Developing Extensions With Security in Mind

Tutorial

Henning Pingel
<henning@typo3.org>

T3CON08, Berlin
October 09, 2008

(Last update of slides: October 16, 2008)

TYPO3

Welcome!

Have a great time
at T3CON08!

Overview

▪ Intro	(15 mins)
▪ Dangers of URI Tampering	(30 mins)
▪ Demystifying HTTP Requests	(45 mins)
▪ **Coffee Break**	(30 mins)
▪ Unsanitized User Input	(45 mins)
▪ Security Issue Handling	(15 mins)
▪ Discussion	(30 mins)

Why a tutorial?

- Four times more time than in a talk
- More time for questions and discussion

Who am I and who are you?

- This tutorial is officially targeted at extension developers.
- How many extensions have you written?
- How many vulnerability types do you know and understand?
  - 1 to 4
  - 5 to 10

The TYPO3 Security Team

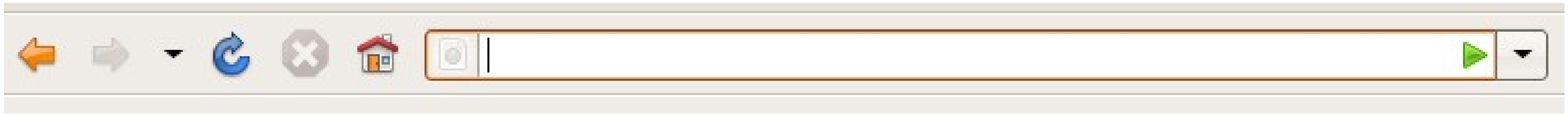
- On T3DD08 in Elmshorn (incomplete)
- Often happy about secure software, but sometimes...

...unhappy about insecure extensions

Security Bulletins in 2008 (so far)

TYPO3-20080924-2, TYPO3-20080924-1,
TYPO3-20080919-1, TYPO3-20080916-1,
TYPO3-20080701-4, TYPO3-20080701-3,
TYPO3-20080701-2, TYPO3-20080701-1,
TYPO3-20080619-1, TYPO3-20080611-1,
TYPO3-20080527-2, TYPO3-20080527-1,
TYPO3-20080515-2, TYPO3-20080515-1,
TYPO3-20080513-4, TYPO3-20080513-3,
TYPO3-20080513-2, TYPO3-20080513-1,
TYPO3-20080505-2, TYPO3-20080505-1,
TYPO3-20080416-2, TYPO3-20080416-1

Dangers of URI Tampering



- There are infinite possibilities to fill the URI bar of the web browser.
- “Try everything you want and see what happens.”

Information Disclosure

- through certain files of content-type
  - text/plain or
  - text/html
- through PHP script files
  - directly executable PHP includes or
  - forgotten debug scripts

TYPO3 Extensions...

- ... often consist of a large number of files.
- Those files can contain different “flavours” of information.
- Not all files within an extension are addressed at the same group of people.
- Metaphorically speaking, an extension can be a box of...

Chocolates

Target Groups of Files in Extensions

- Images, CSS and HTML-Templates are there to be served to the frontend, to the whole world.
- Some images may only be there to be displayed in a backend module.
- SXW-Files, readme files, trace files are addresses to the administrator.
- Artefacts: files uploaded by accident (*.bak files, project files, subversion files)

Possible Impact

- All extension's files are accessible via HTTP on a TYPO3 default installation. (.htaccess protection is beyond the scope of this tutorial.)
- File structure is publicly available on Extension Repository (TER). But this is not the problem!
- Impact: **Information Disclosure** may be possible through **Forced browsing** and **File location guessing.**
- Is there a real life metaphor for these terms?

The Fresh Milk Metaphor

How to get the milk with the best best-before date...

Harmless Example

tt_news Changelog File: A piece of information...

- ...that is available from any T3 site using tt_news (via **Forced browsing**)
- ...that is of no interest for the ordinary visitor of a web site.
- ...that contains information about which version of an extension is used.

Example for Information Disclosure

I use this real life example with the kind permission of the extension's author. Thank you!

Extension w4x_backup [Demo]

- Bulletin was published in June 2007
- Version 0.9.1 and below are vulnerable
- Version 0.9.2 contains security fixes
- Details: Bulletin TYPO3-20070612-1
- Impact: In worst case, download of backup archive (containing db and file backup)
- Log file (with static filename and path) contains file name of backup archive

Best Practice

If your extension really needs to generate a file to store data (like traces, logs, configuration settings), avoid Information Disclosure by avoiding content-types text/plain or text/html.
Put it into a dynamically generated PHP script with a .php file extension and also avoid guessable file names.

