# Part III

## Cross Site Request Forgery (CSRF / XSRF)

# What is Cross Site Request Forgery?

- quite unknown class of attack on web applications with potential high damage potential
- abuses the trust of a web application in the victim's browser

# „Blind" Browser Trust (I)

- Classic web applications trust the security features of other layers to ensure the origin of a HTTP request
  - TCP/IP / IPSEC / VPN
  - SSL
  - Session Cookies
- All these features ensure that a HTTP Request can be assigned to a browser session

# „Blind" Browser Trust (II)

- classic network security feature secure the origin but not the intention
- not every HTTP request performed by a browser is a result of a user interaction
  - images
  - external style sheet files
  - external JavaScript
  - ...

# Simple CSRF Attacks (I)

- strictly seen is every request to an external resource a Cross Site Request

```
<img src="http://somewhereelse.com/high_quality_image.png">
```

- harmless until the image is embedded in a high-traffic-site that overloads your server

# Simple CSRF Attacks (II)

- What happens when a URL in an IMG tag is not an image but triggers an action in a web application?

```
<img src="http://www.mybank.de/transfer.php?
amount=0,99&account=123456789">
```

# CSRF Example: Step by Step (I)

- Blackhat embeds a link to an external image in user generated content in a movie community

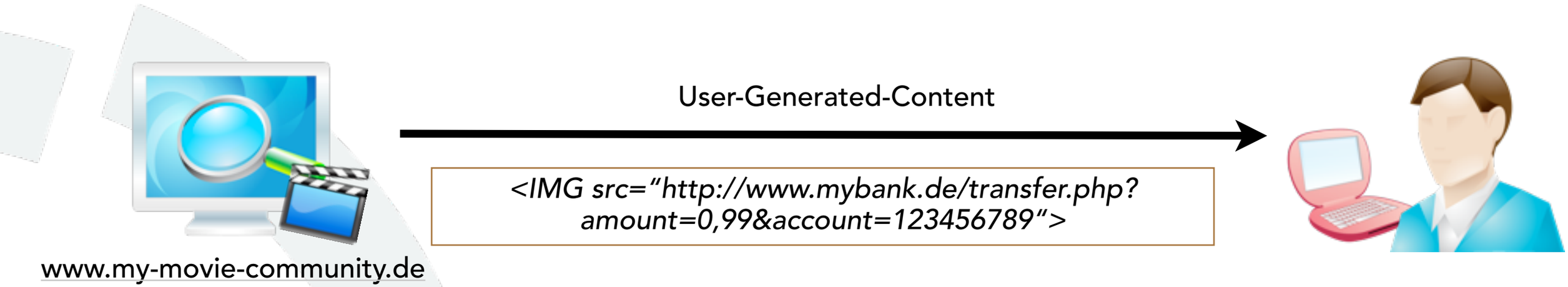
User-Generated-Content

www.my-movie-community.de

*<IMG src="http://www.mybank.de/transfer.php?amount=0,99&account=123456789">*

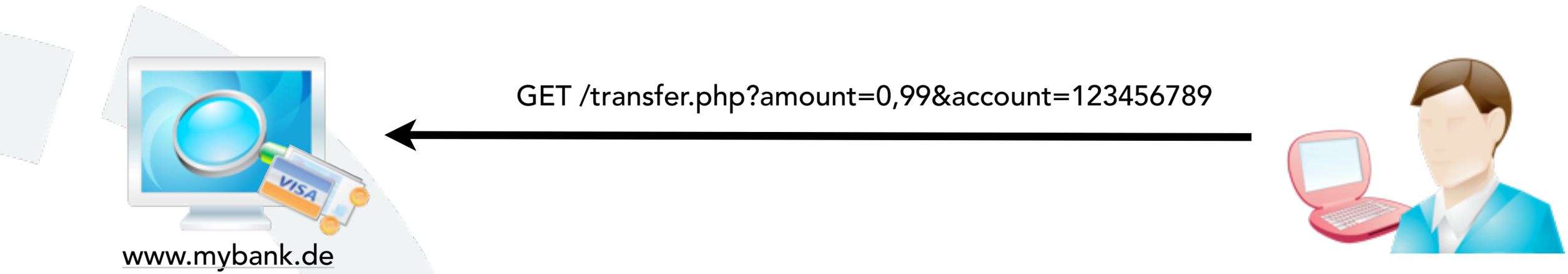
# CSRF Example: Step by Step (II)

