Växjö University

School of Mathematics and Systems Engineering

Reports from MSI - Rapporter från MSI

The PHP programmer's guide to secure code

Richard Clarinsson

Samuel Magnusson

Maj
2005

MSI
Växjö University
SE-351 95 VÄXJÖ

Report 05046
ISSN 1650-2647
ISRN VXU/MSI/IV/E/--05046/--SE

Abstract

Authors: Richard Clarinsson, Samuel Magnusson
Title: The PHP programmer's guide to secure code
Course: IVC730 – Bachelor thesis
University: Växjö Universitet
Institution: School of Mathematics and Systems Engineering
Date: 2005-05-24

Content:

Security threats against computer systems are a big problem today which also includes PHP made applications. The report is focused on protection with the help of code and not how you protect a web server. Its purpose is not to educate the readers of the thesis how to make a PHP application, the purpose is how to program a safer PHP application.

The thesis contains information about common security threats against PHP scripts. It contains in most cases examples of what an attack can look like and how a protection for that example can be achieved. We have tested all code examples if they work by installing our own server with the configurations according to the delimitations of the thesis and putting up small PHP applications, which we have attacked and then protected.

The contents and result of this thesis can benefit developers that use PHP as a programming language for creating web applications, by giving them information about common threats and protection.

Keywords: security, PHP, security threats, programming, code, protection

Preface

This thesis has been very interesting and educational to write, but also a challenge. It has taken a lot of time and effort to write leaving us with very little free time, but it has been worth it.

We want to give a special thanks to our tutor *Jonas Werner* who have help us in troubled times.

Read and enjoy!

Richard Clarinsson & Samuel Magnusson
2005-05-09 Växjö, Sweden

Table of content

Table of figures

Table of examples

1 Background

The web programming language PHP is used by a lot of developers today, which means that a lot of web applications are made in PHP. There is a threat against PHP applications because it is not that hard to perform an attack if the programmed application is not protected.

We both have an experience of programming PHP, but the security part with all threats that lures out there and how you protect your script has been blurry for us.

Our background with PHP is that we both enjoy making web application programmed in the language. We have also studied courses about PHP on a University level, but a very small part of what have been taught in those courses is about web application security. We believe that security is very important and our interest of it has increased during the time.

We have read articles about how insecure some well used applications are. We have seen hacked websites which have been exploited by attacks. We have talked to web developers that do not know much about security or cares about it at all. Because of this, our impression is that security is not prioritized or cared about. We do hope that our impressions will change in the future and that security thinking will be more prioritized.

1.1 Purpose

The purpose of this bachelor thesis is to find out how you should write secure PHP code in a way that a hacker can not exploit. The purpose is also to get information about what common security vulnerabilities a PHP script could contain. The information collected will be put together and should be looked upon as a guide to secure PHP code.

Our goal with the effect of this report is that PHP programmers can get information about how to program their PHP scripts in a way that hackers can not or will have a hard time to exploit.

1.2 Target group

This bachelor thesis is written towards PHP programmers with basic knowledge or greater and who have an interest in PHP programming and security.

1.3 Problem discussion

PHP is a flexible language which gives a lot of possibilities but also a lot of vulnerabilities. To create a working script in PHP does not require an enormous expertise but to create a secure one requires knowledge in the subject. This could mean that a lot of websites on the Internet are not secure from persons who exploit the security weaknesses in PHP scripts.

1.3.1 Problem formulation

What can you do code-wise to protect your PHP script from common security threats?

1.4 Delimitation

This report is written towards PHP programmers with basic knowledge or greater about PHP. The report will not explain common code or teach how to program PHP in general. It is focused on the code that concern security and not on the server side configuration. Server side configurations are important to security and cannot totally be ignored here, this report is build on one set of configurations that are not the ultimate security configuration so that we have to make the system secure with PHP code instead.
Different databases can be used with PHP. This report is focused on the common database MySQL. No other databases will be taken in consideration.

These are the important configurations that the report is based on:

Server:
Apache version 2.0.53, PHP is installed as an apache module.

PHP version:
4.3.11

Database:
MySQL 3.23.42

PHP configuration:
register_globals: ON
magic_quotes_gpc: OFF
magic_quotes_sybase: OFF
magic_quotes_runtime: OFF
file_uploads: ON

1.5 Scientific methods

1.5.1 Research approach

This is an *explorative* (R. & B. Davidsson 2003, p. 12) investigation where we want to light up different aspects of PHP vulnerabilities and describe this to give collected view of common PHP threats. We are not only describing and discussing the problems of different PHP vulnerabilities, we are also testing and verifying the different vulnerabilities found. The tests are made on an apache web server set up by us

The subject that we study is in its nature very logic. We want to find common security vulnerabilities in the PHP language and how they can be eliminated. On the base of the facts found in our sources and our own testing, it is possible to validate our assumptions in a logical way. We are using empirical studies to find our vulnerabilities, logically trying these vulnerabilities in our test environment and finding ways of eliminating them by logical trying. We are using a *positivistic* view in this sense, since we use logical conclusions to gain a result and we working with very "hard" variables, since it is possible to try if the vulnerabilities exists and also if there is a possible solution (T. Thurén 2002, p.15).

We will use or base understanding of PHP and take this a step further, not only focusing on the PHP commands and the functionality itself, we will focus more on how to use this functionality to create secure applications and not only to create an application.

1.5.2 Usage of sources

We are using secondary sources of information for our investigation. The information is found on different web pages and in literature. Derived from the information found, we will describe, test and analyze possible threats against a PHP script and how to eliminate them. The whole or parts of the script created of us is presented in this thesis. To present the entire script in all cases would be too extensive, but we will present enough to give a general understanding of the problem areas and their solution.

1.5.3 Critical assessment of the sources used

The materials we use relies much on web pages where much of the samples and information about vulnerabilities are found. The information used is taken from web pages specialized on this subject area. The selected sources of information have a purpose to inform programmers in PHP, how to make it in a safe and good way. For example, the site *www.phpsec.org*, from where much information is gathered, have no business reasons to angle the information in a certain way, their target is to increase the knowledge about security in PHP. The fact that most of the sources are independent from business influences increases the reliability of them (R. & B. Davidsson 2003, p. 65).

The information we have found is not published just once at one place. It is possible to find the information used in this report on other web pages or in literature than the source used

here. The fact that it is published in different places by different authors increases the reliability of the information.

1.5.4 Why we have chosen the threats written in this thesis

To get the understanding of what threats that is most common or most vulnerable, we have looked for sources writing about security threats in PHP. We have studied these sources, what threats they describe and why these are vulnerable. We have used the information gathered to present what we think are the most common security threats to a PHP application that can be protected with code.

1.5.5 Testing the security solutions

The example solutions that are given in this thesis are tested by us. A web server has been set up and PHP has been installed on the server. The examples that we have given are in most cases inspired by solutions from other sources but been modified and been put in a little different context. We have made working PHP scripts with these examples and we have tested them if they work and if they are secure with the help of our own knowledge and the information written about security threats in this thesis. To find out if they are secure, we have made attacks according to the information we gathered about how to perform an attack. All examples are in our understanding secure against the attacks we made on them.

2 Theory

2.1 Security

Security is events that decrease the risk for unwished actions or accidents to occur. High security means low risk for accident or unwanted actions to occur. (http://www.ne.se/jsp/search/article.jsp?i_art_id=322447&i_word=s%e4kerhet, 2005-03-30).

IT security is protection of the data within an IT system. The word covers everything from communication, storing and processing of data (http://www.ne.se/jsp/search/article.jsp?i_art_id=676090&i_word=it-s%e4kerhet, 2005-03-30).

IT security can be separated in data security and information security. Data security is the protection of data while information security is the protection of the information based on the data. The purpose of both terms is the same, that the right data/information reaches the right person or component at the right time. It is also to be sure that the sender of the information is right and that it has not been changed on the way to the receiver.

The work of IT security contains three important areas:

- *Integrity:* Ensure that the data is accurate for the people using the system
- *Confidentiality:* Ensure that the information is only reached by authorized people
- *Availability:* Ensure that the stored data is available whenever it is needed (M. Alexander 1996, p. 2)

2.2 Security is in every part of development

Security thinking is in every part of in the development of a system. If the creator of the system has not got the security of the system in mind from the beginning when the system was designed, it will be a hard to compensate that later on in the development process (http://phpsec.org/projects/guide/1.html#1.4, 2005-04-02).

Security is also a matter of usability. High security can decrease the usability of a system. To balance the usability of the system with high security is not always an easy task. Session timeouts[1] for example, can be frustrating for the legitimate users but good to prevent unauthorized person to gain access to information (Ibid.).

[1] The user are logged out after a time of inactivity for example

2.3 Threats from users

2.3.1 Attackers

An attacker is a person that breaks into a system or violates it in other ways. The purpose of breaking into a system is to steal or destroy data of the system or in other ways cause problems. (Anonymous 2002, p. 47). Some attackers use tools and code they find on the web in a purpose to cause damage. Some of these attackers have a small understanding of what they actually are doing. Attackers use vulnerabilities in the application' s code to gain unauthorized access to information or destroy the information. (Converse & Joyce 2004, p. 533)

2.3.2 Ordinary users

A common security shortage is the ordinary users of the system. These are users with authorization to use the system. Not all users choose good passwords for user authorization, and are therefore a security risk because they are easy to figure out. Bad passwords are nicknames, your dogs name, love etc. A good password is a mix of small and big characters, a mix of digits, characters and symbols such as ¤#% and at least 12 characters long (M. Alexander 1996, p. 55).

2.4 Methods for discovering passwords

Even if the web application is secured from intrusion due to insecure coding, it does not matter if a malicious user got the password of yours. As said before, the user itself can be a threat against the system by choosing bad passwords. We have found three ways of discovering passwords.

2.4.1 Dictionary

A very common way to discover passwords that are not well formed is to use a dictionary with common words used for passwords. Every word of the wordlist is tried out each at the time until and if the correct one is found. Of course this requires that the password is not a good chosen password (M. Alexander 1996, p. 51)

2.4.2 Brute force

This method is very time consuming for long passwords, so it is not preferable for password more the three to four characters. The principles for how brute force works is that a script tries out every combination possible until it find the right one. This may be a method for retrieving

four digits pin codes but it is not a good idea for twelve character passwords because it would take extremely long time. (M. Alexander 1996, p. 52)

2.4.3 Social engineering

The easiest way to figure out a password is to find out about the family, pets, interest of the person whose password you want to steal and then guessing the password on the basis of what you know about the person. This is only useful as long as the person have not chosen a good password (M. Alexander 1996, p. 13).

2.5 PHP

PHP, "*PHP: Hypertext Preprocessor"* is a scripting language used for creating dynamic websites. The content of a PHP script can be a mix of PHP and HTML code. (http://se.php.net/manual/sv/faq.general.php, 2005-03-30)

2.5.1 History of PHP

In 1995 a man called Rasmus Lerdorf created a series of Perl scripts that he called "Personal Home Page Tools". Later he made a C implementation that was much more advanced and had functions that reminds of how PHP works today. It was called PHP/FI, "Personal Home Page / Forms Interpreter".
In 1997 version 2.0 of PHP/FI was released, in 1998 came PHP 3.0, in 2000 came PHP 4.0 and version 5.0 was released in July 2004.
PHP is one of the most used scripting languages for the web today.
(http://se.php.net/manual/sv/history.php, 2005-03-30)

2.5.2 Apache web server

Apache web server is the most used web server today, it is almost three times bigger than the second most used, Microsoft Internet Information Server. Apache is a free and stable web server, which makes it a good choice (J. Ek & U. Eriksson 2001, p. 43).
PHP can be installed in three different ways on the server: a CGI –program, as an apache module which is compiled into the web server or as an apache module which is loaded every time apache is started, the latest is the one we have chosen. (J. Ek & U. Eriksson 2001, p. 115)
Without PHP or other similar application installed on the web server, it is not possible to make dynamic web pages (J. Ek & U. Eriksson 2001, p. 114).

