Lecture Notes for Web Security 2019

Part 2 — Apache and PHP Security, Regular Expressions

Martin Hell

1 The Apache Web Server

Apache is the most common web server in use, hosting about 60% of all web sites in July 2012 [1]. While there are several implementations of web servers, this widespread use of Apache is one motivation why Apache will be used as example in this discussion about web server security. This section will not provide a complete introduction to the Apache web server as many aspects will be ignored. For a thorough overview, refer to `http://httpd.apache.org`. Instead, this section will only discuss some selected topics related to security.

1.1 Apache Configuration

The main configuration file for Apache is usually `httpd.conf`. This file is used to define ports to listen to, which directory to use as root directory for requests to the server, which additional modules to load etc. Both global configuration for the server and local configuration for virtual hosts and specific directories are specified. The file is read when the server starts and any changes require a restart of the server. The `Listen` directive is used to tell the server to listen for incoming connections on a specified port. The `ServerRoot` directive specifies the home directory of the server, i.e., where configuration files and log files are kept. The root directory seen by user agents is specified using the `DocumentRoot` directive. If this is defined as /var/www, then a request to www.example.com/index.html will refer to the file /var/www/index.html.

Directives in the configuration files can be limited to only certain requests using a *configuration section container*. A section container can be used to match a request to e.g., a specific directory, file or location. Directories and files are used to match requests to specific parts of the filesystem, as seen by the server, while location is used to match requests to resources as seen by the user agent, or web browser. Section containers are specified using `<...> ... </...>`. An example is given below.

```
ServerRoot /etc/Apache
Listen 80
DocumentRoot /var/www
<Directory /var/www/dir>
  # directives applicable to files in
  # /var/www/dir directory
</Directory>
```

Using `<Directory ~ >` PCRE regular expressions (see Section 3) can be used to match a directory to apply the directives to.

1.1.1 Distributed Configuration

An alternative to using `<Directory>` in the main configuration file is to put directives in a distributed configuration file inside a directory. The default name for these files is `.htaccess` but it can be changed using the `AccessFileName` directive. A `.htaccess` file with configuration directives will apply to that directory and all subdirectories. The directives are applied, and possibly override directives in the main configuration file, if this is permitted according to the `AllowOverride` directive. An important difference between the main file and the `.htaccess` file is that changes to the latter are immediately applied, without requiring a restart of the server.

Distributed configuration with `.htaccess` files should only be used if there is no possibility to use the main configuration file, e.g., if users want to change the configuration and only the administrator has access to the main file. Anything that can be put in a `.htaccess` file can also be put in the main file. Using distributed configuration has (at least) two important disadvantages compared to using the main configuration file. First, every time a file is requested, the server will look for `.htaccess` files, both in the requested directory and its parent directories. Looking for, and loading, these files for every request will slow down the server. The second reason to avoid distributed configuration files is that it will allow users to make changes to the server, which is a potential security threat. An administrator has to be very careful determining which configuration settings should be allowed to be overridden. Added complexity makes more room for mistakes.

1.1.2 Configuration Order

When the configuration of a resource is determined, the sections in the configuration files are read in a specific order. It is important to understand the order in which information is read so that the configuration is as expected. The order is given as follows.

1. The <Directory> section and `.htaccess` files are read simultaneously, with `.htaccess` possibly overriding <Directory>. The order is given by the length of the directory component, starting with the shortest. If there are

duplicates, the order is given by the order in which they appear in the configuration file.

2. The <Directory> sections using regular expressions and <DirectoryMatch> sections are read in the order they appear in the configuration file.

3. The <Files> and <FilesMatch> sections are read in the order they appear in the configuration file.

4. The <Location> and <LocationMatch> sections are read in the order they appear in the configuration file.

Subsequent information updates/overwrites previous information and after processing all information, the resulting configuration is applied to the resource. If virtual hosts are used, the sections inside <VirtualHost> are applied after those outside, for each item above.

1.1.3 Server Reporting

By default, the web server sends the `server` HTTP response header, specifying which version of the server is used and other information such as the version of PHP and MySQL and which operating system is used on the server. Sending this information to user agents is not a security problem in itself. However, if the server software is rarely updated and patched, known security weaknesses can be used and exploited. Giving an attacker specific information about version numbers will greatly simplify the process of attacking the server. In Apache, the directive `ServerTokens` is used to control what information is given to clients. The default value is "full", which provides the client with the maximum amount of information. Other possibilities are given below together with an example of the information sent.