- Victim visits the „Movie Community“
- and gets the malicious IMG tag delivered as part of the user generated content



# CSRF Example: Step by Step (III)

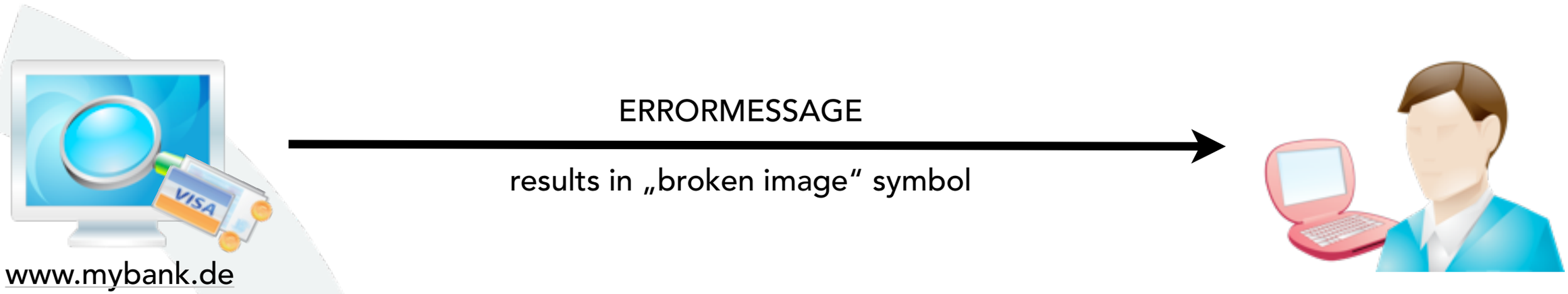
- Browser of victim sends a GET Request to www.mybank.de, to retrieve an image
- This triggers the execution of the script transfer.php



# CSRF Example: Step by Step (IV)

## Alternative 1: Victim was not logged in

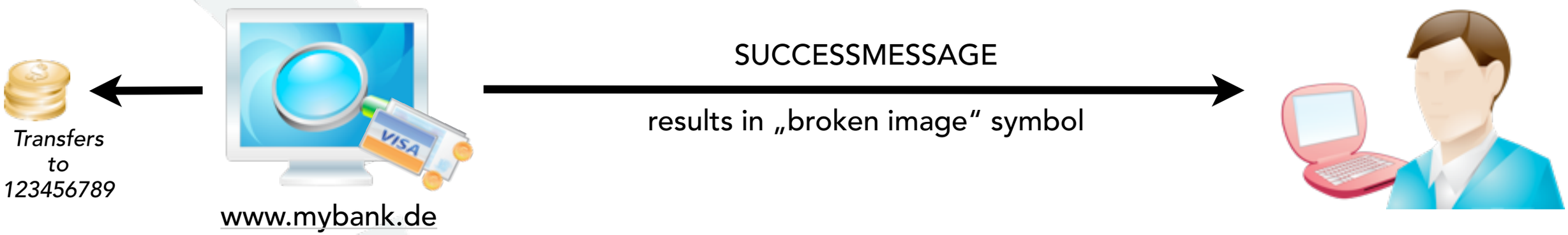
- www.mybank.de cannot assign the request to a user session and therefore returns an error message
- error message will not be displayed in the browser, because an image is expected - „broken image“ symbol



# CSRF Example: Step by Step (V)

## Alternative 2: Victim was logged in

- www.mybank.de can assign the request to a user session because the browser request also contains the domain cookies and maybe HTTP auth informations
- bank transfer is executed
- success message will not be displayed in the browser, because an image is expected - „broken image“ symbol



# CSRF Example: Step by Step (VI)

- Blackhat gets rich and parties...