2.5.3 How it works

PHP is a scripting language used on the server side. The creator of the PHP script uploads it to a server ready for usage.

1. A user uses a web browser to request the PHP script.
2. The server interprets the PHP code, generates and returns HTML code to the users browser.
3. The web browser interprets the HTML code and displays the result on the users screen. (Converse, Joyce & Morgan, 2004, p. 3, p. 27)

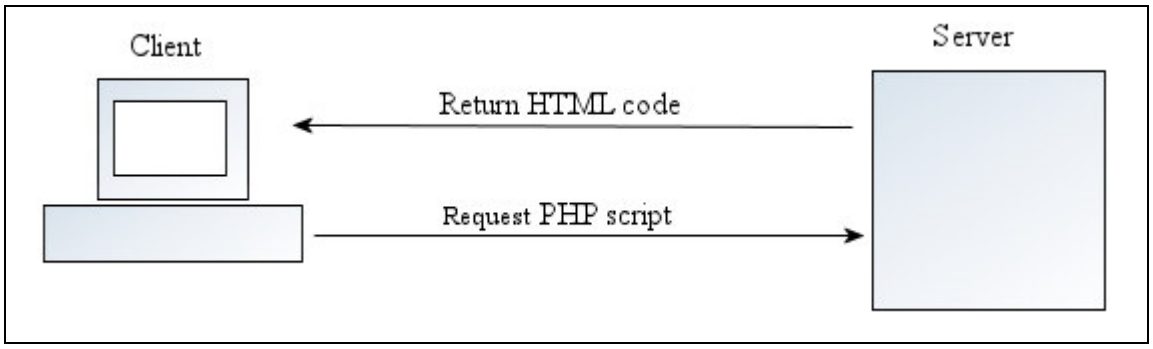

Figure 1

2.6 Important php.ini directives

2.6.1 register_globals

In the beginning, the use of *register_globals* was a way to make PHP programming easier. It did not only make the programming easier, it also made it insecure. (http://www.devshed.com/c/a/PHP/PHP-Security-Mistakes/1/, 2005-04-03)

From version 4.2.0 of PHP, the register_globals directive is turned OFF by default. The intention of register_globals is good, and it is not insecure by it self, it is the use of it in a wrong way that makes it insecure (http://se2.php.net/manual/en/security.globals.php, 2005-04-03).

If register_globals is set to on, variables from forms, cookies, sessions or URL will be used in the same way (T. Converse & J. Park, p. 540). There is no control of what kind of source the variable comes from, if it is an URL, a cookie or a form does not matter, the variable is ready to be used by the receiving script without any manipulation or initializing. (http://www.php.net/manual/en/security.globals.php, 2005-04-04)

2.6.2 magic_quotes_gpc

This is a configuration for PHP that can be set either as on or off. When it is on, a variable that contains the signs single or double quotes, backlashes or NULs will get a backslash in front of the sign. Magic_quotes_gpc runs automatically a function called *addslashes()* that adds backslashes on the mentioned signs. This function can be used without having magic_quotes_gpc on, but then the variables must be called with this function manually in the code.

Examples:

- “ becomes \”, e.g. the quote “Ich bin ein Berliner” becomes \”Ich bin ein Berliner\”
- ‘ becomes \’, e.g. the name O’Brian becomes O\’Brian

The gpc in magic_quotes_gpc stands for get, post, cookie. A slash is only added if the data of the variable is send with the GET, POST or cookie operations, an example is input data from a form send to a PHP script. Variables that do not come through these operations will not have any automatically added slashes (http://se2.php.net/manual/en/ref.info.php#ini.magic-quotes-gpc, http://se2.php.net/manual/en/function.addslashes.php, 2005-04-04).

Why should anyone use this configuration or the function addslashes()? The fact is that PHP communicates with e.g. databases which interpret some characters like quotes, slashes and nulls differently. If a database interprets a value not as text but instead as a command or a part of a command, this could lead to something an attacker could exploit. More about this further on in the chapter about SQL injections. The added backslash shows that e.g. the single quote character is text and not a part of a command.
Let us say that the value where the backslashes were added to is going to be printed on the screen, the function stripslashes() could be used to remove the backslashes (Converse, Joyce & Morgan, 2004, p. 149, p. 150).

2.6.3 magic_quotes_runtime

Similar to magic_quotes_gpc but it will only add backslashes in front of single and double quotes. A big difference is that magic_quotes_runtime will affect values returned from text files and databases. If magic_quotes_sybase also is set as on, no backslash will be added in front of single quotes, another single quote will be added instead (http://se2.php.net/manual/en/ref.info.php#ini.magic-quotes-runtime, 2005-04-04).

2.6.4 magic_quotes_sybase

This is also a similar configuration as magic_quotes_gpc but it will add a single quote instead of a backslash and only in front of single quotes. It will not add anything else to other characters. It can be set as on or off. If magic_quotes_sybase is set as on, it will override magic_quotes_gpc which means that if magic_quotes_gpc also is set as on it will function as it was off (http://se2.php.net/manual/en/ref.sybase.php#ini.magic-quotes-sybase, 2005-04-04).

2.7 Superglobals

PHP contain some predefined superglobals. These are global variables that can be used thru the whole script no matter if it is in a function or a class. The purpose of them is a little bit different.

$_SERVER: The server sets the content of this global variable, it may be information about paths, headers or script location. For example, $_SERVER['PHP_SELF'], will contain the path to the place where the current script is located.
(http://www.php.net/manual/en/reserved.variables.php, 2005-04-08)

$_GET: A variable to receive information from a form with GET specified as method in the form. The GET variable is also used to retrieve information sent in the URL. For example www.somesite.com/script.php?id=4, the information after the "?" in the URL can be retrieved by using *$identity= $_GET['id'];* this will set the variable $identity to 4
(http://www.php.net/manual/en/language.variables.external.php, 2005-04-05)

$_POST: Similar to $_GET but its only used to retrieve data from forms using the post method. The post method of sending data from a form is more secure because it does not show the information sent in the URL. The POST method also allows a larger amount of data to be passed (T Converse & J Park, p.124).

$_COOKIE: Cookies are information that the server side can use to retrieve and store information from the client side of a connection.
(http://wp.netscape.com/newsref/std/cookie_spec.html, 2005-04-08) The contents in the COOKIE variable are data received from the cookie stored on the client side.
(http://www.php.net/manual/en/language.variables.external.php, 2005-04-08)

$_REQUEST: This can be used to receive data from all the three GET, POST and COOKIE types. (http://www.php.net/manual/en/reserved.variables.php, 2005-04-08)

2.8 CAPTCHA

To be able to prevent bots from flooding a guestbook or create 10000 new e-mail accounts on a page for example, a method called CAPTCHA can be used. CAPTCHA stands for "*C*ompletely *A*utomated *P*ublic *T*uring Test To Tell *C*omputers and *H*umans *A*part".

The idea of CAPTCHA is an image with distorted text, which a human can interpret but bots cannot. These pictures can be seen for example when registering a hotmail account and on many other places on the web. The method could also be used to prevent brute force attacks to discover passwords. A page can let the user enter both their password and the CAPTCHA phrase. The user can only be allowed in the system if both the password and the pass phrase from the CAPTCHA image are correct. (http://www-2.cs.cmu.edu/~biglou/captcha_cacm.pdf, 2005-04-14)

2.9 Database

A database, according to the Swedish encyclopedia *Nationalencylopedin*, is organized data collected in registers (http://www.ne.se/jsp/search/article.jsp?i_art_id=150823, 2005-04-01). Usually a system is used for communications with the database, like selecting, adding, deleting and changing the data. These systems are called DBMS which is short for Database Management System. (Connolly & Begg, 2002, p. 4, p. 16)

2.9.1 MySQL

MySQL is one of the most commonly used databases for web systems today. The first version was released in 1995. MySQL is a RDBMS, Relational Database Management System, and its query language is based on SQL. MySQL is an open source product and can be free depending on licence. (Converse, Joyce & Morgan, 2004, p. 4)

2.10 Security threats

The security threats that we have found and considered to be valuable to know about when we researched this topic on websites discussing security threats for PHP are:

- Cross site scripting or short XSS
- Cross-Site Request Forgeries or short CSRF
- Spoofed forms submissions
- SQL injections
- Bots
- HTTP request
- Session threats
- File uploads

2.11 Cross site scripting

Cross site scripting, or short XSS, is a threat against web applications. An XSS attack takes advantage of a PHP script's trust towards external data. The core problem is that if the script trusts external data as it is, attackers could take advantage of this and send bad data (http://shiflett.org/articles/foiling-cross-site-attacks, 2005-04-06).

A web browser interpret some data as text and some data as commands, e.g. if you write <html> in a print command for PHP, the web browser will not print <html> on the screen, it will interpret it as html code.

```
< ?php
print "< html> ";
?>
```

Example 1

2.11.1 Where could external data come from?

External data could come from many sources depending on the functionality of the PHP script. External data could come from:

- A form. Many web pages have forms with input fields where the users could fill in data and send it to a PHP script, e.g. contact forms, guestbooks, forums, communities etc.
- An external database. If the script is connected to an external database, you will probably not have any control over how the data looks like when it was added in that database.
- XML/RSS file. Today a lot of data swapping is being made with XML/RSS. Knowing if the XML/RSS is valid just before you receive the data is not easy.

2.11.2 What is the threat in XSS?

The threat becomes real when the data is stored in the webpage's database or coming from an external source, collected and printed out on the webpage visible for other users. As mention earlier, a browser could interpret some data as a command instead of text. The commands could not just be html code, it could also be JavaScript, ActiveX and more (http://www.phpadvisory.com/articles/view.phtml?ID=4, 2005-04-07).
Some threat scenarios can look like:

- An attacker fills a guestbook with banner advertisement.
- Users are sent to another webpage because of added JavaScript code.
- Steal other user's cookies, and then trying to use that user's account.

These example scenarios are not the limit for XSS, it is up to the attacker's imagination.

2.12 Cross-Site Request Forgeries

Cross-Site Request Forgeries, or short CSRF, is another threat against web applications. It takes advantage of applications that trust their users. The attacker uses this trust for his/her purposes.

An attack can be against e.g:

- A user in a forum. The attacker can post the text he wants with the help of the rightful users account but without the user's knowledge. The new post will be signed by the valid user.
- A customer in a web shop. The attacker orders merchandise for the customer, by the customer without the customer's knowledge of what happened.

The attacker uses info that is sent by a HTTP request, which is a protocol sent from the web browser of the client to the server
Usually the image tag in HTML is used for CSRF attacks, where the attacker puts an URL that runs a command instead of putting a URL to an image. This can be exploited because the server does not know that it was an image that was requested, the request handles all the URLs in the same way (http://shiflett.org/articles/foiling-cross-site-attacks, 2005-04-08).

2.13 Spoofed forms submissions

Input from users to a PHP system is made in web page forms. To spoof a form submission means to send data from another form than the form expected. A user can catch the source code of a page and then modify it. The reasons for doing this can be a way to circumvent client side restrictions, such as predefined options or JavaScript controls. (http://phpsec.org/projects/guide/2.html#2.1, 2005-04-08)

The method can be used to overwrite variables if register_globals is set to on. Someone can then send variables from a self-made form that will overwrite a variable in the receiving script. If register_globals is off the user will need to catch the variable in a $_POST, $_GET or $REQUEST superglobal. In the chapter 'variable overwriting" it is shown how variable overwriting accidentally can occur. Using spoofed form submission can be a way of modifying variables intentionally.
(http://academ.hvcc.edu/~kantopet/php/index.php?page=php+form+data&parent=php+client+side, 2005-04-08).