Directly Executable PHP Scripts and Includes

- Visible PHP/MySQL error messages
- Are TYPO3's mechanisms of authentication, user privilege checks and permission checks respected before their code get's executed?

Example

- Extension ftpbrowser (similar to quixplorer)
- Bulletin was released in July 2007
- Version 0.1.2 and below are vulnerable
- Version 0.1.3 contains security fixes
- Bulletin TYPO3-20070709-1
- Impact: Incorrect Authentication allows file upload if **register_globals** is activated in php.ini.
- You may analyse the code yourself in DIY phase.

I use this real life example with the kind permission of the extension's author. Thank you!

How to prevent script execution outside of TYPO3 context?

- Best case: All PHP code is wrapped in a class, class is not instantiated in file, nothing can happen.
- `die()` if some elemental TYPO3 constant doesn't exist (Example, or search TYPO3 core code for more examples)
- Backend modules: $BE_USER->modAccess($MCONF,1);

Embedding 3rd Party Tools in TYPO3 Extensions

- Everything already said also applies to 3rd party tools.
- Additional problems:
  - Different authentication concepts (Example: Folder based authentication contra file based authentication.
  - Different architecture: Direct access to Superglobals
  - Different user types / session management
  - Keep up with upstream security announcements
- Example cases: phpmailer, mysqldumper, phpmyadmin

URI-Tampering: A Small Dilemma?

- TYPO3 V4 Extension architecture is “like it is”, but it is the same situation for many currently popular web application.
- An extension developer can still prevent all vulnerabilities through careful design.
- Site administrators may put rules into a .htaccess file that prevent access to files via HTTP.
- FLOW3 / TYPO3 V5 have a different architecture.

Questions?