ServerTokens Prod	Apache
ServerTokens Major	Apache/2
ServerTokens Minor	Apache/2.2
ServerTokens Min	Apache/2.2.14
ServerTokens OS	Apache/2.2.14 (Ubuntu)
ServerTokens Full	Apache/2.2.14 (Ubuntu) PHP/5.3.2-1ubuntu4.9

Still, it is important to remember that most security vulnerabilities and attacks are not based on flaws in the actual server. Instead, it is the programs written by users that contain the majority of vulnerabilities.

1.2 Authentication

Recall that authentication is the process of verifying the claimed identity of a user, or more generally, the correctness of some data. Authentication with username and password through Basic or Digest Access Authentication is supported by Apache. The default is to store the username and password information in a

textfile, but other options, such as using a database or LDAP are also possible. A minimal realization of Basic Access Authentication is given below.

```
<Directory /var/www/protected>
  AuthType Basic
  AuthName protected
  AuthUserFile /some_path/passfile_basic
  Require valid-user
</Directory>
```

The directory `/var/www/protected` is only accessible by providing username and password. The `AuthType` directive specifies that Basic is used as authentication protocol and `AuthName` specifies the realm. The `AuthUserFile` directive gives the path to the file containing the usernames and hashed passwords. If other ways of storing passwords are used, the directive `AuthBasicProvider` can be used. As this defaults to `file`, this directive is not needed here. The `Require` directive is used for authorization (see Section 1.3) and `valid-user` means that anyone that is successfully authenticated is also authorized access to the resource. It is also possible to specify individual users or groups of users. Adding users to the password file can be done using the `htpasswd` program.

Note that, since Basic Access Authentication sends passwords in plain text, this should never be used without SSL.

Configuring Digest Access Authentication is very similar. The main difference is that `AuthType Digest` is used to define the protocol. An example is given below.

```
<Directory /var/www/private>
  AuthType Digest
  AuthName private
  AuthUserFile /some_path/passfile_digest
  Require valid-user
</Directory>
```

Users are added to the password file with the program `htdigest`.

1.3 Authorization

Authorization is the process of determining if a given user has access to a resource. Authorization in Apache was significantly changed in version 2.4, deprecating the directives used before that. This section will cover both the old `order`, `allow` and `deny` directives, as well as the newer `Require` directive which is now used for all types of authorization.

1.3.1 Authorization in Apache 2.2 (and before)

The access to a resource in Apache 2.2 is controlled by `Allow` and `Deny` directives. They can be used to restrict access based on the requesting machine's host

name or IP address. These can in turn be specified using either full or partial domain names or full or partial IP addresses. An example is given below.

```
<Directory "/dir">
  Order Deny,Allow
  Allow from lth.se
  Allow from 192
  Deny from se
  Deny from 192.168.0.0/16
</Directory>
```

In the example, hosts from domains ending in .lth.se or IPs beginning with 192 are allowed access while hosts from domains ending with .se and IPs beginning with 192.168 are denied access. Note that only complete domain labels and IP address bytes are considered. Hosts from tlth.se would not be explicitly allowed by the lth.se directive. The `Order` directive defines how the access directives are read and determines the result if one host matches both or none of allow and deny. If `Order Deny,Allow` is specified, the `Deny` directives are read first, possibly overwritten by subsequent `Allow` directives. Thus, if a host matches both, it will be allowed. If it matches none it will also be allowed access. Using `Order Allow,Deny` has the opposite effect, i.e., if a host matches both or none, it is denied access. The default order is `Deny,Allow`.

1.3.2 Authorization in Apache 2.4

In Apache 2.4, the `Order`, `Allow` and `Deny` directives are deprecated and replaced by the `Require` directive. Thus, this directive is used also for authorization based on host name and IP address. Examples of using it are given below.

```
Require all granted
Require all denied
Require ip 130.235
Require host lth.se
```

Note that these can not be used together, they just show the syntax. The first two directives unconditionally grants and denies access to the resource, respectively. The third is used to require that the requester comes from the an IP address starting with 130.235 while the last requires that the host name is resolved to something ending with `lth.se`. It is also possible to negate the directive with e.g., `Require not ip 130.235`. A negated directive can only fail or return a neutral result and can thus not be used by itself to grant access to anyone. In other words, the example can not be used to grant access to hosts that has an IP address differing from 130.235 since it will not return success. To do this we need additional directives. A set of `Require` directives can be used together with three container directives to construct arbitrarily complex authorization rules.