2.14 SQL injection

SQL-injection is a way to exploiting web application by changing the SQL-statement used to retrieve, add, delete or change data in a database in a way that is not supposed. These attacks are possible due to bad filtering of data entered by users. Many e-commerce sites, both large and small, suffer from this vulnerability. SQL injection is not a programming language specific problem. (http://www.linuxexposed.com/Articles/Hacking/Introduction-to-SQL-Injection.html, 2005-04-07)

How dangerous SQL-injection can be depends much on the code and what database used, if no data filtering is performed and the database used allows multiple queries, it can be very

dangerous. MySQL are not allowing multiple queries until recent versions of the database, which limiting the possible damage. (http://phpsec.org/projects/guide/3.html#3.2, 2005-04-08)

2.15 Bots

Websites with guestbooks, polls, blogs or any other kind of sites with a form are possible targets where bots can submit forms. The most common use of these bots is to spam guestbooks for example, to increase the search result ranking in search engines. (http://phpsec.org/articles/2005/text-captcha.html, 2005-04-14)

Bots can also be constructed to create accounts on sites where it is possible to register for an account, for example free e-mail services, where bots can sign up for thousands of e-mail addresses in a short amount of time and then use this accounts for sending e-mail spam. Another real life example is a web poll where the webpage slasdot.com asked which the best school of computer science was. Students at CMU (Carnegie Mellon) created a bot voting for their school and MIT (Massachusetts Institute of Technology) created their bot voting for their school. The poll turned up to be a matter of who created the best bot, instead of a measurement of which school that was best.(http://www-2.cs.cmu.edu/~biglou/captcha_cacm.pdf, 2005-04-27)

2.16 HTTP request

An HTTP request is a request sent to the web server to start an action. This request consists of a URL, form data if it is a form that starts the request, what web browser used, what type of request etc. The web browser commonly creates the request and is nothing the programmer has to care about (V. Jonsson 2001, p. 93).

A HTTP request starts with the method used, for example POST. After the method, the URL to the file on which the request is applied, what version of http protocol used is also included in the request. Followed by the version of the http protocol is the request header, such as content-type and content-length (http://www.jmarshall.com/easy/http/#whatis, 2005-04-16).

An http request using the POST method can look like this:

```
POST /script.php HTTP/1.1
Host: somehost.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 9

name=John+howard
```

Example 2

The POST on the first row is the method used, */script.php* is the path to the script which are going to process the data and the host is the host where to send the request. *Content-Type: application/x-www-form-urlencoded* is a description of what kind of data sent, in this case form data and also the length of the content is included in the statement Content-Length. At last the data is sent.

2.17 Session threats

2.17.1 What is a session?

According to the PHP company Zend Technologies "session management is a mechanism to maintain state about a series of requests from the same user across some period of time" (http://www.zend.com/zend/tut/session.php, 2005-04-14).
Sessions are used to handle users that visit a webpage. The server can not tell the difference between what one user does on the webpage or another. A session identifies a user so that different information can be displayed, e.g. after a user has logged in on a member service, the user can perform operations that will effect him, like ordering a book from a web shop. Session management was not "build in" in PHP 3, but was included in PHP 4 (Ibid).

Examples of applications that a session can be used for:

- Login functions, checks if logged in.
- Member services, e.g. ordering products.
- Users adding products into a shopping cart.
- Changing language on a webpage.

Every session has an ID that is unique. The ID identifies the session and can be stored on the client's computer using a cookie, but also in the URL (http://se2.php.net/manual/en/ref.session.php, 2005-04-15). An URL with an included session ID could look like this, where the *PHPSESSID* variable contains the session ID:

```
http://www.example.eg/index.php?PHPSESSID=123456789
```

Example 3

2.17.2 Session guessing

If the session identifier is used in the URL using a GET variable, like in example 3, someone could guess the session ID in the URL for active sessions (http://shiflett.org/articles/security-corner-feb2005, 2005-04-16).
If it is stored in a cookie, the attacker could change the session ID in his own cookie file to another ID by guessing. (http://shiflett.org/articles/the-truth-about-sessions, 2005-04-16)

2.17.3 Session hijacking

Session capturing is when the attacker takes over someone else's session. Let us say that the attacker has a web log where referring URLs are showed. If the referring URL contains a sessions ID like in example 3, the attacker could try to take over that session if it is still active. The attacker could also try to sniff network traffic to get hold of a session ID. (http://se.php.net/session, 2005-04-16)

2.17.4 Session fixation

Session fixation is when the attacker decides the session ID for a user. The user is lured to set this session ID. If the ID becomes valid and is set, the attacker uses the session ID to get hold of session specific information (http://shiflett.org/articles/security-corner-feb2004, 2005-04-16).
Sessions IDs can be set with the help of:

- URL, the attacker wants the user to login with an URL that contains a sessions ID.
- Forms, e.g. external login from another website with a hidden input field containing a sessions ID.
- Cookies, the attacker can try to set a cookie containing the session ID (http://www.acros.si/papers/session_fixation.pdf, 2005-04-15).

2.18 File uploads

A web application made in PHP can use file uploads if the PHP configuration *file_uploads* is set as ON. File uploads can be used for letting users uploading files to the server.
This feature is used in some applications e.g. web forums for image uploading etc. (Converse, Joyce & Morgan, 2004, p. 542)

The risk that follows file uploads can be:

- Someone uploads a file that contains a virus of some kind. If there is a link to the file on the webpage or the file is sent by email from the uploaded directory, suddenly the web application became a virus spreader.
- Someone uploads files that are not allowed by you or by law, e.g. child pornography.
- Someone uploads large files of the intent to slow down the server, crash it or maybe want to occupy bandwidth (Ibid.).

Before making a file upload function in a web application, the developer must think about if it is really necessary in the application, because file uploading is one of the most insecure functions in PHP (Converse, Joyce & Morgan, 2004, p. 545).

3 Result

3.1 Control origin of a form

To ensure that the data comes from the right form it is important to prevent different vulnerability to be exploited. This is useful to prevent spoofed form submissions, bots and CSRF.

3.1.1 Origin control using $_SERVER[HTTP_REFERER]

This is a simple way to determine whether the page that the user came from to the current script is correct corresponding to the right side of the comparison in example 4.

```
function check_source()
{
        if($_SERVER[HTTP_REFERER]= = 'http://www.ourhost.com/form.html')
                return true;
        else
                return false;
}
```

Example 4 (T. Converse & J. Park p. 541)

This method has some weaknesses, and should therefore not be used as the only validity test before the data sent in the form is processed. The problem is that the $_SERVER[HTTP_REFERER] variable are not set by all clients and are because of that not reliable, it should only be used as an extra complement to other filtering methods. The $_SERVER[HTTP_REFERER] is a superglobal variable that contains the address to the place from where the user came (if any) to the current page. (http://se.php.net/manual/en/reserved.variables.php#reserved.variables.server, 2005-04-18)

3.1.2 Origin control using sessions

Another better method to control that the form sending the data actually is the right one is to use sessions. Before the actual form, we have some rows of PHP code that creates a random md5 encrypted number. This number is called *idnumber* in this script and is put into the session variable *$_SESSION['idnumber']*, so we can use it at the receiving page as well. This unique number for each form is also sent in a hidden form field.

```
<?php
session_start();
$idnumber = md5(uniqid(rand(),true));
$_SESSION['idnumber'] = $idnumber;
?>

<form method="post" action="result.php">
Type in the number of items: <input type="text" name = "amount">
<input type="hidden" name="idn" value="<?php echo $idnumber; ?>" />
<input type="submit">
</form>
```

Example 5 (http://phpsec.org/projects/guide/2.html#2.4, 2005-04-19)

In example 6, the script receiving the form data is shown with some control procedures. First of all, the script controls so that some data has been filled in, if not an error message appears. Next control check so that the id number, called *idn*, is equal to the one sent in the session. If both the one sent in the hidden field and the session id number is right, the data will be processed.

```
<?php
session_start();

 if (isset($_POST['amount']))
{
          if($_POST['idn'] == $_SESSION['idnumber'])
          {
                    $message = htmlentities($_POST['amou nt']);
                    print "The value entered was: $message";
          }
          else
          {
                    print "this is not right source!!";
          }
}
else
{
          print "A value has to be filled in!";
}

?>
```

Example 6 (http://phpsec.org/projects/guide/2.html#2.4, 2005-04-19)

This is a very reliable method to control the source of the data. If a user tries to cut away the hidden tag, it will only lead to that the data is not processed. Also if the id-number in the hidden tag is modified in some way, it will not correspond to the value of the session variable and the form submission will be considered invalid. This example is based of a similar example found at (http://phpsec.org/projects/guide/2.html#2.4, 2005-04-19).

3.2 register_globals enabled

The example 7 shows one risk with register_globals is set as ON. First the function *user_ok()* is called to determine whether the user has the access to the system or not. If the function returns true, the user is okay and can be logged into the system.

The security problem with this is that it is possible to change the value of $authorized only by writing *login.php?authorized=1* in the address bar of the web browser and by that gain access to the system. The variable *$authorized*, in this case, does not care whether the variable comes from the URL or is set by the *if(user_ok())* control function (T. Converse & J. Park, p. 821).

```
// file: login.php

if(user_ok())
{
        $authorized = 1;
}
if($authorized == 1)
{
        print "you are allowed to see the  secret information";
}
else
{
        print "You are NOT allowed to se the secret information";
}
```

Example 7

3.2.1 Variable overwriting

Having the register_globals set to on is also a security risk because variables tend to overwrite each other, if they are not correctly named. The example in example 8 and 9 shows the problem with variable overwriting. In the code 9, we are sending the information to the page called product.php?product=jeans (Code 8). This specifies that jeans are the product the customer wants to buy.

Also the name of the product is named "product", because this is the list of different products available. When the customer then submits the form, the information is printed in the file called product.php (example 8). This supposes to explain that the customer wants to buy jeans, and what brand. When register_globals is set as on, PHP make no separation between variables that comes from a form or from somewhere else, in this case a variable sent in GET-format. The output of this script will be for example, "You chose to buy Crocker with the brand Crocker", if the user chose Crocker as the brand to buy, instead of "You chose to buy jeans with the brand Crocker". (T Converse & J Park, p. 540)

This is just an example and the problem would be easy to avoid, it is easy to directly see that both variables are called product and then just change one of them, but in large project where many programmers are involved it can be hard to know from where a variable comes and can cause this kind of problem (T Converse & J Park, p. 821).

What variable that is overwritten depends on the priority set in the php.ini file. By default it is set to EGPCS (Environment, GET, POST, Cookie, Server). This explains why Crocker is printed but not Jeans. The later in the order of letters the higher priority it has, and because POST is later then GET, variable coming from the entered form will overwrite the GET variable with the type of product entered.
(http://se.php.net/manual/en/ini.core.php#ini.variables-order, 2005-04-04)

```
//product.php recives the information from form.html

<?php
print " You chose to buy $product with the brand $product";
?>
```

Example 8

```
//form.html

<form method="POST" action="product.php?product=jeans">
<SELECT name="product">
<OPTION value = " Crocker " > Crocker</OPTION>
<OPTION value = " Levi´s "> Levi´s</OPTION>
<OPTION value = " Lee"> Lee</OPTION>
</SELECT>
<input type="submit" name="ok" value="OK" >
</form>
```

Example 9 (J. Park & C. Morgan p540)

3.2.2 Protection from register_globals vulnerabilities

It was not a random decision to turn register_globals off in later versions of PHP, it is made to increase the security of PHP scripts. One way of protecting scripts from vulnerabilities related to register_globals is simply to turn it off in the php.ini file. This would force the programmer to change the script because it could not automatically use the variable sent from a form directly, it have to be initialized before it is possible to modify, we would have to receive the variable in one of the PHP superglobals, such as *$_POST['']*. In a script using register_globals set as on, the initiation is not necessary, but if not made, it can lead to dangerous consequences (http://www.php.net/manual/en/security.globals.php, 2005-04-18).