# Real Damagepotential of CSRF

- triggering a bank transfer by CSRF is unlikely
- real damage depends on the unprotected actions that can be triggered
    - simple user logout
    - up to complete system compromise
- but an attacker must exploit the victim while the victim is logged in
- **Firewall Bypass:** attacks against the intranet are possible, because the browser is able to access it

# Attackvectors

- external images in bulletin boards
- malicious web sites
- malicious links in comments / bulletin board threads
- HTML emails
- Adobe Flash files
- Adobe PDF files
- SVG files
- ...

# CSRF Protections

CSRF Protections can be grouped in

- not working protections
- working protections

# Not working CSRF Protections (I)

- „Referer“ protection
  - checking the „Referer“ HTTP header
  - ignoring the HTTP requests if „Referer“ is incorrect
- not working because
  - (desktop-)firewalls block/modify the „Referer“
  - browser sometimes do not send „Referer“
  - attack from within Adobe Flash can spoof „Referer“

# Not working CSRF Protections (II)

- „POST instead of GET“ protection
  - GET only for data retriebal, POST for data manipulation
- positiv
  - represents the original sense of GET and POST
  - stops exploits through IMG tags or other external resources
  - data manipulation not through search engine robots

# Not working CSRF Protections (III)

- „POST instead of GET“ is not a CSRF protection
  - JavaScript can send POST requests to external URLs
  - Adobe Flash can send POST requests to external URLs and depending on crossdomain.xml configurations it is even possible to read the results

# Working CSRF Protections (I)

- all working CSRF protection are based on at least one request parameter that is unknown and not guessable by an external attacker
- request authorization by
  - entering the password / a TAN
  - CAPTCHA (limited)
  - secret request tokens
- protections are **less or not secure** in case of XSS but then it isn't CSRF anymore *(SSRF Same Site Request Forgery)*

# Working CSRF Protections (II)

Protection by entering the password / a TAN

- attacke usually does not know the password / TAN
- otherwise there is a bigger problem than CSRF
- request with valid password / TAN is intended

➡ CSRF not possible

# Working CSRF Protections (III)

Protection by CAPTCHA

- makes automatic abuse hard or impossible
- CAPTCHA image is protected by same-origin-policy
- attacker won't be able to see the image without a browser vulnerability
- requests with valid CAPTCHA answer can only be intended
  *(if CAPTCHA URL is not guessable)*

➡ CSRF not possible

# Working CSRF Protections (IV)

Protection by „secret“ request tokens

- „secret“ = (one-time-)token is stored in the session
- works in the background - e.g. hidden form fields
- system works also for AJAX requests
- protected by same-origin-policy - attack cannot determine valid requests tokens *(without another vulnerability in place)*

➡ CSRF not possible

# Generation of „secret“ Form Tokens

```
function generateFormToken($formName)
{
   $token = md5(uniqid(microtime(), true));
   $_SESSION[$formName.'_token'] = $token;
   return $token;
}
```

# Checking the „secret“ Form Tokens

```
function verifyFormToken($formName)
{
  $index = $formName.'_token';

  // There must be a token in the session
  if (!isset($_SESSION[$index])) return false;

  // There must be a token in the form
  if (!isset($_POST['token'])) return false;

  // The token must be identical
  if ($_SESSION[$index] !== $_POST['token']) return false;

  return true;
}
```

# Using the „secret“ Form Tokens

```
if (!isset($_POST[‘submit‘])) {
  $newToken = generateFormToken(‘loginForm‘);

  ... Display Form ...

  echo “<input type=‘hidden‘ name=‘token‘ value=‘$newToken‘>“;

  ... Rest of Form ...
  die();
}

if (!verifyFormToken(‘loginForm‘)) {
  die(‘CSRF Attack detected.‘);
}

... Form Processing ...
```

# Questions ?