Security Strategies for a Web-Based Student Peer Review System

Zachary Forster, Isaac Schemm, David Spiegel, Matthew Wisby,
Joline Morrison and Mike Morrison
Department of Computer Science
University of Wisconsin-Eau Claire
Eau Claire, WI 54702
Corresponding author: morrisjp@uwec.edu

Abstract

Peer review is a proven learning approach that allows students to observe and critique different solutions to a problem as well as to receive feedback on their own work. This paper investigates the security requirements for implementing a Web-based application that supports student peer reviews of class assignments. We investigate the capabilities and associated costs of using the secure hypertext transfer protocol (HTTPS), Web-based encryption using JavaScript and server-side programming, and implementing and storing passwords in a database using hash codes. We also consider the pros and cons of simply not addressing security in the application.

1 Introduction

The Web provides a popular environment for hosting and distributing user applications. However, it also poses security challenges in terms of authenticating users and distributing sensitive data. In this paper, we explore different approaches for implementing security in a Web-based application that supports student peer reviews of class assignments. Specifically, we investigate the capabilities and associated costs of using the secure hypertext transfer protocol (HTTPS), Web-based encryption using JavaScript and server-side programming, and implementing and storing passwords in a database using hash codes. We also consider the pros and cons of simply not addressing security in the application.

The first section explores security approaches in existing peer review systems. The next section explores the security approaches of HTTPS, web-based encryption, and database password implementation using hash codes, and describes how we have implemented these approaches in our peer review system. The final section investigates the risks of ignoring security in this type of application.

2 Background: Peer Review System Security

Peer review is a common educational practice to promote accuracy, communication, and learning. There is a commonly held belief that peer review prepares students for real-world situations, including giving or receiving critiques, which are invaluable skills. Computer-based systems provide a way to automate the peer review process, and many articles describe this approach [e.g., 8, 12, 19, 22, 23]. These papers discuss a wide range of topics, ranging from the very broad, including the benefits of peer review and the different forms it can take, to the minutiae, such as important steps to be implemented or details of separate graphic interfaces (i.e.: Student View, Instructor View, etc.). Unfortunately these papers contain surprisingly little insight regarding the implementation of security measures within these web applications. A study done by Giuliani et. al. [9] found that, of the five most popular database products on the market, all of them sent queries and usernames unencrypted over a network. These researchers were then able to view this information with little to no difficulty. Furthermore, some of the products revealed the length of the password, as well as the actual password itself.

In the age of the internet, the Web provides an attractive medium for automating the peer review process. In order to run a secure web application, encryption and other security techniques must be specifically implemented, but previous studies on peer review systems do not address web-based security approaches or the consequences of choosing not to use them. We explore these topics with our research.

3 Security Approaches

3.1 HTTPS

The Hypertext Transfer Protocol Secure (HTTPS) protocol allows encrypted Web connections by funneling the standard HTTP protocol through an SSL/TLS encryption layer [11]. When visiting an HTTPS domain, the web browser receives a digital certificate from the server. A digital certificate contains the certificate holder's public key (which will be used both to encrypt communication *to* the server and decrypt communication *from* the server), as well as an expiration date and the digital signature of a trusted certificate authority (which is made using the certificate authority's private encryption key) [5]. A certificate authority is a trusted third party that verifies the identity of the certificate holder [6]. The user's web browser checks to verify that the certificate was signed by a recognized certificate authority from a list built into the browser. It is possible to add a custom certificate authority to the browser as well; we’ll describe this process in a later paragraph.

When setting up our Web-based peer review system, we will want to use HTTPS to ensure security. There are two approaches for obtaining a digital certificate. One option is to get a certificate signed by an authority recognized by major web browsers; however, signed certificates aren't free, and it might be difficult to get a certificate for a website that we are hosting within our university’s domain.

The other option is to have a certificate that is either self-signed or signed by a free service such as CAcert (http://CAcert.org). If we use this approach, major browsers will display a warning page when a user requests the site that states that the web site may not be legitimate. This is because without a signed certificate, there's no way to verify the web site is real and not a scam. Communication between the server and browser will still be encrypted, but the web browser warnings try to steer the user away from visiting the site, and they often aren't user-friendly. For example, in Firefox 10, the error screen has a clearly visible “Get me out of here!” button, but requires the user to click “I Understand the Risks” and manually add an exception in order to visit the site.