If register_globals is still set as on, there are some important considerations to make. The most important thing: always initialize the variables. If we change the login example to look like this instead of what presented in example 7, the possibility to change the variable authorized to 1 in the address bar no longer exist:

```
// file: login.php
$authorized = false;

if(user_ok($user,$pass))
{
        $authorized = 1;
}

if($authorized == 1)
{
        print "you are allowed to see the secret information";
}
else
{
        print "You are NOT allowed to se the secret information";
}
```

Example 10

The variable authorized is now by default set to false, and will not change unless the right username and password defined in the function *user_ok()* is correct, this is because variables defined in the script are never automatically overwritten by the variables defined by the GET or POST method. (http://www.php.net/manual/en/security.globals.php, 2005-04-18)

As it looks now, it is still possible to send the username and password via the URL by writing, *login.php?user=usersusername&pass=userspassword*, in the address bar of the web browser. In this case, it is not a problem because we still have to know the right username and password to gain access. But to narrow the possibilities of how variables is received by the script it is a good idea to use the POST[] superglobal. The call of the function for validating the user input could then look like this:

```
if(user_ok($_POST['user'],$_POST['pass']))
{
        $authorized = 1;
}
```

Example 11

This means the username and the password typed by the user, have to be sent by the POST method to be processed, it does not stop the user from try out the password multiple times, but it narrow the possible ways of doing it. (J. Park & C. Morgan p541)

3.2.3 Protecting the script from variable overwriting

When having register_globals on, the programmers have to be more careful of how variables are named then if it is turned to off, to ensure variable overwriting not to occur. Because no difference are made for variable that comes from GET, COOKIE or POST method. The best thing to prevent this to happen is to initialize every variable and receive them as variable from the environment they come from.

```
//product.php recives the information from form.html

<?php

print "You chose to buy ".$_GET['product']." with the brand ".$_POST['product'];

?>
```

Example 12

If we do like example 12, defining what variable that comes from the GET method and what variables that come from the POST method, it does not really matter if both the form variables in example 9 is called product, because they are handled different. If register_globals was set to off, this is the way we would have been forced to do it, because the variable would not be useable in case we do not receive it into a superglobal first. (J. Park & C. Morgan p540)

The way we solve it without using superglobals, is to define the names more carefully in the form, for example it can look like this:

```
<form method="POST" action="product.php?product=jeans">
<SELECT name="brand">
<OPTION value = " Crocker " > Crocker</OPTION>
<OPTION value = " Levi´ s "> Levi´ s</OPTION>
<OPTION value = " Lee"> Lee</O PTION>
</SELECT>
<input type="submit" name="ok" value="OK">
</form>
```

Example 13

Now we have separated the product type from the brand of the certain product. When sending this to the receiving form the product variable will be set to jeans and another variable will be called brand.

3.3 Cross site scripting (XSS)

3.3.1 What could a XSS attack look like?

Let us say that a highly respected organization/company like a firm of undertakers has a website where the users could add the names of their love ones that has past away in a special section of the site. The names are then listed on the website in a very respectful way.
Let us say that this function of adding names does not have any protection against XSS attacks, the names are just added directly to the database and then gathered from the database.

```
// a part of a php script, showing the name printing...

$name= $row;  // variable name gets the value from the database
print $name;  //  the name is printed
```

Example 14

An attacker takes advantage of this and instead of writing a name, the attacker writes a JavaScript code like this:

```
<script language="javascript">window.location="http://www.xxx.sex";</script>
```

Example 15 (Converse, Joyce & Morgan, 2004, p. 532)

The web browser will interpret the ‘name” as a JavaScript and not as plain text when listing the names. The JavaScript that was submitted is a redirect script that opens another website. Every user that has JavaScript enabled in their browser, which is standard, will be redirected to, let us say, a hardcore porn website (Converse, Joyce & Morgan, 2004, p. 542-543).
This is probably something that the users of the website would not want to see in this time, and this was probably not good for the reputation of the firm of undertakers. Hopefully this fictional XSS attack was not made during the webmaster’s summer vacation.

The scenario above was an example how very few lines of code, that does not require advanced programming skills, that does not do anything else than just redirect, could do a lot of damaging on a company’s reputation but also to the users feelings.

3.3.2 Protecting a script from XSS attacks

In previous chapters we have learned what XSS is and what it can do. But how can we protect our scripts against a XSS attack?

The PHP Security Consortium recommends us to consider all data as evil until the data has been proven harmless (http://phpsec.org/projects/guide/2.html#2.3, 2005-04-18).

All data that is sent from a user or an external source and also when displayed for the users should be filtered. There are many ways to filter data. The PHP Security Consortium recommends us to use the built-in filtering functions in PHP, because they are more tested and faster than a self-made function that does the same thing (Ibid.).
Let us look at an example:

```
// $invalid_data contains the value:
// <script language="javascript">window.location="http://www.xxx.sex";</script>

<?php

$valid_data=htmlentities($invalid_data,ENT_QUOTES);
print $valid_data;  // the value is printed

?>
```

Example 16

In the example above the built-in PHP function htmlentities is used. The function will convert code that could be interpreted as commands by the web browser to html characters. The script will print out the command in text format instead of executing the code. *<script language="javascript">window.location="http://www.xxx.sex";</script>* will look like this on the screen, but when looking at the source code for the webpage, it actually looks like the example below.

```
&lt;script
language=&quot;javascript&quot;&gt;window.location=&quot;http://www.xxx.sex&q
uot;;&lt;/script&gt;
```

Example 17

When using htmlentities(), the data that are going to be filtered and converted, should be put in the middle of the parentheses. A parameter can be added in the parentheses separated with a comma after the variable containing the data. The parameter can be:

- ENT_COMPAT, is default if no parameter is added. Converts all double quotes but not single quotes.
- ENT_QUOTES, converts both single and double quotes.
- ENT_NOQUOTES, does not convert any type of quotes (http://se2.php.net/manual/en/function.htmlentities.php, 2005-04-18).

Another function for converting tags is strip_tags(), which will convert HTML and PHP tags. The function strip_tags() supports a parameter for allowed tags. Let us say that we want to allow the HTML tags
 and <hr/>.

```
// $invalid_data contains the value:
// hello, this is a picture <br/><hr/><img src="example.eg" /><br/><hr/>

<?php

$allowed_tags="<br/><br><br /><hr><hr/><hr />";   //list of allowed tags

$valid_data=strip_tags($invalid_data,$allowed_tags);      //makes valid data
print $valid_data;  // the value is printed

?>
```

Example 18

The example above shows which tags that are allowed and will not be stripped. The $invalid_data variable contains the data: *hello, this is a picture
<hr/>
<hr/>*
It will strip the <img> tag because it is not in the allowed list. The strip_tags() function is different from htmlentities. One difference is that it does like the name of the function, it strips the tags, it does not convert them to other signs. The <img> tag will therefore be deleted from the variable $valid_data. (http://se2.php.net/manual/en/function.strip-tags.php, 2005-04-18)

It is also recommended to convert parentheses to html characters like (to (and) to) because e.g. JavaScript uses parentheses in its code (http://www.phpadvisory.com/articles/view.phtml?ID=4, 2005-04-18).
The functions htmlentities(), or strip_tags() will not take care of parentheses. There are other functions for replacing characters, one of them is str_replace().

```
<?php

$replace=array("(",")");
$valid_char=array("&#40;","&#41;");
$valid_data=str_replace($replace,$valid_char,$invalid_data);
print $valid_data;  // the value is printed

?>
```

Example 19

The variable $replace contains an array with characters that are going to be replaced. The variable $valid_char contains an array with characters that the characters in $replace are going to be replaced with. The first parameter in str_replace is the characters that are going to be replaced. The second parameter contains the replacing characters. The third parameter contains the data where characters are going to be replaced. This function does not only work with replacing parentheses or one character in a time, you can replace words or sentences if

that is your goal (http://se.php.net/manual/en/function.str-replace.php, 2005-04-18). Replace characters that are a threat to your script.

3.4 Cross-Site Request Forgeries

The example below shows an URL that does not lead to an image.

```
<img src="http://www.examplewebsite.org/formaction.php?name=mrattacker
&email=haha@hoho.ha&message=hahahaha">
```

Example 20

In this case let us say that it leads to a form submission command. Let us also say that the attacker uses an XSS attack on a website with a lot of traffic and adds this HTML-code as the one above, the receiver of this form will get a lot of "hahahaha" messages.

Another example that is similar to the one above can be a CSRF attack against a forum that is not protected against it. Let us say that a high member of a political party is a member on their web forum. If the forms of the forum use the get method it will be possible to send the same variables through the URL. The attacker just has to look in the source code for the form in the html code to figure out what variables that are used. The attacker lures the politician in to another webpage with an image tag that requests the submission of messages in the forum. If the politician is logged in, or is automatically logged in by a cookie, the request from the image will be handled in the same way as if the request was done in the forum. The result can be that the attacker succeeds to publish, let us say an embarrassing message on the forum "signed" by the politician.

But CSRF does not only have to do with posting messages. If a web shop is not protected and its commands for ordering are handled by the get method, as the forum example, the attacker could take advantage of a membership of a customer and order merchandise to the customer by the customer. If the attacker is also successful with an XSS attack against a popular website and posts the image tag there and the web shop is also a very popular shop, this could maybe lead to a lot of new orders for the web shop but rather fake ones. (http://shiflett.org/articles/foiling-cross-site-attacks, 2005-04-08)

3.4.1 Protecting a script from CSRF attacks

The developer of the PHP script should think of using the POST method when important operations are done, because the POST method is more secure. If register_globals is on, the posted values should be receive by a $_POST. $_REQUEST is not secure.
It is also important to go thru more steps than just one for an important operation, e.g. having a confirmation form. Maybe it is more convenient for the user to do as little as possible, but convenience should not get in the way of security, maybe it is not so convenient when the user is exposed for a CSRF attack (http://phpsec.org/projects/guide/2.html#2.4, 2005-04-18).

Let us take a look at an example of a form.

```
<!-- form.html  -->

<form method="post" action="formaction.php">
Name: <input type="text" name="name" />
Email: <input type="text" name="email    " />
Message: <textarea name="message"></textarea>
</form>
```

Example 21

The form above shows how the information is sent. The method used is POST.

```
// formaction.php
// script receives data from a form using the POST method

<?php

$sender= $_POST['name'];
$reply_to= $_POST['email'];
$posted_message= $_POST['message'];

?>
```

Example 22

The PHP script above shows how the script receives the data through $_POST. If someone tries to use an image tag containing a URL which uses the GET method, nothing special would happen, because the script only receives data send by the POST method, sent from a real form, not from an URL.

Another protection from CSRF attacks is to use a random token to identify that the user uses the form where data is supposed to be sent from (Ibid.). Read more about this in chapter 3.1.

3.5 Spoofed form submission

3.5.1 Spoofed form submission example

Example 23 is a form where the user choose his/her year of birth. It can be for example a part of a registration procedure for a webpage. This is just an example list with some years choosen to exemplify a spoofed form submission. When the user clicks okey, the form will be submited and the data containing year of birth will be send to age.php, specified in the <form> tag. The user is limited to the choise of the list.

```
<form method="POST" action="age.php">
<p><font size="5">Choose your year of birth:</font></p>
<SELECT name="agelist">
        <OPTION value = "1983" >1983</OPTION>
        <OPTION value = "1984">1984</OPTION>
        <OPTION value = "1985">1985</OPTION>
        <OPTION value = "1986">1986</OPTION>
</SELECT>
<input type="submit" name="ok" value="OK">
```

Example 23 (http://phpsec.org/projects/guide/2.html#2.1, 2005-04-18)

In fact, the user is not really limited to the choice of this list. A user who has bad intention with his usage of the site can easily circumvent this form just by copy the source code on the site and modify it. The tricky user could in a few minutes create a form that looks like the one in example 24. The user can change the action-statement to the entire address of the receiving script, and change from a list to a text field. In the text field, the user can write whatever values he wants to, and it will be used by the PHP-script as the value sent by the ordinary form if no data filtering or source validation is made.

```
<form method="POST" action="http://thepage.com/age.php">
<p><font size="5">Choose your year of birth:</font></p>
<INPUT type = "text" name="agelist">
<input type="submit" name="ok" value="OK">
</form>
</body>
```

Example 24(http://phpsec.org/projects/guide/2.html#2.1, 2005-04-18)

3.5.2 Defense against spoofed form submission by data filtering

First of all, filter the data sent by the form on the server-side. Filtering it with just JavaScript or other client-side method is not enough because it is easy to just turn JavaScript off or cut the code away, and submit the form without validation of the data. (http://phpsec.org/projects/guide/2.html#2.1, 2005-04-18)

An example of how we can determine whether a value is valid or not for further processing is shown in example 25.

```
function age_okey($age_var)
{
        $valid_age = array("1983","1984","1985","1986");
        $okey=in_array($age_var,$valid_age);
        if($okey==false)
        return false;
        else
        return true;

}
```

Example 25

As a parameter to the function we have the value submitted by the user. The array called $valid_age contains all the valid values possible. The function in_array() checks if the value submitted by the user is in the array or not, if it exist in the array of valid values, the function will return true, otherwise false which will cause that the data is not processed.