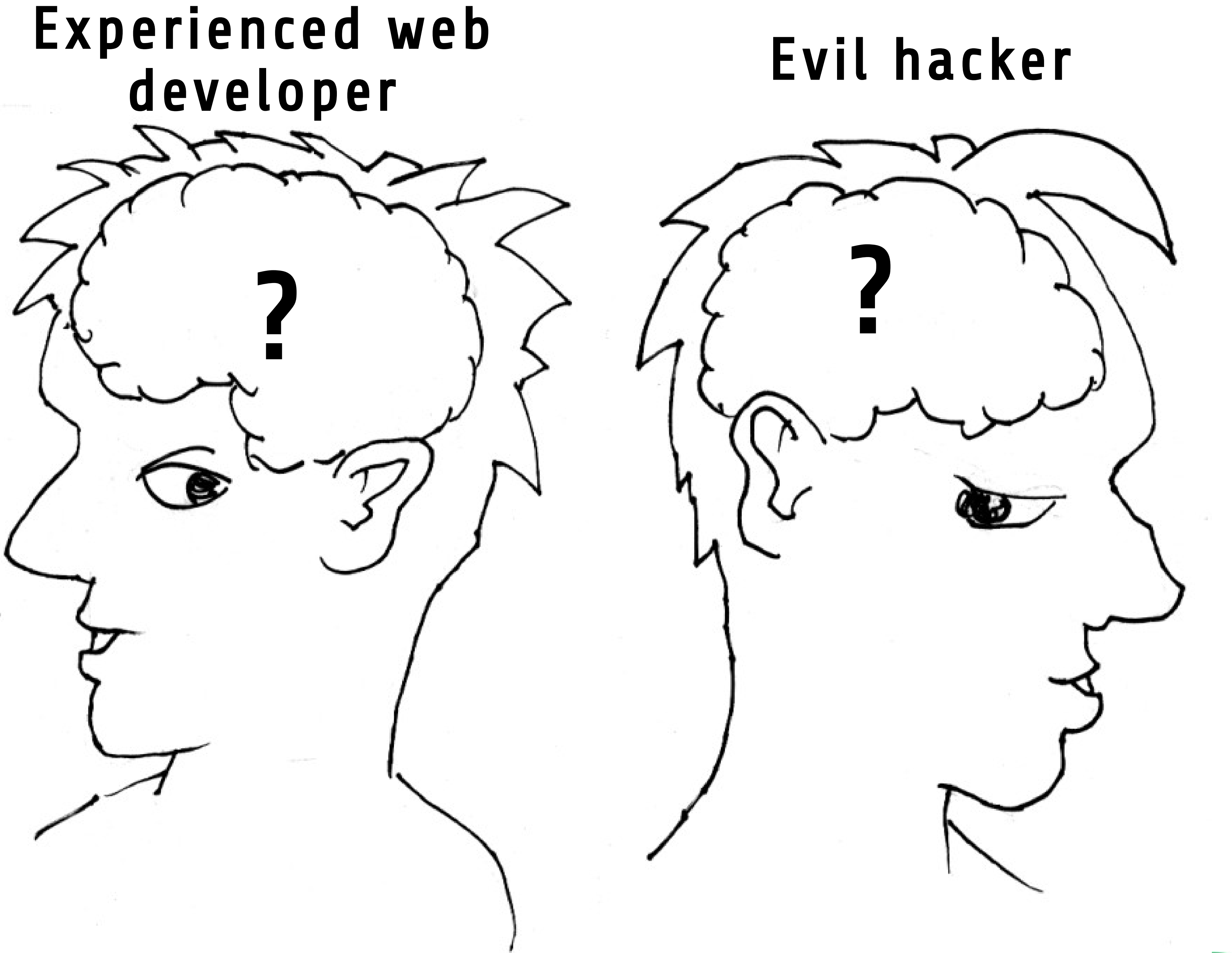
Demystifying HTTP Requests

How important is knowledge about HTTP?

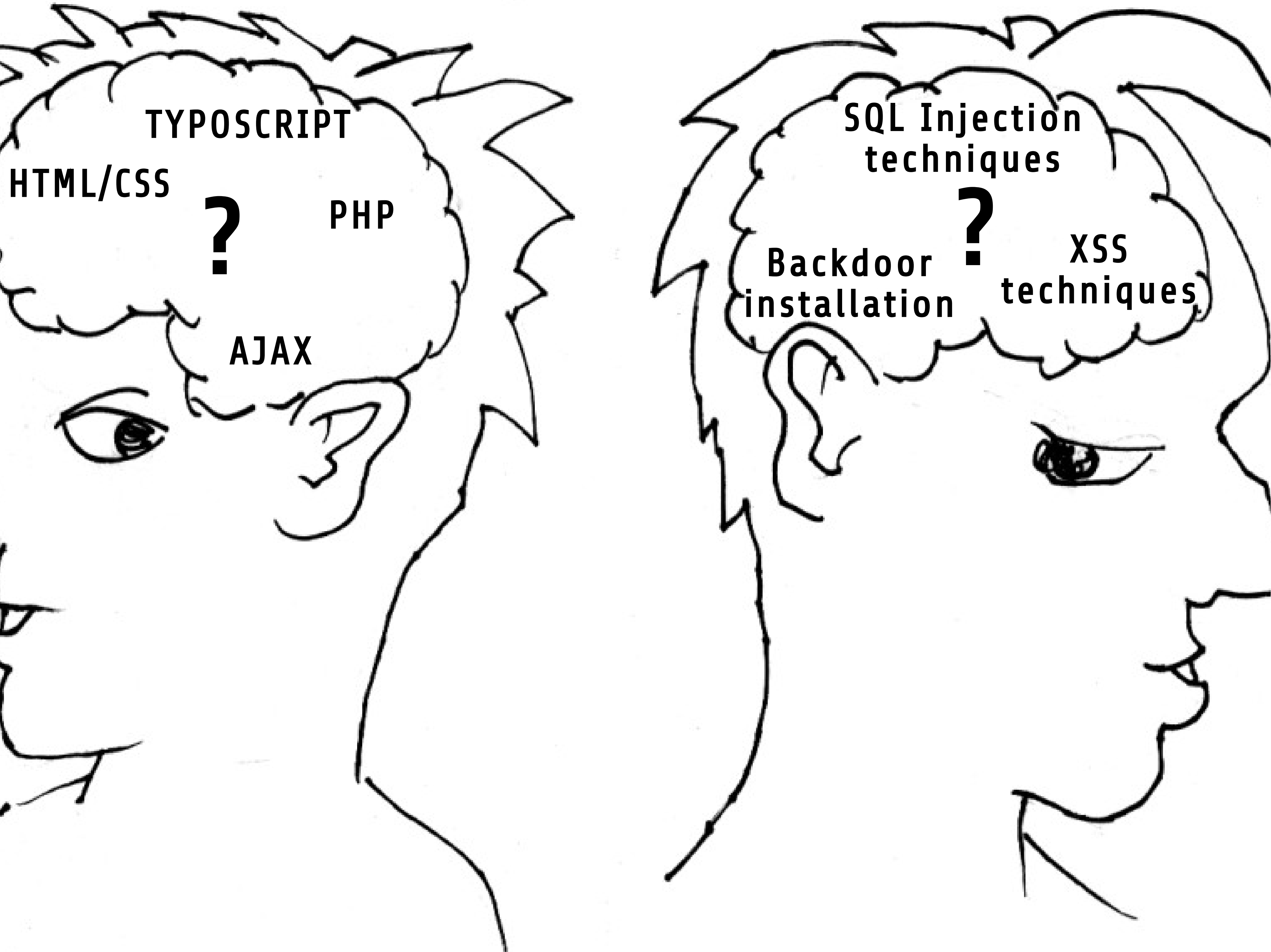
Every web server is a HTTP server...