Because our proposed Web-based peer review system is primarily designed for computer science students who are used to following directions for accessing software, and because of the cost of getting a signed certificate both in money and in research effort and time, we will probably use a self-signed certificate to at least provide some degree of security, unlike unencrypted HTTP. Although browsers will display an error message, most have a configuration approach for permanently trusting a self-signed certificate for a given web site. In Firefox, the “permanently store this exception” check box will keep users from seeing the error message every time they visit [1]. For Internet Explorer, the user must follow a process to add the certificate as a trusted root certificate on the local computer. The Windows versions of Google Chrome and Safari use the Windows certificate store as well, so following the steps in Internet Explorer will also remove the error message on Chrome and Safari [4].

3.2 Web-Based Encryption

While there are many ways to encrypt data on a webpage, HTTPS is by far the method that would come to mind first and is the most widely known. However, there are other ways to encrypt data.

One such method is using a “no right click script.” Using a no right click script on your web page makes it impossible to access the context menu, which is the menu that appears when the user clicks the right mouse button. This menu enables users to access the source code used to render the page, so using such a script prevents the user from seeing how the page was written. However, there are many ways to get around this, as the context menu can be accessed via the keyboard as well. Developers can create scripts that block the context menu from every way, but this usually just annoys users as most of the information that pages try to hide isn't accessible from the context menu [14]. Because this method isn't very reliable and makes the website more difficult to use, we probably will not use it.

Another encryption approach is Pretty Good Privacy (PGP), which is used for encrypting emails. PGP has since turned into OpenPGP, and the developer's website claims the software is "the most widely used email encryption in the world" [15]. This approach could be helpful for the peer review system because it could be used to encrypt the emails which contain student passwords. It is superior to a no right click script because it would not interfere with the web site's usability.

Another popular encryption algorithm is Blowfish, which is unpatented and free. It was created in 1993, and has been gaining popularity as a strong algorithm [18]. There are also many other encryption algorithms, many of which must be purchased, that claim to make web pages secure. Many different studies have been done on various algorithms, but each follows the basic pattern of web encryption by encrypting data and providing a key for users who are allowed to see the information.

Although there are many algorithms that work, it is often possible for visitors to the website to find the code for the algorithm used because it is often contained in one of the source files connected to the web site. It is possible to "minify" files to reduce the code to the minimum number of characters possible, but if someone is very determined they could still spell out the original code from the minified version. This means that they can then crack the algorithm and obtain the secure information. While it is likely that no one will be this determined to view the data the our peer review system uses, it would still be best to protect the student passwords so they can get the most out of the system. Because of this, HTTPS is the method we have chosen to use because of its reliability and safety.

3.3 Database Password Storage

Security goes beyond sending information back and forth between the web browser and the server. Information stored in the database that a web site uses must also be protected so that it can’t be read by those who shouldn’t be allowed to see it. In our system, this includes student and instructor passwords, which shouldn’t be stored in plain text in the database or else they would be too easy to read. One method to protect such information is hashing.

Hashing converts strings of data, disregarding length, into other small strings of data that usually contain both letters and digits. To “hash” a password, one must use an algorithm that is impossible or close to impossible to reverse. There are many commonly used hashing functions, including MD5, CRC32, and SHA-1 (Secure Hashing Algorithm 1). Essentially, the user registers by entering his or her password. That password is then run through the hashing function, and the resulting hashed password is stored in a database. When the user logs in and types his or her password, the password string is run through the same function, and the resulting hashed password is compared with the stored value [10].

One minor issue that must be considered is hash collision, which occurs because there are a finite number of possibilities for a hashed password. There are far more possibilities for different passwords, so sometimes multiple passwords end up resulting in the same hashed password, which decreases the time required to crack the password. In order to lessen the chance of this happening, one must to widen the range of possible hashed passwords by using a function that creates 128-bit or 160-bit hashes, such as SHA-1, which is the current standard. In fact, ever since Chinese cryptographers created an algorithm to speed up collision detection for SHA-1, the functions SHA-256 and SHA-512 seem to be the way of the future. These functions create even wider ranges of possible hashes, and dramatically decrease the feasibility of trying to find collisions. With today’s algorithms, hash collisions are exceedingly rare, but technology is always advancing, so security measures must also advance [10].