If this kind of server-side control procedures is made, it does not matter if a user with malicious intends tries to send other values then expected to the script.

3.5.3 Defense against spoofed form submission by origin control

Another method, which can increase the security more, is to determine the source of the data, i.e. determine if the data is sent by the right form, not a spoofed one. This would consider data sent from wrong form invalid and would not be processed, how to perform this can be read in chapter 3.1.

3.6 SQL injection

3.6.1 Example discription

Our example of the principle for how a SQL injection works is a login procedure in a system. The information in the database looks like in the table login, figure 2. There is a user-id for each user of the system, a username and a password. Example 26 is the login form in which the user enters their username and password. Example 27 is the script that receiving the login information, looking up the user in the database and determine if the user is valid or not.

Table: login		
Userid	**Username**	**Password**
1	Johnny	C9sfF4As
2	Andreas	rockstar
3	Tony	wanna_be_famous
4	John	Fido

Figure 2

The user enter their username and password in the form, the data will be sent by POST method to the page login.php (example 27). The username is sent in the variable "user" and the password is sent in the variable called "pass".

```
<form method="POST" action="login.php">
Username:<input type="text" size="15" name="user"><br>
Password:<input type="password" size="15" name="pass"><br>
<input type="submit" name="ok" value="OK">
</form>
```

Example 26

Let us take a look at what happen when the user have entered their login information in the form and submited it. First the the script will perform a connection to the database server and choose which database to connect. In this case, the database is called login.

The two variables called $user and $pass contains the information the user entered in the form.

A SQL statement is produced in the variable *$question*. The question says:
Retrieve the username in the table called login of the user with the username and password entered in the form. The question will then be executed with the command *mysql_query($question, $db)* and the result will be saved in the variable *$answer*.

The line, *$amount = mysql_num_rows($answer);* counts the result received from the database query. A result of more then zero means there is somebody with that username and password in the database and he/she are then allowed to login to the system and a text "Welcome <user>" will be displayed. If the user wrote the wrong username and password, he/she will be refused by the system and the text "you don't have access" will appear.

```
//login.php
<?PHP


  $db = mysql_connect("localhost","Username","Password") or die("can´t connect!");
  $selectedDB = mysql_select_db("login",$db);      //connecting to and choosing database

  $user = $_POST['user'];
  $pass= $_POST['pass'];


  $question = "select username from login where username = '$user' and password =
'$pass'";
  $answer = mysql_query($question,$db);

  $amount = mysql_num_rows($answer);   //counting the number of results
  if($amount > 0)  //if the number of results are more      then 0
  {
            $row = mysql_fetch_row($answer);   //print the users name
            print "welcome $row[0]<br/>";
  }
   else //otherwise the user will not be logged into the system
   {
          print "You don't have access";
  }


?>
```

Example 27

3.6.2 Performing the SQL injection

If the right data were entered in the login form, the SQL query would look like this, for example:

```
"select username from login where username = 'Johnny' and password = 'C9sfF4As'"
```

Example 28

This would give the user Johnny access to the system because it is the right username and password. A malicious could enter this instead:

Username: Johnny
Password: x' OR username =' Johnny

He would also gain access to the system without knowing the password for Johnny. The SQL-code would then look like this:

```
select username from login where username = 'Johnny' and password = 'x' OR
username = 'Johnny'
```

Example 29

This means, select the username from the table called login where username is Johnny AND the password is x OR the username is Johnny. The first part of the question will not return any result because Johnny's password is not x. What we have done here is breaking the original SQL statement with the sign ' (singe quote) and added another condition to the question. By adding x' OR username = 'Johnny the database is asked to look for the username "Johnny" in the username-row in the database. The username Johnny will be found which means that the variable *$amount* will be more then zero. The malicious user will be logged into the system as Johnny.

The trickiest part is to figure out the row names that contain the username and password. But it is certainly easier than figure out the password for the user. It may take a while of guessing before right row-names is found, but when (and if) a positive result is found, attackers can gain access to users accounts either by knowing the usernames or guessing them, no password is needed.

3.6.3 Defense against SQL – injection

One important action to take for preventing SQL injections to be possible to exploit, is data filtering. Filtering the data so just the right data for right purpose is allowed (http://phpsec.org/projects/guide/3.html#3.2, 2005-04-19). In our example with the login we could use for example the function shown in example 30 to first check the length of the username and password to prevent long SQL statements. It is a simple function that uses the strlen() (http://se2.php.net/manual/en/function.strlen.php, 2005-04-19) function to determine if the condition is true or false.

```
function validate_logindata($username, $password)
{
        if(strlen($username)> 12 || strlen($password)> 12)
        {
                return false;
        }
        else
        {
                return true;
        }
}
```

Example 30

This function just limiting the length of the SQL-statement, it does not prevent it from being a possible target. So a function like the one in example 31, is also necessary to determine so the password or whatever string we want to check fulfils our condition.

```
function valid_password($password)
{
        $pattern = "[^ a-zåäö0-9A-ZÅÄÖ#¤% &_]";
        $valid = ereg($pattern,$password);
        if($valid)
        return false;
        else
        return true;
}
```

Example 31

This is an example function using regular expression for how to validate the password field. Only the characters from a-ö, A-Ö, 0-9 and #¤%&_ are allowed for further processing of the data (V. Jonsson 2001, p165). So, if a malicious user tries out an SQL - injection it will not work because the sign ' (single quote) are not allowed.

3.6.4 Escaping the data

Sometimes it is not possible to filter away characters like (single quote) because it may needed in, for example, a name like O'Neill So, another action to take for making SQL - injection impossible is to escaping the data using the function *mysql_escape_string(), mysql_real_escape_string()* or *addslashes()* (http://phpsec.org/projects/guide/3.html#3.2, 2005-04-19).

Which one of the functions mysql_escape_string() or mysql_real_escape_string() that is used really does not matter. The difference between them is that mysql_real_escape_string() takes a link identifier, i.e. a connection to the database as an argument (http://se2.php.net/manual/en/function.mysql-escape-string.php, 2005-04-19).

A standard way of doing the escape is to using the function addslashes(). The two earlier functions mentioned are MySQL specific functions. The functions adds an \ (backslash) to signs like ' (single quote), "(double quote),\ (backslash) and NULL. The purpose of doing this is to prevent possible SQL-attacks to being possible. The extra \ added will not be inserted in the database, it will only make it possible to insert text with for example the character ". (http://se.php.net/manual/en/function.addslashes.php, 2005 -04-20)

One important consideration with using addslashes() and similar functions like mysql_real_escape_string() is to be sure about that PHP directive magic_quotes_gpc is turned off. Otherwise the string will be escaped twice. The function get_magic_quotes_gpc() is in good use for determine this. (http://se.php.net/manual/en/function.addslashes.php, 2005-04-20)

```
<?php
if(get_magic_quotes_gpc())
{
        //do not do anything, ma   gic_qoutes_gpc is on
}
else
{
        //use addslashes()
}
?>
```

Example 32

3.6.5 Using htmlentities() to prevent SQL-injection

Another way of protecting from sql-injection attacks is to convert the input with the function htmlentities(). This function converts characters to html-specific characters. For example the character ' (singelquote) will be converted to &#039, assumed that the second argument for the htmlentities-function is set to ENT_QUOTES, which is a way of telling the function to also include both single and double quotes in the conversion (http://se2.php.net/manual/en/function.htmlentities.php, 2005-04-29).

```
$user = $_POST['user'];
$pass= $_POST['pass'];

$user =  htmlentities ($user, ENT_QUOTES);
$pass = htmlentities($pass, ENT_QUOTES);

$question = "select uid,username from login where username = '$use   r' and password =
'$pass'";
```

Example 33

If a malicious user would now enter *x' OR username = 'Johnny* in the password field, the comparison in the database would look *x' OR username ='Johnny* instead, and would therefore not return any valid result, since the singe quote can no longer break the origin SQL-statement.

3.7 Usage of bots

A very common feature on different kind of web pages is a poll. It can be used to get the users opinion about the web page or just a feature to make the homepage more alive. The example 34 is an example of the code, which is the form for the poll where the users enter their opinion.

```
<form method="post" action="vote.php">

Which car do you like best?<br>
<input type="radio" value="1" name="poll"> Volvo <br>
<input type="radio" value="2" name="poll"> Ferrari <br>
<input type="radio" value="3" name="poll"> Fiat <br>
<input type="submit" name="Submit" value="Vote">
</form>
```

Example 34

Someone who wants to write a bot that votes plenty of times for one of the options can easily write a script based on the information found in the source code from the form. A simple bot is shown in example 35.

```
<?php

$pquery = "poll=3"; //the option of vote
$path= "/voting/vote.php"; //the path to where the script recieveing the data is
situated
$host = "www.somehost.com"; //the host of which to connect
$out= "POST $path HTTP/1.1 \r\nHost: $host\r\nContent-type: application/x-www-
form-urlencoded\r\nUser-Agent: Firefox 1.0\r\nContent-length:
".strlen($pquery)."\r\nConnection: close\r\n\r\n$pquery"; //creates a post
submission


        for($i=0;$i<=10;$i++) //sending the post submission ten times
        {
                $fp = fsockopen("$host",80,$errno,$errstr,3  0);
                if (!$fp)
                echo "$errstr ($errno)<br/>\n";
                fwrite($fp,$out);
        }
  while (!feof($fp))/
  {
    echo fgets($fp,128);
  }
  fclose($fp);

?>
```

Example 35 (http://se.php.net/manual/sv/function.fsockopen.php, 2005-04-16)

This bot is simply running ten times in a for-loop and for each time it is posting a new submission with the option two chosen i.e. Volvo. What we want to post is specified in the variable *$pquery*. The variable *$path* is the path to where the script is situated that receives the answer of the poll. The *$host* variable is the host where to connect.

The script vote.php, which is processing the data, is expected to get the data from the form shown in example 34. If the form sends the data, the web browser of the client sending the data will create the post request, the user does not have to care about this. What we do in this script is sending the same post request over and over again as many times as the creator of the script wants to. The script works as an own client, sending the same data as if it would have been a user client sending the data from a form (http://phpsec.org/projects/guide/2.html#2.2, 2005-04-30).

3.7.1 Protect from bots using CAPTCHA

Lets say, hypothetically that our poll script described in "usage of bots" is a very important poll with many people having different opinion and there is a risk somebody creates a bot to vote for one certain opinion plenty of times.

Protecting the poll from being a matter of who creates the best bot and instead get the opinion that is most common, CAPTCHA is a good choice. The bot can not read the CAPTCHA phrase presented in an image, but most humans can, which tells that 'this is a human, let him/her vote"

There are many different free CAPTCHA script available for PHP. One of them is hn_captcha available at http://hn273.users.phpclasses.org/browse/package/1569.html (2005-04-27). It is a CAPTCHA PHP class released under the GNU general public license that means it is free to use or modify the script as long as the source code is made available for others. (http://www.opensource.org/licenses/gpl-license.html, 2005-04-27). Showing how to use it in here would be too much but the whole poll script is showed in appendix 1.