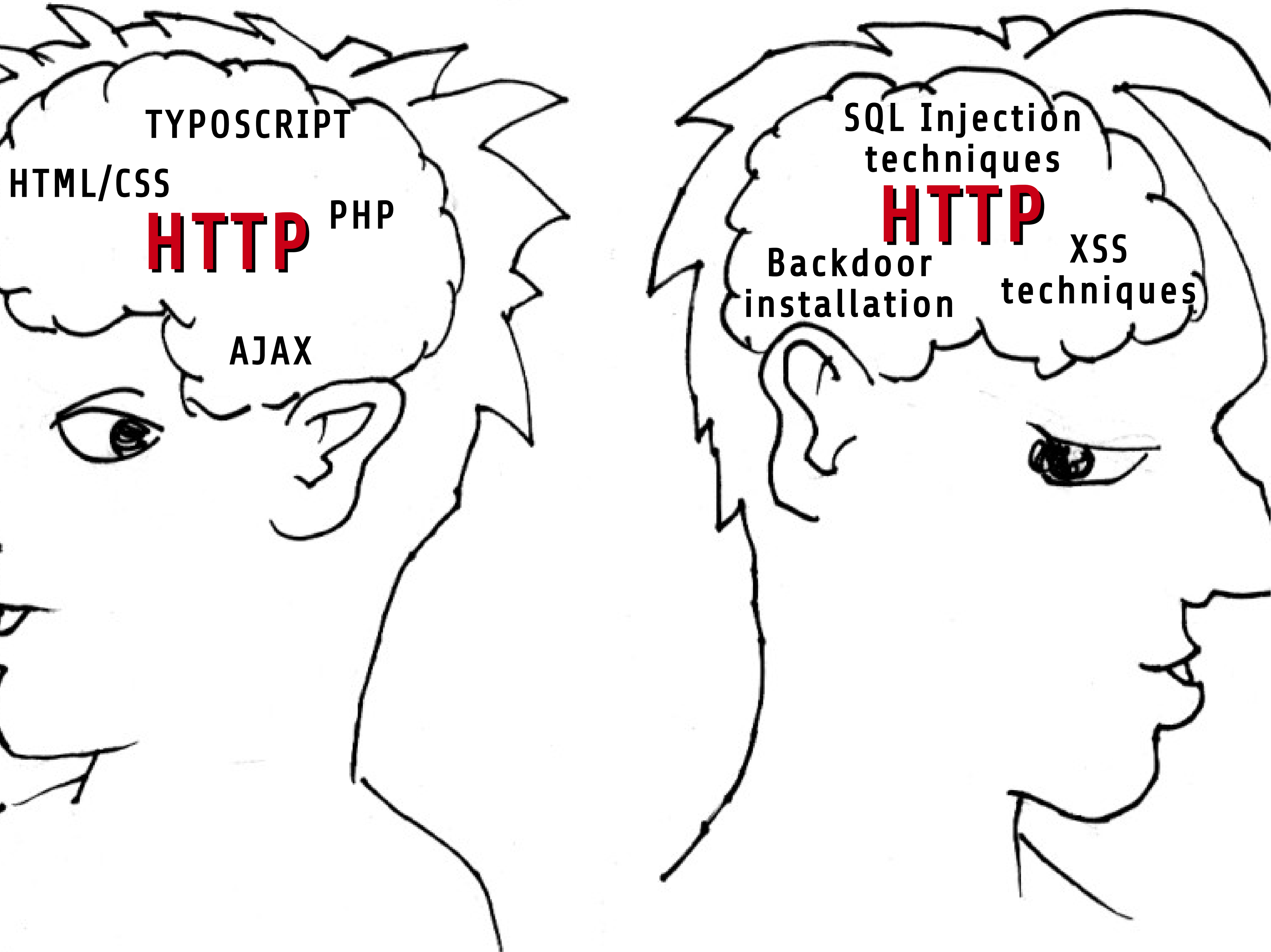
Understanding HTTP basics as the key to web application security

