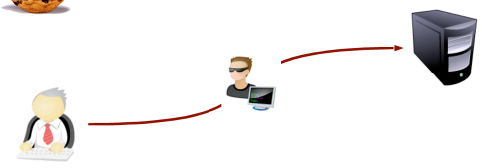
Session Management (Cookie Poisoning)

GET /store/buy.asp?checkout=yes HTTP/1.0 Host: www.onlineshop.com
Accept: */* Referrer: http://www.onlineshop.com/showprods.asp
Cookie: SESSIONID=570321ASDD23SA2321; BasketSize=3; Item1=2892; Item2=3210; Item3=9942; TotalPrice=16044;



Session Management (Cookie Poisoning cont'd)

GET /store/buy.asp?checkout=yes HTTP/1.0 Host: www.onlineshop.com
Accept: */* Referrer: http://www.onlineshop.com/showprods.asp
Cookie: SESSIONID=570321ASDD23SA2321; BasketSize=3; Item1=2892; Item2=3210; Item3=9950; TotalPrice=16052;



Session Management (cont'd)

- Proper Session Management
  - Session IDs creation
    - Sufficient Length (prevent Brute-Force)
    - Sufficient Entropy (to prevent guessing and collisions)
  - Session IDs transmission
    - Use of SSL/TLS

Conclusions

- The Web has become much more than a way of accessing and sharing information
  - Web is a platform (maybe no need for an Operating System?)
  - Web is a means of doing business.
  - Web is a service.
- The Web is vulnerable
  - SQL injection attacks
  - XSS attacks
  - CSRF
  - ...

Conclusions

- Web Application Security is a undisputed requirement for:
  - Academics
  - Developers/Engineers
  - Users
- Web Application Security is here to stay (top skill for Security Professionals)...

#2 - System, Network, and/or Web Penetration Tester* - Top Gun Job

Top Security Jobs (SANS)