The basic principle is that we create a captcha-init array, to define properties for the CAPTCHA picture. Example 36 is the script receiving the information from the voting form where the user enters their choice (Example 34).

```
CAPTCHA_INIT = array(
        'tempfolder'    => $_SERVER['DOCUMENT_ROOT'].'/',
        'TTF_folder'    => $_SERVER['DOCUMENT_ROOT'].'/TTF/',
        'TTF_RANGE'     => array('COMIC.TTF','JACOBITE.TTF'),
        'chars'     => 5,
        'minsize'       => 20,
        'maxsize'       => 30,
        'maxrotation'   => 25,
        'noise'         => FALSE,
        'websafecolors'     => FALSE,
        'refreshlink'   => TRUE,
        'lang'          => 'en',
        'maxtry'        => 3,
        'badguys_url'   => '/',
        'secretstring'  => 'Some very secret string',
        'secretposition'    => 15,
        'debug'         => FALSE);

            $captcha = & new hn_captcha($CAPTCHA_INIT);
```

Example 36 (http://hn273.users.phpclasses.org/browse/package/1569.html, 2005-04-27)

Here is some properties defined, for example where to put the CAPTCHA image (tempfolder) on the server, what font type to use (TTF_RANGE) and where they are located (TTF_folder) and how many tries the user will be given to enter the correct pass phrase (maxtry). This array is passed when creating an instance of the hn_captcha class.

These properties are all we have to care about when creating the CAPTCHA image; the hn_captcha class handles the rest. The script will now show an image look like this:

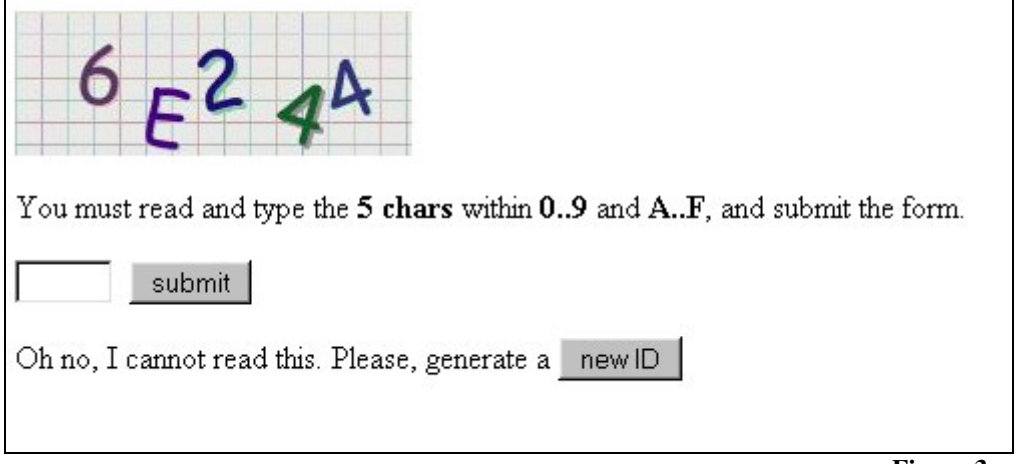

Figure 3

The button with the name "new ID" is a function to create a new image if the one showed is unreadable also for the human eye. When this is made, the user enters the pass phrase (6E244) into the text field bellow it.

Now the script determine what is going to happen if it is the right pass phrase and what is going to happen if it is the wrong phrase.

```
switch($captcha -> validate_submit())
{
        // was submitted and has valid keys
            case 1:
        process_data($vote );
        break;

// was submitted with no matching keys, but has not reached the maximum tries
            case 2:
        echo $captcha -> display_form();
        break;

// was submitted, has bad keys and also reached the maximum tries
        case 3:

echo "< p> < br> Reached the maximum tries of  ".$captcha -> maxtry." without
success!";
        echo "you don't not seem to be a human!";
        break;
                // was not submitted, first entry
        default:
        echo $captcha -> display_form();
        break;
}
```

Example 36 (http://hn273.users.phpclasses.org/browse/package/1569.html, 2005-04-27)

If it is the right phrase, the information will be processed, if it is the wrong phrase but the number of tries has not reached the maximum limit, the CAPTCHA image will be showed again. If the user cannot enter the right pass phrase in the scope of the specified amount of times, the text "you don't not seem to be a human!" will be displayed.

3.7.2 Other ways of protection

Of course data could still be added by a human running the script over and over again entering the right CAPTCHA phrase and voting. Combining the CAPTCHA image with some other method would also make this impossible.

Using cookie and IP address for controlling that the client already has voted can be useful to stop a spam bot voting over and over again. Using only cookie is not an appropriate method. Let's say the poll script has to create a cookie to validate if the client has voted or not. If there is a cookie with the correct information, the client can vote, otherwise not. Advance bots could circumvent this process by adding the cookie every time it votes and delete it afterwards. So, the bot can vote over and over again by deleting the cookie every time it has voted. (www.nettwerked.net/form-hacking.pdf, 2005-04-23)

A better method is to record the clients IP address. The script receives the IP address of the client and adds it to the database. If a client with the same IP address tries to vote more then once, the vote will be refused (http://www.nettwerked.net/form-hacking.pdf, 2005-04-23), the principle of that may look like this:

```
$ipnumber = $_SERVER[' REMOTE_ADDR' ];

if(!check_ip_nr_in_database($ipnumbe  r))
{
        //voting allowed
        add_ip_number_to_database($ipnumber);

}
else
{
        //voting refused
}
```

Example 37

The IP address is received with the command $_SERVER[' REMOTE_ADDR']. (http://www.php.net/manual/en/reserved.variables.php#reserved.variables.server, 2005-04-23) The IP address is sent to a function checking in the database if the address already exists, if not, the client is allowed to vote, otherwise the vote is refused.

Using IP address is only useful when a procedure is allowed only once. In cases like forums and guestbooks where the user must be allowed to perform a submission to the same script more then once, this method are not available to stop spamming.

3.8 Password protecting methods

A good way of decreasing the risk of that a user's password choice does not become a problem for guaranteeing the integrity and the confidentiality of the data is simply to force the user to choose a well-formed password. This can be done with a function validating so the user suggested password fulfils the criteria set up for the password. If the user enters a good password the user will be accepted, otherwise he/she needs to specify a new. The function in this example has this requirement to consider the password as valid:

- The password can contain the characters: A-Ö, a-ö, 0-9 and #¤%&_
- The password cannot be below six characters and not more then twelve
- The password must contain at least two digits or other symbols

```
function valid_password($password)
{
        if(strlen(!($password)< 6 || strlen($password)> 12))
        {

                $valid = ereg("[^ a -zåäö0-9A-ZÅÄÖ#¤% &_]",$password);
                if(!$valid)
                {

                        $digit_pattern = "([0 -9#¤% &_]).*([0  -9#¤% &_])";
                        $amount = ereg("$d igit_pattern",$password);

                        if($amount= = 1)
                                return true;
                        else
                                return false;
                }
                else
                {
                        return false;
                }


        }
        else
        return false;
}
```

Example 38

This code is an example of how a password validating function could look to determine if a user's password is a good choice or not. It first validates so the password is in the range between six and twelve characters. It then determine so the password does not include any invalid characters, and at last, it validate so it contains at least two of the characters 0-9 or #¤%&_, if all this requirements are fulfilled, it will return true and which mean it is an valid password.

This is a good protection against all the methods for discovering passwords. Still there is a risk that the password can be discovered, if the user have a password like "sweden999", it is not impossible this exists in the dictionary, used for a dictionary attack. Using brute force is theoretical possible but it would take unreasonably long time to try out all the combination possible since the password must be more then six characters. To be really sure that the brute force or dictionary method is not a choice, combining password protection with a CAPTCHA image will increase the security further (http://www.captcha.net/, 2005-04-28).

3.8.1 Encrypt password

Once the password reached the server it is a good idea to encrypt it with a one-way encryption function such as MD5. This will ensure that even if someone gains access to the password in the database, with the help of an SQL-injection for example, they will not have use of it. (http://www.phpadvisory.com/articles/view.phtml?ID=6, 2005-04-29)

When the user has created a login and has chosen a good password, it is time to insert it to the database together with the username. Using the md5() function, the password 3Fgs#3_ would be transformed to 64c746ee7e92e66b39104b8a48571533 for example.

```
if(valid_password($password) && valid_username($username))
{
        $valid_md5_password = md5($password);
        //create account with the username and password choused

}
else
{
        //ask the user to chouse a better password
}
```

Example 39

When the user login on the page, the entered password will first be transformed with the md5 function and then compared with the encrypted password in the database. (http://se2.php.net/manual/en/function.md5.php, 2005-04-29)

3.9 Session threats

Let us say that we have a web shop that sells gold. Gold which is an expensive metal is in our shop bought in the amount of kilograms. The customer is identified by a session identifier. Let us take a look at the example below:

```
//order.php
<?php
session_start(); // session is started
if (!isset($_SESSION['totalgold'])) { // if the session variable totalgold has not been set
  $_SESSION['totalgold'] = 0; // it is set with the value 0
} else { // if it has been set before, the amount bought in this ord      er will be added to totalgold
  $_SESSION['totalgold'] = $_SESSION['totalgold'] + $_POST['gold'];
}

?>
<html>
<body>
<form action="session.php">
<input type="hidden" name="PHPSESSID" value="<?php echo session_id(); ?>" />
Buy gold (kg):
<input type="text" nam e="gold" />
<input type="submit" name="buying" value="Buy"><br />

<hr>

You have totally ordered: <?php print $totalgold; ?>    KG gold.
</body>
</html>
```

Example 40 (http://phpsec.org/projects/guide/4.html#4.1, 2005-04-16)

The example above shows how this web shop handles orders. It saves the total amount of bought gold in a session variable. Every time the customer adds new gold to the order, the total amount of gold increases with that amount. This code above is not the most secure code in the world; it shows one of the simplest ways to initiate a session and storing session variables.
An attacker could try to use session guessing, session capturing or session fixation to hijack a session. Let us say that the attacker gets hold of an active session ID and fills it in the URL in his/her web browser like the example below.

```
http://goldshop.eg/order.php?PHPSESSID=821d6683829148a76cf5015be027b355
```

Example 41

When the attacker entered this URL, he/she got access to the customer's order in this web shop. The attacker can see the customer's total amount of ordered gold, and can add new gold to the customer's order.

3.9.1 Protecting against session threats

Sessions are hard to protect. If someone gets hold of a sessions ID, the rightful owner of that ID is at risk. But how you want to protect a session and if it is necessary depends on what your application does and how it is built (http://shiflett.org/articles/the-truth-about-sessions, 2005-04-22).
Make sure that the users have the ability to log out and end a session when they do not want to use the application anymore. An inactive session ID is no good for the attacker to steal. Automatically end a session if it is not active. The time as a session is active the same time is there for an attacker to get hold of the session. Short time is better for security, maybe not as good for the user (http://phpsec.org/projects/guide/1.html#1.1, 2005-04-22).
If you use the in-build session handler in PHP, you can regulate the session time with the PHP configurations *session.gc_maxlifetime* and *session.gc_probability*. The first mention configuration *session.gc_maxlifetime* sets the amount of time in seconds the session is active after the last access to it. *session.gc_probability* sets the clean up of inactive sessions. The value is set by percent. The value 100 means that it will clean up on every request on the server, which could lead to server slowdown. A lower value that fits your sever and amount of traffic should be set (http://www.zend.com/zend/tut/session.php, 2005-04-22).

It is more secure to use session cookies instead of sending the ID through the URL. But then it will only work for those who have cookies enabled, which means that sessions will not work for the users not using cookies. It is up to you, what you find is necessary. Using cookies will not be secure because someone can still guess or steal a cookie. But the session ID will not show up in some web statistics. If you decide to use the sessions ID in the URL make sure that it will not be showed as a referer in a web statistics log if you have outgoing links (Converse & Joyce 2004, p. 457-458).
If you want to use session cookies with the in-built session handler in PHP, you must enable it with the PHP configuration *session.use_only_cookies*
(http://se2.php.net/manual/en/ref.session.php, 2005-04-22).