Demystifying HTTP means demystifying the web browser

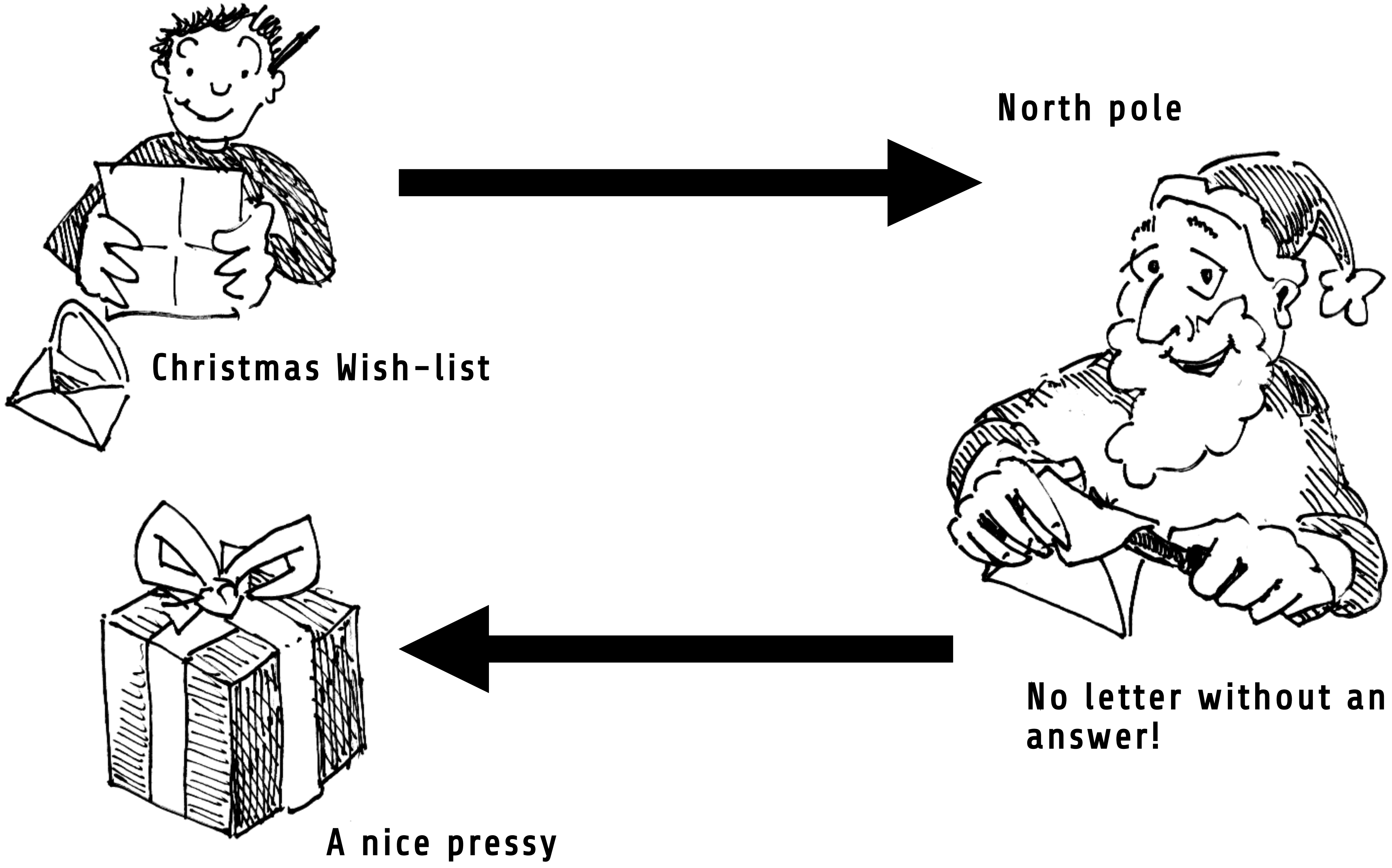


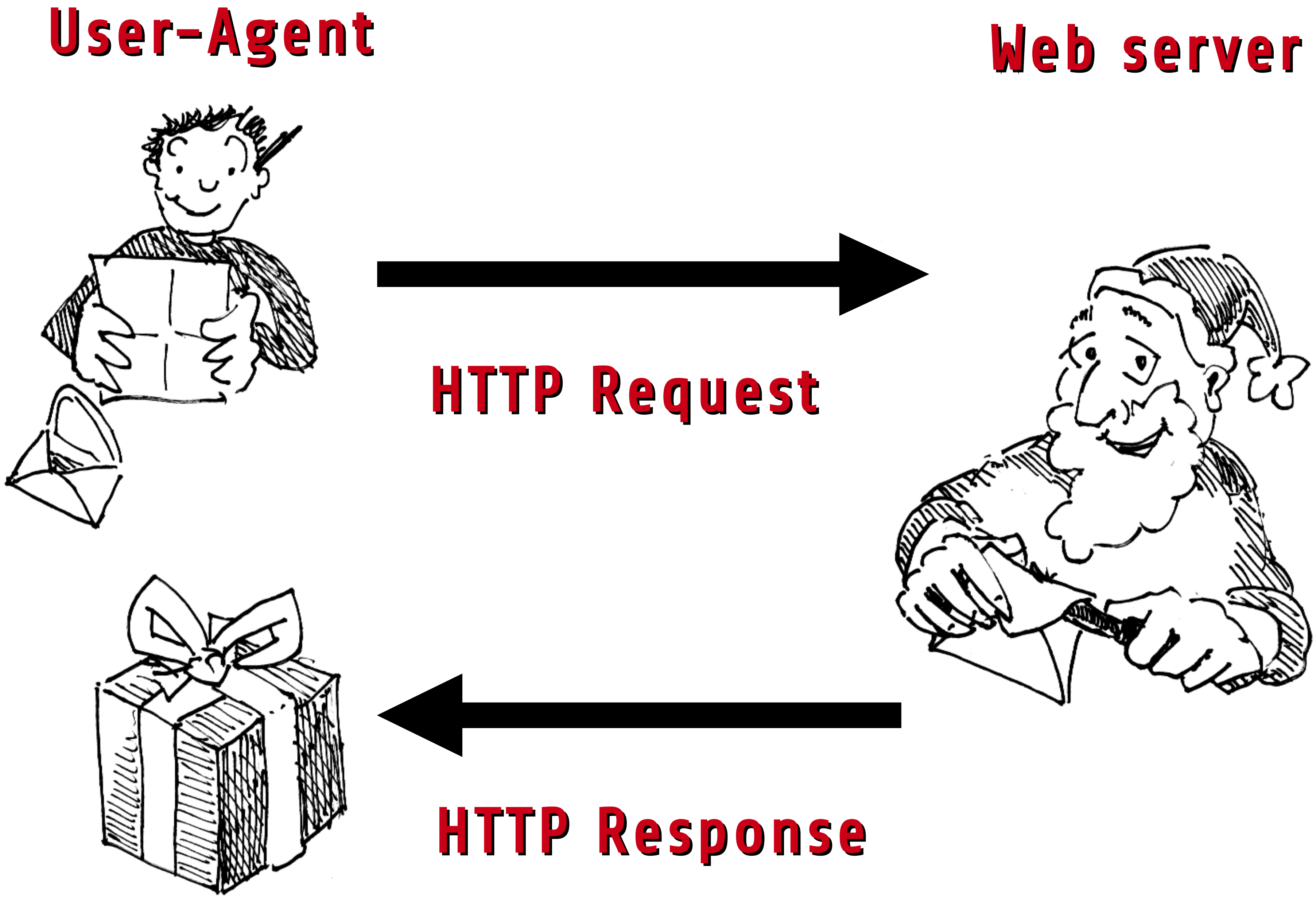
TYPO3


TYPOSCRIPT
HTML/CSS
?
PHP
AJAX
SQL Injection techniques
?
Backdoor installation
XSS techniques


TYPOSCRIPT
HTML/CSS
HTTP
PHP
AJAX
SQL Injection techniques
HTTP
Backdoor installation
XSS techniques

The Christmas Wish-list Metaphor




User-Agent
Web server
HTTP Request
HTTP Response

Talking to a Web Server

- Common HTTP request types
  - GET
  - POST
- Differences between GET and POST?

A Short (but valid) GET Request

```
GET /go/dummy-4.2.2/ HTTP/1.1↵
Host: localhost↵
↵
```

Contains information:

- Request type (GET)
- Absolute path of URI (/go/dummy-4.2.2/)
- Protocol version (HTTP/1.1)
- Network location (Host: localhost)

Important: End of request header is marked by double CRLF (↵).