- `<RequireAll>`. This will require that no directive in the enclosed group fails, and that at least one succeeds in order for `<RequireAll>` to return success.
- `<RequireAny>`. At least one of the directives in the group must succeed for the `<RequireAny>` to return success. If none succeeds or fails, the result is neutral, otherwise it fails.
- `<RequireNone>`. If at least one of the directives in the group succeeds, the `<RequireNone>` fails. Otherwise the result is neutral.

If several Require directives are used without being placed in a container, they are implicitly understood as being inside a `<RequireAny>` container. Using containers, it can now be seen how it is possible to restrict a specific range of IP addresses.

```
<Directory "/dir">
  <RequireAll>
    Require all granted
    Require not ip 130.235
  </RequireAll>
</Directory>
```

The directives above give access to anyone that is not using 130.235 as client IP. For 130.235 the second directive will fail, causing `<RequireAll>` to fail, while for everyone else, the first directive will succeed and the second will return a neutral result. This will satisfy the requirements for `<RequireAll>` to succeed. The `<RequireNone>` container has the same property as negated expressions and can also not be used alone.

The three container directives can be nested, creating very detailed authorization rules. However, negated `Require` directives can not be used inside `<RequireAny>` or inside `<RequireNone>` as they can have little useful impact on the result.

2 PHP

PHP is commonly used when programming dynamic websites. It can be incorporated into Apache by loading the `php5_module`, which is the most convenient way, but it is also possible to run PHP as a CGI script. Refer to one of the very many online PHP tutorials for an introduction to the language and the syntax. The PHP website `http://www.php.net` contains lots of information. This section will cover parts of the PHP configuration and common security issues that arise when programming with PHP. Some specific PHP functions will be described as they are used in examples. As PHP gives the programmer the power to dynamically interact with users and user input, great care must be taken when writing the programs. PHP itself provide programmers with many tools to secure their programs, but unless the tools are used, and used correctly, vulnerabilities are very likely to arise.

2.1 PHP Configuration

The main configuration file in PHP is `php.ini`. However, many directives can be overridden in the PHP files, using the `ini_set()` function, making it somewhat similar to the situation with `.htaccess` in Apache.

2.1.1 Server Reporting

Similar to the ServerTokens in Apache, PHP will by default send information about the fact that PHP is used and which version. Again, this is not a security problem in itself, but can be valuable information for an attacker if the server and/or PHP is not properly updated and patched. In order to hide the fact that PHP is used, setting ServerTokens to e.g., OS is not enough. By default, an `X-Powered-By` header is added to the HTTP response, specifying the PHP version in use. This information can be suppressed using the `expose_php = Off` directive. Some combinations are given below.

```
-- Combinations --          -- HTTP response headers --
ServerTokens Full          Server: Apache/2.2.14 (Win32) PHP/5.3.2
expose_php = On            X-Powered-By: PHP/5.3.2

ServerTokens Full          Server: Apache/2.2.14 (Win32)
expose_php = Off

ServerTokens OS            Server: Apache/2.2.14 (Win32)
expose_php = On            X-Powered-By: PHP/5.3.2

ServerTokens OS            Server: Apache/2.2.14 (Win32)
expose_php = Off
```

Setting `expose_php` to `Off` will suppress the information also in the `Server` header.

2.1.2 Register Globals

In PHP, unassigned variables will always default to false. Thus, if a variable is used, e.g., in an if-statement, without having been initialized, it will always take the value false. This fact can be used by programmers, since is not necessary to explicitly assign false to a variable before it is used. If the configuration directive `register_globals` is set to `On`, then it is possible to assign variables through GET, POST, Cookies, environment variables and server defined variables. Links or forms can be constructed such that variables are assigned in the target PHP script. While this could be useful in some circumstances, it is also a security threat since anyone can set the variables to any value by constructing their own requests. Consider the following PHP script.

```
function authenticate_user() {
  ...
}
if (authenticate_user()) {
  $auth=true;
}
if ($auth) {
  echo "sensitive data...";
}
```

A function is used to authenticate a user. It returns true if authentication is successful, and the if-statement will set the global variable `$auth` to `true`. If the user is not authenticated, `$auth` is not initialized and will default to `false`. In the last step, sensitive data is returned to the client if `$auth` is `true`, i.e., if the user was successfully authenticated. This program would work, but does not take into account that the global variables can be set in e.g., a GET requests. If a user submits the GET request

```
GET /script.php?auth=1 HTTP/1.1
```

then the variable will be initialized to 1 (true) and the sensitive data will be returned in the response even if the user is not authenticated. This vulnerability can be avoided in two ways. First, following good programming practice, all variable should be initialized before they are used. Initializing `$auth` to `false`, would remove the vulnerability as this would overwrite the value sent in the request. Second, PHP can be configured to disallow initializing variables through e.g., GET, POST and cookies using the `register_globals = Off` directive. In fact, since PHP 4.2.0, the default value of this directive is `Off` so it must be explicitly turned on if the functionality is required. Moreover, it has been deprecated since PHP 5.3.0 and its use is highly discouraged. In PHP 6 it will be completely removed. Still, the potential severity of the related vulnerabilities motivates that users are aware of this problem.