The problem with sessions is that it only relies in one ID. If someone knows or gets hold of an active ID they get holds of everything that is connected to this session. A way to protect is to rely on more than just the ID.
Some applications uses the users IP address as an identifier. But you should know that more than one person/computer can have the same IP address, like a shared network at a company. The attack can come from the inside of the network, from a colleague. A user can also have different IP addresses for every request. That may make your visitors angry if they have to login on every page. When knowing this fact it is up to the developer of the system to consider if an IP check is worth it or not (http://shiflett.org/articles/the-truth-about-sessions, 2005-04-25).

When a user visits a webpage he does an HTTP request. There are some information about what is requested and some information about the user, like User-Agent. User-Agent contains information about the program used for accessing the webpage like web browser and operating system. This information can be used as an identifier. This is more secure than not using it, but the attacker can try to replicate this information or just use the same accessing programs as the session's rightful user is using.

Let us look at an example.

```
<?php
//order.php
session_start(); // session is started
if (isset($_SESSION['HTTP_USER_AGENT'])) // if the session variable is set before
{
   if ($_SESSION['HTTP_USER_AGENT'] != md5($_SERVER['HTTP_USER_AGENT']))
   // if the users sessions varible is not the same as the encypted Use      r-Agent from the User
   {
      //redirect to the login
      exit;
   }
}
else //if the session variable has not been set before
{
   $_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
   //session variable encrypted and set
}

if (!isset($_SESSION['totalgold'])) {  // if the session variable totalgold has not been set
  $_SESSION['totalgold'] = 0;// it is set with the value 0
} else {  // if it has been set before, the amount bought in this order will be added to totalgold
  $_SESSION['totalgold']= $_SESSION['totalgold']+ $_REQUEST['gold'];
}

?>

<html>
<body>
<form action="order.php">
<input type="hidden" name="PHPSESSID" value="<?php echo session_id(); ?>"/>
Buy gold (kg):
<input type="text" name="gold"/>
<input type="submit" name="buying" value="Buy"><br/>
<hr>
You have totally ordered: <?php print $totalgold; ?>    KG gold.
</body>
</html>
```

Example 42 (http://shiflett.org/articles/the-truth-about-sessions, 2005-04-25)

The example above shows how the User-Agent that the user have is compared against the User-Agent stored as a session variable. If the session variable was not set before it will be set as the User-Agent that belongs to the user that is requesting the page. This is not a complete security solution, but it makes it a little more secure because it is another obstacle for the attacker to get pass (Ibid).

When it comes to session fixation you should also protect your application with a regenerated session ID after logging in or when the users privilege on the webpage changes. This can be done with the help of this code:

```
<?php // after logging in where password and username is correct
session_regenerate_id();   // a new session ID is regenerated
$_SESSION['login'] = true; // login becomes true
?>
```

Example 43 (http://shiflett.org/articles/security-corner-feb2004, 2005-04-25)

By regenerate the session ID, the fixated ID is made useless, when the person that logged in gets a new sessions ID and the attacker is stuck with his made up ID (http://shiflett.org/articles/security-corner-feb2004, 2005-04-25).

When developing a website with sessions you should not think that your application is completely secure. You should assume that some of your user's sessions will be hijacked. While developing, this should always be considered. It is up to the developer to decide if the information and the functions that would be available when hijacked is that secret or harmful that it should never get in the attackers hands. You can always force the user to fill in his password again when very important functions or secret information is in use (http://www.sitepoint.com/article/anthology-2-1-access-control/3, 2005-04-29).

3.10 File uploads

It is pretty simple to make a file uploading function. First, the setting *file_uploads* must be set as on in the php.ini file. We need to make a form where the files are going to be uploaded through. A simple form can look like this:

```
<form enctype="multipart/form-data" action="upload.php" method="post">
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Select a image (*.gif) on your computer:<br />
<input type="file" name="imagefile" />
<input type="submit" value="Send" />
</form>
```

Example 44 (Converse, Joyce & Morgan, 2004, p. 543)

When using a file uploading form, the enctype *multipart/form-data* must be specified for the form to be able to work (Converse, Joyce & Morgan, 2004, p. 543).
The hidden input field with the name *MAX_FILE_SIZE* should not be looked upon like a security feature because you can easily create another form with changed attributes. Instead, it is a good way to tell users that their file is to big before the whole file is uploaded (http://se2.php.net/features.file-upload, 2005-04-24).

There must be a PHP script that handles the upload. It could look like this:

```
<?php
// upload.php

$uploaddir = 'uploads/';//directory for uploaded files
$uploadfile = $uploaddir.basename($_FILES['imagefile']['name']);
//path including filename

if (move_uploaded_file($_FILES['imagefile']['tmp_name'],$uploadfile)) {
//file is moved to the directory if it was successful
  echo "File successfully uploaded.";
} else {
  echo "Upload failed";
}

?>
```

Example 45 (http://se2.php.net/features.file-upload, 2005-04-24)

The PHP script above moves the file if it is in the size limit specified in the form to the specified location. Use $HTTP_POST_FILES instead of $_FILE if a PHP version below 4.1.0 is used (http://se2.php.net/features.file-upload, 2005-04-24).
This is what is needed to make a file upload. But as this is just basic, there is no real security from an attacker.

3.10.1 How to make file uploads safer.

As mentioned before, *MAX_FILE_SIZE* can not be relied on. There are two different ways to make a more reliable solution.
You can limit the max size of a file upload with the PHP configuration *upload_max_filesize*. Default configuration is set to 2 MB.
But you can also limiting it by PHP code. The file size can be collected from *$_FILES['form_input_upload_name']['size']*. Let us look at an example:

```
<?php
$uploaddir = 'uploads/';// directory for uploaded files
$uploadfile = $uploaddir.basename($_FILES['imagefile']['name']);
//path including filename

if ($_FILES['imagefile']['size'] < 100000 && $_FILES['imagefile']['size'] != null) {
        if (move_uploaded_file($_FILES['imagefile']['tmp_name'],$uploadfile)) {
                echo "File successfully up loaded.";
        } else {
                echo "Upload failed";
        }
} else {
        echo "Upload failed";
}
?>
```

Example 46 (http://se2.php.net/features.file-upload, 2005-04-24)

If the file size is smaller than 100000 bytes, then the file should be uploaded to the right directory, else an error message is presented.
As mentioned in chapter 2.18, it is a risk if the users can upload every kind of file formats. In our example we wanted people to upload images in the gif format. The file information is collected through *$_FILES['form_input_upload_name']['type']* . A gif image file will have the value *image/gif*. If we add a check for this, it would look something like this:

```
<?php
$uploaddir = 'uploads/';// directory for uploaded files
$uploadfile = $uploaddir.basename($_FILES['imagefile']['name']);
//path including filename

if ($_FILES['imagefile']['type']=='image/gif' && $_FILES['imagefile']['size'] < 100000
&& $_FILES['imagefile']['size'] != null) {
        if (move_uploaded_file($_FILES['imagefile']['tmp_name'],$uploadfile)) {
                echo "File successfully uploaded.";
        } else {
                echo "Upload failed";
        }
} else {
        echo "Upload failed";
}

?>
```

Example 47 (http://se2.php.net/features.file-upload, 2005-04-24)

If the image is not a gif then the file will not be uploaded to the specified directory. This is one way to check the file type, another is to check the last digits of the file after the last dot e.g. gif. As with every user input, the data should be filtered from not allowed signs.

4 Conclusion

PHP security is a matter of effort in many cases. It is very easy to create a guestbook on a personal homepage that works, but it takes more effort to create one that is not vulnerable to XSS or SQL-injection for example. Security is not only a part of the programming phase of a system, it is important that security is a part of the system design and planning as well.

The chosen kinds of attacks and vulnerabilities written in this report, are what we think are the most common and interesting kinds of attacks and vulnerabilities. The programming language PHP will keep on developing, the attackers will probably keep on trying to attack PHP applications. There will most likely arrive new kinds of attacks and vulnerabilities that will be used widely, but also new methods of protecting PHP scripts.

It would be impossible to cover all different combination of how the vulnerabilities explained could be exploited, but with principle of how it works, it is also possible to understand how to protect a script from being a possible target of attacks. Based on the result in this thesis, we have found some general actions to take in order to create more secure PHP system. Thinking of these things will make the system less vulnerable to possible attacks.

Always filtering user input data - Filtering the data so only right data for right purpose is allowed. Filtering the data is not only useful for avoiding different kind of attacks, it is also important because it gives a higher reliability of the data entered by users.

Always escape data when inserting to a database - Escape the data when using it for insertion to a database will block the possibility to perform SQL-injections. This can be done either by turning the magic_quotes_gpc in the php.ini file to on, or using one of the methods addslashes() or mysql_escape_string().

Always control the origin of a form: Control so that the right form is the one sending the data to a script is a good way of making sure that the data comes from the right form and not a spoofed one. It will also protect the form from CSRF attacks.

Use CAPTCHA when possible - Using CAPTCHA to determine whether a user is a human or a computer will hinder bots to send data thousands of times.

Always initialize variables – Initializing variables, especially when register_globals is on, will stop variables to be unintended change by someone. Make it a habit, always initializing the variables even though PHP does not require it.

Use the superglobals – Using the superglobals will ensure that variables only can be set with data from the right environment. If data comes from a form using POST method, the superglobal POST shall be used to retrieve it.

We have found out how important it is to filter the data to be sure about its integrity, having just correct data for correct purpose is a foundation for a secure application. A web page where it is possible to perform an SQL injection attack cannot ensure the confidentiality of the data, when someone without access can walk straight inside from the backdoor. Ensuring that the data comes from the right source and has not been changed on the way is an important security concern. Being vulnerable to CSRF or spoofed form submission is examples where the origin of the data cannot be ensured.

There are many threats and a lot have to be done to protect a PHP script. But it will be worth it, when the users can use your application as intended, when someone cannot exploit vulnerabilities to attack your brand, when you do not have to close down your services while stopping and preventing attacks. Security is a must.

You have read in this thesis about how you can protect a PHP script from different kinds of threats. The threats have been structured in what they do and categorized in different chapters. But an attacker does not have to follow our guide of attacks. The attacker can try to make combinations of the attacks written in this report to reach a security breach. The attacker can try to find vulnerabilities in your script alone. If you program your application in a horrible way, who knows how many ways there are to attack your application. Likewise, the ways of protecting your script does not mean that you must follow our recommendations to the letter or that our examples are the right way to protect your PHP application. Most PHP scripts differs, are programmed in different ways and have different server configurations etc. You as a programmer has to look at your script, find out how and the best way to protect it. Hopefully this security guide has given you enough information to do this. Good luck!

4.1 Reflection about this report

This report contained what we think is the most common security threats against a PHP script that could be protected with PHP code. Of course you can argue if these are the most common threats, maybe there are other threats that are more important to be written about.

We have mainly used knowledge from Internet sources but also book sources that we believe are trustable to get the information about the problems and their solutions. There is a risk that these sources contain information that is not right, that they lack of complete information and that we interpreted them wrong.

The text written by us that explains and gives advices on different ways to protect your PHP script can contain faults, and maybe it is not the right way to protect your script or it is not a complete security solution.

The code examples that we have written about how you can protect your PHP application against an attacker, have in some cases been inspired by examples from another sources, but also in most cases transformed to fit the example we wanted to present. We have tested all examples to see if they work, and we do believe that all examples in the report work. But we do not claim ourselves as PHP security experts. There is a risk that these examples contain faults.

Even though all risks of faults in the report, we personally believe that the risk of faults are very limited. We do believe this is a good report which gives usable information about PHP security.

4.1.1 Goal fulfillment

We think that we fulfilled our official goal and the problem formulation: What can you do code-wise to protect your PHP script from common security threats? The report contains information about both the problems so that a general understanding about the threat can be understandable and how you can protect the script against threats with code.