The Most Basic POST Request

```
POST /go/dummy-4.2.2/index.php?id=1&no_cache=1 HTTP/1.1↵
Host: localhost↵
Content-Type: application/x-www-form-urlencoded↵
Content-Length: 70↵
↵
user=henning&pass=ddd&submit=Login&logintype=login&pid=2
&redirect_url=↵
```

Content-Length must exactly match the postvar string length.

Alternative Content-Type „multipart/formdata“ is being ignored here.

Further Common Request Fields

- User-Agent
- Referer
- Cookie

Caution: These values can be freely defined by the "instance" that creates the HTTP request.

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; de;
rv:1.9.0.3) Gecko/2008092510 Ubuntu/8.04 (hardy)
Firefox/3.0.3↵
Referer: http://localhost/go/dummy-4.2.2/index.php?
id=1&no_cache=1↵
Cookie: fe_typo_user=4c29c2fc83285e658569921fa91cf46e;
be_typo_user=e83066ebd9a71f2f13e0c17990d3cc2e;
PHPSESSID=cb689e18617bdc9ec5cbd77dd8992451↵
```

HTTP Tools for Firefox

- Extensions for monitoring requests
  - FireBug (output looks nice, but is often not complete)
  - LiveHTTPHeaders (output looks not very nice, but Is reliable)
- For other browsers:
  - MSIE: HTTPWatch
  - Safari/Webkit: Built-in Web Inspector

Hand Crafting a HTTP Request Using Ancient Technologies

- Why using telnet?
  - Easy way to communicate via TCP
  - Available on every platform
  - Plain text usage
  - Can't get any simpler
- Alternative on Windows: PuTTy

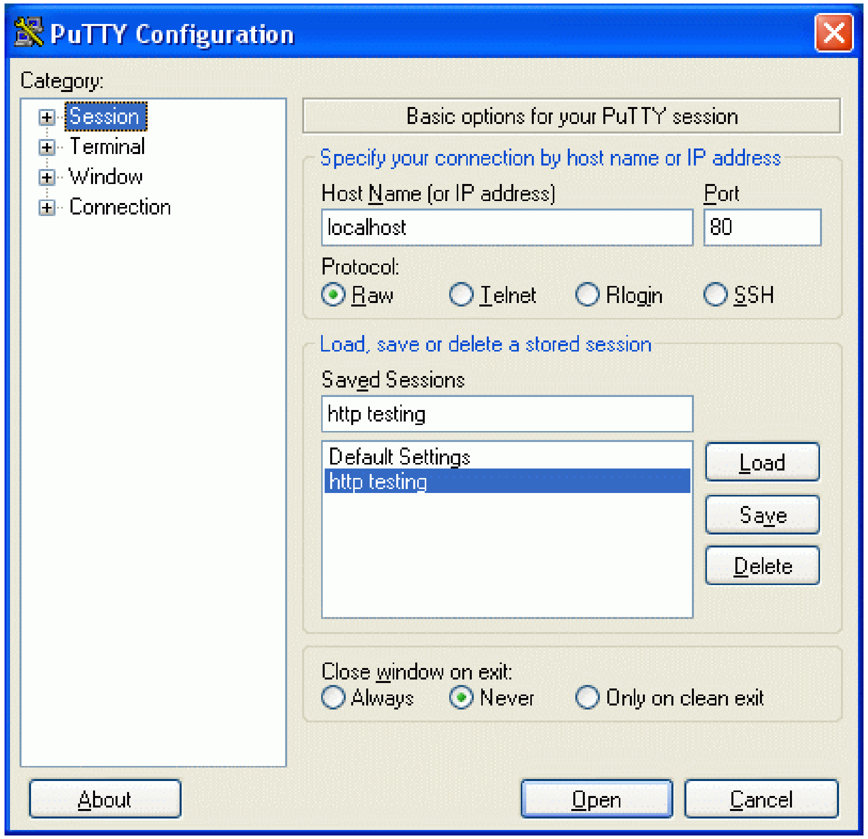
Examples

Demo

DIY Phase

- The USB stick data contains a PDF with short instructions.
- Alternatives:
  - Analyse the source code of extension ftpbrowser
    - Learn about register_globals if you are not familiar
    - Find the PHP include script which contains the security holes
  - Craft HTTP requests yourself using telnet (or PuTTY)
    - Task 1: Install LiveHTTPHeaders and use it to copy request data.
    - Task 2: Create a simple valid GET request (no “Bad request” response)
    - Task 3: Create a simple valid POST request

On Windows: Use PuTTy



PuTTY Configuration

Category:
- Session
- Terminal
- Window
- Connection

Basic options for your PuTTY session

Specify your connection by host name or IP address

Host Name (or IP address): localhost
Port: 80

Protocol: [✓ Raw] Telnet Rlogin SSH

Load, save or delete a stored session

Saved Sessions: http testing

Default Settings
http testing

Close window on exit: Always [✓ Never] Only on clean exit

- Putty.exe is on the USB stick, just start it
- Download: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Grab a Coffee!

We will continue with the tutorial at 11:45.

What Did We Learn?

- HTTP server has no way to enforce specific user agent.
- No way to hide sensitive data by GET or POST (besides HTTPS).
- There is no intimacy between the web browser and the web server. Nothing we can't to ourselves.

Impact of Unsanitized User Input

What is user input?

Now we know:

Nearly the complete HTTP request can be seen as user input

PHP Superglobals

- $_GET
- $_POST
- $_REQUEST (configurable mixture)
- $_FILE
- $_COOKIE
- $_SERVER

The Evolution of Superglobals

Extension	tslib_pibase::piVars (selection with prefix)			t3lib_div::getIndpEnv()
TYPO3 Core	t3lib_div::_POST()	t3lib_div::_GET()	t3lib_div::GP()	