2.1.3 Error Reporting

Error reporting is very useful during the development phase. It helps the developer to locate problems when the applications are not executing as wanted. However, once the application is in production, errors reporting should be turned off. Errors can give valuable information to an attacker, e.g., file paths, file names, uninitialized variables, and arguments to functions, which in the worst case could include passwords to databases used. Error reporting is controlled in `php.ini`. The directive `display_errors` specifies if errors should be displayed on the screen. This defaults to `On` but should be turned off in production stage. Instead, errors should be logged to a file. This can be done by setting `log_errors = On` and specifying the file to log to using the directive `error_log`.

3 Regular Expressions

A regular expression (regex) provides a way to match a string to text. They can be used for many purposes other than only security related. When searching for a specific pattern in a text, regular expressions is a flexible and powerful alternative to just searching for the literal string. When validating user input, regular expressions can be used to check if the supplied data obey a certain set of rules. This section provides a short introduction to regular expressions. For a more in-depth tutorial, refer to `http://www.regular-expressions.info/`.

Regular expressions come in several flavours, and there are also many different implementations available, resulting in small differences depending on where it is used. POSIX basic regular expressions (BRE) and POSIX extended regular expressions (ERE) have been standardized in order to provide compatibility. The UNIX command `grep` implements both flavours, providing `grep -E` for ERE, while BRE is default. The UNIX command `egrep` is the same as `grep -E`. ERE provide more functionality than BRE and is not fully compatible with BRE syntax. Still, ERE is rather limited compared to modern regular expressions variants, e.g., Perl Compatible Regular Expressions (PCRE). PCRE is an open source library which implements the regex syntax used in Perl 5. Many modern implementations provide similar functionality as these implementations. The implementation used in the PHP `preg` functions are based on PCRE. ERE support exists in PHP using the `ereg` functions, but is deprecated since PHP 5.3.0. This introduction will primarily be focused on PCRE, but as only the basic syntax is described, it is applicable to most modern flavours.

The most straightforward way of using a regex is to match with a literal string or character. The regex `abc` will match the first occurrence of the sequence "abc" in a string, similar to a plain string search. However, this does not take advantage of the power and flexibility in regular expressions. Special characters, or metacharacters, are used to provide more functionality. Any special character that should be interpreted as a literal character must be escaped using a backslash. One special character is the dot ".". A dot can be seen as a wild card that matches any character, except the newline character "\n". Thus `a.c` will match the first occurrence of an "a", followed by any character, followed by a "c".

Character Classes, [ ]
Using a character class, the regular expression can define several different characters and match either one of them. The character class is specified using square brackets metacharacters, so `[ab]` matches the first occurrence of either "a" or "b" in a string. A range of characters can be specified using a hyphen. The regex `[a-z]` will match any lower case letter and `[a-zA-Z0]` matches any upper or lower case letter or the digit "0". The character class can be negated, meaning that any character except those in the class will match. This is accomplished by placing a caret immediately after the opening square bracket, e.g., `[^aA]` will match any character except an "a" or "A". A metacharacter inside a character class does in general not have to be escaped, even though it is permit-

ted. Those that do have special meaning, i.e., `^`, `]`, and `-` can either be escaped or placed where their literal meaning can not be confused by their meaning as a metacharacter, e.g., immediately after the opening bracket. As an example, `[-a-z0-9]` is equivalent to `[a-z0-9-]` and `[a-z\-0-9]`. Note though, that a backslash must always be escaped in a character class. There are shortcuts for some common and useful character classes. The shortcut `\d` matches any digit, `\s` matches any whitespace and `\w` matches any word character, i.e., a `\w` is the same as `[a-zA-Z0-9_]`. The complements to these classes are defined by `\D`, `\S` and `\W`. Thus, `[a-zA-Z0-9]` is the same as `[^\W_]`.

Alternation, `|`

Two or more regular expressions can be used for a match by combining them into one expression. It has the same meaning as a binary OR as only one of them has to match in order to get a match for the regex. It is similar to character classes but instead of matching one out several character, one out of several expressions are matched instead. The separation is done using a vertical bar. This operator has the lowest precedence of all operators, reflecting the fact that complete expressions are separated, not just parts of one expression. In order to match either "one" or "two", the regex `one|two` can be used.