Another issue is that tables can be created by “calculating the hash values of commonly used words and their combinations” if a hacker has the algorithm. These are called rainbow tables, and they can be used to easily find all of the password hashes in a database. One way to avoid this happening is to add a unique “salt,” which is a random combination of letters and numbers to be added before each password, so that the password will not be on any rainbow table [10].

The final issue to overcome is the fact that today’s computers can simply try every possible password until they find one that works. This is called a brute force attack. This can be fixed by using an algorithm with a cost parameter, which basically determines the number of times that the algorithm will run. This can also be accomplished with a simple loop. This may be unnecessarily redundant, but it would make brute force attacks impractical because of the amount of time they would take. The CRYPT function accomplishes this [10].

We will not address details of the actual hashing algorithms here, partially because one does not actually need to know them in order to use them. The functions are included in many common languages, such as PHP [7] and Java. We plan to use these functions in our Web-based peer review system to store user passwords and other information of a personal nature.

4 Risks of Ignoring Security

All of the security approaches described so far involve a cost in terms of developer time and increased system complexity. As a result, we need to consider the risks of ignoring security in this system.

While our system will not contain sensitive information such as birth dates, social security numbers, or credit card numbers, it will contain logon credentials (user names and passwords) for instructors and students. It will also contain student assignment solutions and peer reviews student solutions. Malicious users could log on using stolen credentials and steal or delete student assignment solutions, or view, modify, or delete peer reviews. This would seriously undermine the integrity and credibility of the system.

If we choose not to encrypt information being sent and received by our peer review application, how likely is it that our system would come under attack? The primary threat is through packet sniffing. A packet sniffer can intercept and read information traveling through any shared communication media (like wireless networks). According to Qadeer et. al. [17], packet sniffers were not originally intended to be used maliciously, but instead were supposed to be used as a powerful tool for network administrators. Ansari et. al. [2] assert that packet sniffers can be used to help maintain networks by solving communication problems, logging network traffic, analyzing network performance, and detecting intruders. Unfortunately, they can also be utilized to maliciously intercept sensitive data such as passwords.

Packet sniffers can operate in different ways that depend on the type of network the packet sniffer is observing. If the network is an old-style hub-based LAN, then a message is sent to all of the computers on the local network. In order for a packet sniffer intercept messages on this type of network, the network interface card (NIC) must be set into “promiscuous mode.” The NIC is tasked with accepting or discarding messages depending on whether the message is addressed to the machine. A NIC in promiscuous mode does not discard any packets, but instead accepts all sent packets. This style of sniffing is largely irrelevant today because most modern network do not use hubs anymore.

Most modern networks use switches, whereby a computer only has access to messages directed to it. In order to sniff on this type of network using promiscuous mode, a packet sniffer most install itself at the main “pipe” for the network. This main “pipe” can be located between the router and CSU/DSU. However, this vital location is typically not easily accessed without authorization [21]. There is a way around this, and that is to set the NIC into “monitor mode”. This mode is very similar to promiscuous mode, but it can also intercept packets that weren't even addressed to the related access point [16].

A solution to this problem is to encrypt the data being sent in the packets. This does not stop the packet sniffer from intercepting messages, but prevents the sniffer from easily reading the information. The sniffer would not have the correct key to break the encryption and thus the message would be meaningless. The computer for which the message was directed would possess the key and would be able to decrypt the data. Marthur and Trappe [13] describe that a determined sniffer might be able to break the encryption depending on the strength of the encryption method. This makes it necessary for us to find a decent encryption method that will serve as a deterrent to stop any potential sniffer.

5 Conclusions, Reflections, and Future Directions

After exploring the different options for web-based encryption, we have decided that it is necessary to implement some type of encryption in our peer review system. Because of its security, we have decided to use an HTTPS protocol for our peer review Web site, and to use hashed database passwords. Although there is no vitally sensitive information in our system, we must keep the student and instructor logon credentials safe in order to maintain the system's integrity and credibility. The purpose of creating the software was to help students become stronger in their computer science skills, and to ensure this we want to keep the system safe from any kinds of outside sources.