PHP	$_POST	$_GET	$_REQUEST	$_SERVER

All values on t3-level are unescaped! No matter how the PHP setting magic_quotes is configured.

We have to validate and escape them.

Methods of Validation

- Cut out line breaks (where not acceptable)
- String length checks (too long?)
- Type checks (intval())
- Regex based checks
- htmlspecialchars()
- mysqlrealescapestring() / fullQuoteStr()
- Checks agains lists (whitelist/blacklist)
- Check array's for unexpected cuckoo's eggs (values + Keys – in case of dynamic key generation)

Sanitization on TYPOSCRIPT Level

- Sanitization of user supplied content
  - when rendered into HTML page via **getText**
  - when used in SQL queries using **CONTENT** object
- stdWrap offers
  - htmlSpecialChars
  - intval
  - removeBadHTML

Vulnerability Types

...that are created by unsanitized user input

SQL Injections

- Prerequisite: Unsanitized user input is used inside of an SQL query string
- Sanitization methods:
  - mysqlrealescapestring() for strings
  - Intval() for integer values (commonly id's)
  - If possible, check if integer value is in an invalid range (for example negative integer values)
- Impact: “Single line catastrophe”

Code Execution / Remote or Local File Execution

- Prerequisite: Unsanitized user input is used
  - inside of an exec() or an eval() statement or similar.
  - Inside of an include() or require() statement.
- Sanitization methods:
  - Don't do this: I can't imagine a situation where this has to be done.
  - White lists of allowed commands
- Impact: “Single line catastrophe”

Path Traversal ("../../../")

- Prerequisite: Unsanitized user input is used to create the path to a file in the file system.
- Sanitization methods: Check path.
- Impact: Access to arbitrary files and file content

Cross Site Scripting (XSS)

- Prerequisite: Unsanitized user input is inserted into the HTML page
- Sanitization methods:
  - htmlspecialchars()
  - RemoveXSS (introduced to core with TYPO3 4.2)
  - BBCode
  - If possible, check if integer value is in an invalid range (for example negative integer values)
- Impact: Cookie theft / Session hijacking possible, other dangerous stuff

Open Redirects

- Prerequisite: Unsanitized user input of type “URI” is used inside of a generated HTTP response header.
- Example: `header('Location: ' . $uri);` or a similar way of redirection (meta tag)
- Sanitization methods:
  - Check URI against user definable white list
- Impact: Inexperienced user can be “hijacked” to a different web site.

CRLF Injection

- Prerequisite: Unsanitized user input is used to generate a dynamic HTTP response header.
- Sanitization methods:
  - Check values against user definable white list
  - Prevent double Linefeeds (Newer PHP versions do that already in function **header()**)
- Impact: Various. Forcing user actions that the user is not aware of.

Check Reliability of Sanitization Methods by Tests

How? Let's discuss.

Visit the tutorial of Oliver Klee

to learn more about unit testing.

Security Incident Handling

- Contact us: security@typo3.org
- Case 1: You have found an issue in your own extension.
- Case 2: You have found an issue in somebody elses extension.
- TYPO3 Security Team policy

Thank You!