Quantifiers, *, +, ?, {n,m}

A quantifier is used to determine how many times the preceding character, or group of characters, should be present for a match. The following table gives the possible ways of specifying this.

Possible Quantifiers	
$*$	Match 0 or more times
$+$	Match 1 or more times
?	Match 0 or 1 time
$\{m\}$	Match exactly m times
$\{m,\}$	Match at least m times
$\{m,n\}$	Match a minimum of m times and a maximum of n times

The regex `A[bB]*C+[0-9]{1,2}` would match e.g., "ABC11", "AC2", "AbCC3" etc. This regex can equivalently be written as `A[bB]{0,}C{1,}[0-9]{1,2}`. By default, the search engine is greedy, meaning that it will try to match as many positions as possible if quantifiers are used. The previous expression would match "ABC11" even though it could have chosen to match "ABC1" instead as only one digit was required. In order to make the search lazy instead of greedy, a `?` is added after the quantifier. The regex `A[bB]{0,}C{1,}[0-9]{1,2}?` would then match "ABC1" instead of "ABC11".

Anchors, `^`, `$`

Anchors are used to denote the beginning and end of a string. A `^` is used to match the beginning of a string while a `$` is used to match the end. Thus, they will not match any specific characters, but instead the space before and

after the first and last character respectively. This is in particular useful for validation of user input, since the purpose is then to verify the complete input provided by users. Using `[0-9]+` is not enough to validate that user input is a number, since it will find a match also if there is a number together with other characters. Instead `^[0-9]+$` must be used in that case. If a string consists of several lines, the anchors will by default represent the beginning of the first line and the end of the last line respectively. For anchors to represent beginning and end of each line, multi-line mode has to be used.

Word Boundary, `\b`

One other notable and useful sequence is `\b`, which represents a word boundary. This is useful to denote the start and end of a word. It is defined as the position where the current and previous character does not both match a word character or both match a non-word character, i.e., one character matches `\W` and the other matches `\w`. The regex `\bbanana\b` will not find a match in the string "bananas" but it will find one in the string "banana!". The complement `\B` can be used to match a non-word boundary. The result of regex `\bbanana\B` would be the opposite to that given above.

Modifiers

Modifiers can be used to tell the regex engine to interpret the regular expression or the string it is applied to in a specific way. Common modifiers are `i`, `m` and `s`. The modifier `i` treats the regular expression as case insensitive. Both upper and lower case letters will match. The modifier `m` treats the string as multiline. The anchors will match start of line and end of line respectively, instead of the default where they match only start and end of string. The modifier `s` treats the string as single line. This has the effect that the dot operator will also match a newline, which is otherwise the only character it will not match. This has no effect on anchors and should not be confused with the effect of the multiline modifier. Another common modifier is `g` which is short for global. This is often used when the regex is used for replacement in order to apply the replacement to all matches and not just the first. In PHP, this modifier does not exist and instead the function to use, or arguments to the functions will control this parameter.

Exercises

Exercise 201 *Consider the following excerpt from* `httpd.conf` *and a* `.htaccess` *file:*

```
-- httpd.conf --
<Directory /var/www/private>
  AllowOverride all
  Require all denied
</Directory>

-- .htaccess in /var/www/private --
Require host lth.se
```

a) Who will have access to the `/var/www/private` *directory on the server?*

b) If `AllowOverride none` *is used instead of* `all`*, what would be the result?*

c) If the container

```
<Directory ~ /var/www>
  Require all granted
</Directory>
```

is added in `httpd.conf`*, what would be the result?*

d) Returning to the original ***`httpd.conf`*** *configuration, what would be the result of replacing the* ***`.htaccess`*** *entry with the following directives?*

```
<RequireAny>
  <RequireAll>
    Require all granted
    Require not host spammer.se
    Require not ip 130.235
  </RequireAll>
  Require ip 130.235.1.1
</RequireAny>
```

Exercise 202 *How does the AuthName directive change the communication between the client and the server. Does the particular choice of AuthName have any impact on security?*

Exercise 203 *By default, Apache sends information about server version, operating system and PHP version in a HTTP response header. How can this information be controlled by an administrator?*

Exercise 204 *Construct a regular expression for checking that a string is a URL.*

Exercise 205 *Construct a regular expression for checking that a string is a number divisible by* 2*.*

References

[1] Netcraft. July 2012 web server survey, July 2012. Available at: *http://news.netcraft.com/archives/2012/07/03/july-2012-web-server-survey.html*.