References

[1] *Adding a security exception. Power Admin.* Web. 14 Mar. 2012. http://www.poweradmin.com/help/sslhints/FireFox.aspx>

[2] Ansari, S., Rajeev, S. G., & Chandrashekar, "H. S. Packet sniffing: a brief introduction". *Potentials, IEEE, 21*(5), pages 17-19, January 2002.

[3] Cunningham, RT. *Using Self-Signed SSL Certificates with Modern Web Browsers. Untwisted Vortex.* Web. 14 Mar. 2012. <http://www.untwistedvortex.com/selfsigned-ssl-certificates-modern-web-browsers/>

[4] *Description of digital certificates. Microsoft Support.* 23 Jan. 2007. Web. 14 Mar. 2012. <http://support.microsoft.com/kb/195724>

[5] *Digital certificate*. *SearchSecurity.* July 2000. Web. 14 Mar. 2012. <http://searchsecurity.techtarget.com/definition/digital-certificate>.

[6] *Digital signature (electronic signature)*. *SearchSecurity.* October 2000. Web. 14 Mar. 2012. <http://searchsecurity.techtarget.com/definition/digital-signature>.

[7] Eastlake, Donald E and Paul E Jones. *US Secure Hash Algorithm 1 (SHA1)*, September 2001. Web. 13 March 2012. <http://tools.ietf.org/html/rfc3174>.

[8] Gehringer, Edward. Electronic peer review and peer grading in computer-science courses. *SIGCSE Bull*, 33(1), February 2001.

[9] Giuliani, Matthew, Eric Lobner, and Paul J. Wagner. "Investigating Database Security in a Networked Environment." (2006): 1-8. Print.

[10] Guzel, Burak. *Understanding Hash Functions and Keeping Passwords Safe*, 17 January 2011. Web. 13 March 2012. <http://net.tutsplus.com/tutorials/php/understanding-hash-functions-and-keeping-passwords-safe/>.

[11] *HTTPS (HTTP over SSL or HTTP Secure)*. *SearchSoftwareQuality*. December 2000. Web. 14 Mar. 2012. <http://searchsoftwarequality.techtarget.com/definition/HTTPS>.

[12] Hundhausen, C., Agrawal, A., & Ryan, K. "The design of an online environment to support pedagogical code reviews". *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 182-186, 2010.

[13] Mathur, S., & Trappe, W. BIT-TRAPS: Building Information-Theoretic Traffic Privacy Into Packet Streams. *Information Forensics and Security, IEEE Transactions on*, *6*(3), pages 752-762, September 2011.

[14] *No Right Click Script*. *About.com*. The New York Times Company, 2012. Web. 12 Feb. 2012. <http://javascript.about.com/library/blnoright.htm>.

[15] *OpenPGP Alliance*. OpenPGP Alliance. Web. 12 Feb. 2012. <http://www.openpgp.org/>.

[16] "Packet Sniffing 101: Promiscuous Mode - HakTip." *Revision3*. Web. 14 Mar. 2012. <http://revision3.com/haktip/promiscuous>.

[17] Qadeer, M. A., Zahid, M., Iqbal, A., & Siddiqui, M. R. Network Traffic Analysis and Intrusion Detection Using Packet Sniffer. *Communication Software and Networks, 2010. ICCSN '10*, pages 313-317, 2010.

[18] Schneier, Bruce. *The Blowfish Encryption Algorithm*. *Schneier on Security*. Web. 12 Feb. 2012. <http://www.schneier.com/blowfish.html>.

[19] Silva, Elaine, and Dilvan Moreira. WebCoM: a tool to use peer review to improve student interaction. *Journal on Educational Resources in Computing*, 3(1), March 2003.

[20] Sitthiworachart, Jirarat, and Mike Joy. "Effective Peer Assessment for Learning Computer Programming." (n.d.): 122-25. Print.

[21] Tiller, J. S., & Fish, B. D. Packet Sniffers and Network Monitors, Part 2. *Information Systems Security*, *9*(3), page 17. 2000.

[22] Trivedi, Abhijeet, Dulal C. Kar, and Holly Patterson-McNeill. "Automatic Assignment Management and Peer Evaluation." (2003): 30-36. Print.

[23] Zhi-Feng Liu, Eric, Sunny S.J. Lin, Chi-Huang Chiu, and Shyan-Ming Yuan. "Web-Based Peer Review: The Learner as both Adapter and Reviewer." (2001): 246-50. Print.