We have also fulfilled our unofficial goal to learn about PHP security. We have learned a lot about security threats and how to protect against them. We are amazed about how many ways and combinations you can use to perform attacks against an application and how simple some of the attacks can be but cause much damage. We have learned the importance of security and why this should be a part of the whole developing process.

Hopefully our knowledge learned and understanding has been presented in this report in a way that can be understood and be committed to the memory of the readers.

4.2 Further research

In January 2005, the PHP security consortium was started to, according to themselves, "promoting secure programming practices within the PHP community" (http://phpsec.org/, 2005-04-28). It will be interesting to see how this consortium develops, because there seem to be a need for enlightening security issues concerning PHP. For future reading about PHP security we will recommend the web page www.phpsec.org, which will hopefully continue advising PHP developers. New versions and fixes of PHP are released constantly with new features and it seems like there is a future for PHP and also the security issues related to it.

There are a lot more aspects of PHP and web application security issues to cover then what is in the scope of this report. It is hard to find any statistics about how common these kinds of security vulnerabilities are, which would be an interesting topic to look deeper into. Another topic could be how security considerations are dealt with on companies developing systems in PHP; do they consider it at all?

5 Reference list

5.1 Books

Alexander, Michel (1996), *The underground guide to computer security*, Addison-Wesley
Anonymous (2002), *Maximum Security*, fourth edition, SAMS, Indianapolis: USA

Connolly, Thomas & Begg, Carolyn (2002), *Database systems A practical approach to design, implementation, and management*, Addison-Wesley, USA

Converse, Tim & Park, Joyce (2004), *PHP 5 and MySQL bible*, Addison-Wesley, Indianapolis: USA

Ek, Jesper & Eriksson Ulrika (2001), *Linux som internet-server*, Pagina, Göteborg: Sweden

Jonsson, Viktor (2001), *Webbprogrammering med PHP,* Studentlitteratur, Lund:Sweden

Patel. Runa & Davidsson, Bo (2003), *Forskningsmetodikens grunder*, Studentlitteratur,Lund:Sweden

Thurén, Torsten (2002), *Vetenskapsteori för nybörjare*, Liber, Malmö:Sweden

5.2 Internet sources

http://academ.hvcc.edu/~kantopet/php/index.php?page=php+form+data&parent=php+client+side, 2005-04-08
http://hn273.users.phpclasses.org/browse/package/1569.html, 2005-04-27
http://phpsec.org/articles/2005/text-captcha.html, 2005-04-14
http://phpsec.org/projects/guide/1.html#1.1, 2005-04-22
http://phpsec.org/projects/guide/1.html#1.4, 2005-04-02
http://phpsec.org/projects/guide/2.html#2.1, 2005-04-08
http://phpsec.org/projects/guide/2.html#2.3, 2005-04-18
http://phpsec.org/projects/guide/2.html#2.4, 2005-04-19
http://phpsec.org/projects/guide/3.html#3.2, 2005-04-08
http://phpsec.org/projects/guide/4.html#4.1, 2005-04-16
http://se.php.net/manual/en/function.addslashes.php, 2005-04-20
http://se.php.net/manual/en/function.str-replace.php, 2005-04-18
http://se.php.net/manual/en/ini.core.php#ini.variables-order, 2005-04-04
http://se.php.net/manual/en/reserved.variables.php#reserved.variables.server, 2005-04-18
http://se.php.net/manual/sv/faq.general.php, 2005-03-30

http://se.php.net/manual/sv/function.fsockopen.php, 2005-04-16
http://se.php.net/manual/sv/history.php, 2005-03-30
http://se.php.net/session, 2005-04-16
http://se2.php.net/features.file-upload, 2005-04-24
http://se2.php.net/manual/en/function.addslashes.php, 2005-04-04
http://se2.php.net/manual/en/function.htmlentities.php, 2005-04-29
http://se2.php.net/manual/en/function.md5.php, 2005-04-29
http://se2.php.net/manual/en/function.mysql-escape-string.php, 2005-04-19
http://se2.php.net/manual/en/function.strip-tags.php, 2005-04-18
http://se2.php.net/manual/en/function.strlen.php, 2005-04-19
http://se2.php.net/manual/en/ref.info.php#ini.magic-quotes-gpc, 2005-04-04
http://se2.php.net/manual/en/ref.session.php, 2005-04-15
http://se2.php.net/manual/en/ref.sybase.php#ini.magic-quotes-sybase, 2005-04-04
http://se2.php.net/manual/en/security.globals.php, 2005-04-03
http://securityresponse.symantec.com/avcenter/venc/data/perl.santy.c.html, 2005-03-31
http://shiflett.org/articles/foiling-cross-site-attacks, 2005-04-06, 2005-04-08
http://shiflett.org/articles/security-corner-feb2004, 2005-04-16
http://shiflett.org/articles/the-truth-about-sessions, 2005-04-16
http://www.acros.si/papers/session_fixation.pdf, 2005-04-15
http://www.devshed.com/c/a/PHP/PHP-Security-Mistakes/1/, 2005-04-03
http://www.jmarshall.com/easy/http/#whatis, 2005-04-16
http://www.linuxexposed.com/Articles/Hacking/Introduction-to-SQL-Injection.html, 2005-04-07
http://www.ne.se/jsp/search/article.jsp?i_art_id=150823, 2005-04-01
http://www.ne.se/jsp/search/article.jsp?i_art_id=322447&i_word=s%e4kerhet, 2005-03-30
http://www.ne.se/jsp/search/article.jsp?i_art_id=676090&i_word=it-s%e4kerhet, 2005-03-30
http://www.nettwerked.net/form-hacking.pdf, 2005-04-23
http://www.opensource.org/licenses/gpl-license.html, 2005-04-27
http://www.php.net/manual/en/language.variables.external.php, 2005-04-05
http://www.php.net/manual/en/reserved.variables.php#reserved.variables.server, 2005-04-23
http://www.php.net/manual/en/reserved.variables.php, 2005-04-08
http://www.php.net/manual/en/security.globals.php 2005-04-18
http://www.phpadvisory.com/articles/view.phtml?ID=4, 2005-04-18
http://www.phpadvisory.com/articles/view.phtml?ID=6, 2005-04-29
http://www.sitepoint.com/article/anthology-2-1-access-control/3, 2005-04-29
http://www.zend.com/zend/tut/session.php, 2005-04-22
http://www-2.cs.cmu.edu/~biglou/captcha_cacm.pdf, 2005-04-27

Appendix 1 – Poll script source code

```
<?PHP

//this sc ript using hn_captcha class avaliable at http://hn273.users.phpclasses.org/browse/package/1569.html
//hn_captcha is distributed under the GNU GPL license
//http://www.opensource.org/licenses/gpl     -license.html

session_start();
$vote = '';

//if the session i  s set to something, give set $vote to the value of session
if(isset($_SESSION['voteing']))
{
        $vote = $_SESSION['voteing'];
}
if(isset($_POST['poll'])) //if the post  -variale poll is set
{
        $vote= $_POST['poll'];//set the var $vote to the value of poll
        $_SESSION['voteing'] = $vote;//set the session called vote to $vote

}


require_once("hn_captcha/hn_captcha.class.php");    //include the captcha class

        // ConfigArray
$CAPTCHA_INIT = array(
        'tempfolder'     => $_SERVER['DOCUMENT_ROOT'].'/';//wh ere to put the image, / in this case
        'TTF_folder'     => $_SERVER['DOCUMENT_ROOT'].'/TTF/';//the path to where the fonts is located
        'TTF_RANGE'    => array('COMIC.TTF','JACOBITE.TTF');//what fonts to use in the picture
        'chars'   => 5,//number  of chars to use
    'minsize'       => 20, //minimum size of chars
    'maxsize'       => 30, //maximum size of chars
    'maxrotation'    => 25, //some var for the apperance of the chars
        'noise'         => FALSE,//some var for the apperance of the chars
    'websafecolors'   => FALSE,//some var for the apperance of the chars
    'refreshlink'    => TRUE,//if a refreshlink shall be shown or not
    'lang'         => 'en',//language to use,   only english (en) or german (de)
    'maxtry'        => 3,//maximum number of tries
                        'badguys_url'    => '/',//where to go if too many tries is made
    'secretstring'   => 'Some very secret string',//a string to generate a md5 encryp    ted string
        'secretposition' => 15,
                        'debug'         => FALSE);//if debuginfo is going to be written out or not

$captcha = & new hn_captcha($CAPTCHA_INIT);   //instantiate the class
switch($captcha->validate_submit())  //a switch to determine t   he result of validation of a submission
{

        // was submitted and has valid keys
        case 1:
                process_data($vote);  //if the right passphrase is given, run the function process_data()
        break;

// was submitted with no matching keys, but has not reached the max       imum tries
        case 2:
        echo $captcha->display_form();
        break;


// was submitted, has bad keys and also reached the maximum tries
        case 3:

echo "<p><br>Reached the maximum tries of ".$captcha   ->maxtry." without success!";
        echo "you don't seem to be a human! ";
        break;
                // was not submitted, first entry
        default:
        echo $captcha->display_form();
        break;
}
```

```
function process_data($vote)
{
if(isset($vote))  //if the is $vote (containing the given option) is set to something
{

   if($vote==1) // if vote is one
   {
     $queryAmount = "select nrofvotes from poll where op = 1"; // create a sqlquery
     $what = 1; // what is a var to determine the choise made 1,2 or 3
   }
   elseif($vote==2)
   {
      $queryAmount = "select nrofvotes from poll where op =    2";
      $what = 2;
   }
   elseif($vote==3)
   {
            $queryAmount = "select nrofvotes from poll where op = 3";
     $what = 3;
   }
   else  //since the value of vote only can be 1   -3 nothing happen
   {
            print "<br>Bad choise<br>";
   }
   if(isset($queryAmount))  //hopefully a question is "created"
   {
    $db = mysql_connect("localhost","","") or die("can´t connect!");
             $selectedDB = mysql_select_db("poll",$db);     //connecting to and choosing database
             $answer = mysql_query($ queryAmount,$db);
    $row = mysql_fetch_row($answer);
    $amount = $row[0];  //the amount of already answered votes in the db

    if($what>=1 && $what<=3)    //another control to determine so the choise is just 1,2 or 3
    {
            update($amount,$what); //call function update to update the amount of existing votes($amount)        to the
right choise($what)
    }
   }
}



}
function update($amount,$what)
{
            $amount++; //add one to amount
            $query = "update poll set nrofvotes='$amount' where op = $what"; //update amount for the right
option
     $db = mysql_connect("localhost","","") or die("can´t connect!");
                        $selectedDB = mysql_select_db("poll",$db);     //connecting to and choosing database
                        mysql_query($query,$db);
                        session_destroy();
            retrive(); //call function to display the stats of the poll
}
```

```
function retrive()  //function to display result
{
          $query = "select sum(nrofvotes) from poll";
     $db = mysql_connect("localhost","","") or die("can´ t connect!");
     $selectedDB = mysql_select_db("poll",$db);
     $total = mysql_query($query,$db);
     $row = mysql_fetch_row($total);
     $total = $row[0];
     $anotherquery = "select * from poll";
     $answer = mysql_query($anotherquery,$db);

     while($ans = mysql_fetch_row($answer))
          {
          $select = $ans[0];
          if($select == 1)
       {
                                        $nr = $ans[1];
                                        $percent = ($nr/$total)*100;
                                        $percent = round($percent);
                                        print "< br> Volvo: $nr [ $percent % ]";
       }
       elseif($select == 2)
       {
                                         $nr = $ans[1];
                                         $percent = ($nr/$total)*100;
                                         $percent = round($percent);
                                         print "< br> Ferrari: $nr [ $percent % ]";
       }
       elseif($select == 3)
        {
                                         $nr = $ans[1];
                                         $percent = ($nr/$total)*100;
                                         $percent = round($percent);
                                         print "< br> Fiat: $nr [ $percent % ]";
       }

     }
}
?>
```

Växjö
universitet
Matematiska och systemtekniska institutionen
SE-351 95 Växjö

tel 0470-70 80 00, fax 0470-840 04
www.msi.vxu.se