Mark Kardash

Professor Carl Eyler

SDEV 460 7380: Software Security Testing

Homework 3: Authentication, Authorization, and Security Management Controls

23 April 2024

University of Maryland Global Campus

In this document, I will be presenting my work for SDEV460 Assignment 3. The assignment is based on Week 5 and Week 6 readings that cover the topics of Authentication, Authorization, and Session Management. All three of these aspects are extremely important in software development. Should the first two fail, sensitive system data could end up in the hands of attackers. If session management is not done effectively, it can make each login session unsafe for the user, exposing the system to various threats. This document will be split into several sections, each one based on the task that is requested by the assignment instructions.

**Test Role Definitions (OTG-IDENT-001)**

As requested by the instructions, there are two roles that are needed for this application: Tutor and Student. The first one would be given permission to perform various tutoring-related activities, such as designing schedules, assignments, and reviewing grades. The Student role would perform other functions, such as make appointments, upload assignments, etc. In the chart below, the two roles are listed, along with their objects, permissions, and constraints.

|  |  |  |  |
| --- | --- | --- | --- |
| Role: | Object: | Permissions: | Constraints: |
| Tutor | Tutor/Student Session | Read, Update | Tutors can update and cancel sessions only from their own account, accept appointments scheduled by students, send information in emails to particular students with whom the appointment is scheduled, and cancel only those appointments associated with them. |
| Student | Tutor/Student Session | Read, Update | Students can schedule, update, and cancel appointments only from their own account, send information to tutors with whom they have appointments scheduled, and cancel only those appointments associated with them. |

**Test User Registration Process (OTG-IDENT-002)**

User registration is a sensitive part of any application, as it involves exposing personal information to the system and validating one’s identity correctly to avoid breaches by malicious agents. And because the methods of cybercriminals are becoming more and more elaborate, developers must find a way to ensure that the person attempting to register is indeed who they claim to be, rather than an effective impersonator. Of course, the complexity of user identification varies based on each system’s security requirements. According to OWASP testing guide, although some companies use automation to make the registration easier, most of them do it manually (OWASP Testing Guide Version 4.0, 2011). When testing a registration system, it is therefore vital to design cases that address every aspect of identifying the user, from credential strength to the validity of their actual name. The Guide requires that registration processes within an app meet business and security requirements, while also stressing the aforementioned importance of correct user validation. To help understand this topic better, the Guide offers several questions to ask while performing this test, which we will attempt to answer below, based on our own registration system, designed for our fictional application. Before we do this, however, we will explain how the system works.

Proposed Registration System:

At the beginning of the school year, when students are just introduced to their virtual tutoring portal, they may press a button on the top right, to make their account. The same goes for potential tutors. Once that button is pressed, they are presented with a window asking for their information. The window includes the following fields: First Name, Middle Initial, Last Name, Student/Tutor ID Number, Date of Birth, Address Line 1, Address Line 2, City, State, Zip Code, Student/Tutor Phone Number, Username or Email, Password, and Confirm Password. Once all of these fields are filled out, the user can press the “Create Account” button. This will save all information provided into the institution’s database. From then on, each time they want to access their account, the user will press the “Login” button, after which they will be required to enter only their username and password.

Password Complexity:

The user will be required to create a password that is at least 8 characters long. The password must contain at least 1 uppercase letter, at least 1 lowercase letter, at least 1 special character (ex. @, #, &, %, etc.), and at least 1 number. An option is also to limit the password to 15 characters. However, this may be unnecessary, as longer passwords are usually more secure.

Answers to Questions:

Meeting Requirements:

1. Can anyone register for access?

Only those with a valid Student or Tutor ID Number (issued by the institution) can register for the portal. Entering an invalid student ID number will throw an “Please enter a valid student/tutor ID number” error message. Students can find their ID number on various forms sent to them by the institution.

1. Are registrations vetted by a human prior to provisioning, or are they automatically granted if the criteria are met?

Once the “Create Account” button is pressed, the system will perform a search for the provided ID number. If a student or tutor with a matching number is indeed studying, or teaching there, registration will be granted. If not, they will be asked to enter a valid number. The verification process will only take a few seconds.

1. Can the same person or identity register multiple times?

Due to an assignment of ID numbers to each portal member, each person can register only once.

1. Can users register for different roles or permissions?

There are two roles that one can register for: Tutor and Student. These options are presented in a small dropdown menu at the “Register” button. Each role will have different permissions, with some of them overlapping (see test matrix above).

1. What proof of identity is required for registration to be successful?

While all pieces of information (Name, Phone Number, etc.) are required to register, the recommendation is that the most important of them should be the First Name, Last Name, and ID Number. The reason for this will be explained within the answer to the next question.

1. Are registered identities verified?

The registered identities are verified by comparing the names of students or tutors, and their ID numbers, to those present within the institution’s records. If all three main criteria (ID, First Name, Last Name) match a record found in the database, the user is registered to the tutoring portal.

Validating Information:

1. Can identity information be easily forged or faked?

Forging of identity information is mostly unlikely due to reliance on previously issued ID numbers. The only way someone unintended can gain access is if they somehow find out the user’s ID number.

1. Can the exchange of identity information be easily manipulated during registration?

Of course, threats like SQL injection and password guessing attacks are always a possibility. For this reason, several recommendations for better code security are made below.

Questions Source: OWASP Testing Guide Version 4.0 (2011).

Security Recommendations:

According to Kime (2023), there are 5 potential ways to protect software from password guessing attacks. These are as follows:

1. Filtering Database Inputs: This involves filtering any code from user input that may potentially be malicious.
2. Restricting Database Code: This involves preventing undesirable queries and keep the code simple to prevent malicious exploration.
3. Restricting Database Access: This means implementing access controls that will prevent any manipulation of the data by malicious agents.
4. Maintaining Applications and Databases: This would mean keeping the application updated, patching any possible bug, fault, or shortcoming. It also means making upgrades whenever necessary.
5. Monitoring Application and Database Inputs and Communications: As the name implies, this measure involves completely controlling all flow of information within the app, to prevent code injection.

(Kime, 2023).

As for password guessing attacks, the author will make some personal recommendations here, specifically related to the fictional application being examined.

1. Creating a Complex Password: As mentioned before, the tutoring portal will require the password to be at least 8 characters long, containing at least 1 uppercase letter, 1 lowercase letter, 1 special character, and 1 number. An option would be to also make inadmissible any passwords that look too similar to the user’s own name.
2. Change Password Every Term (4 Months): Changing the password every 4 months (approximately equivalent to a school term) will lessen the chances of someone getting a hold of it, thus keeping the user data more protected.
3. Not Reusing Old Passwords: The system should not permit the user to utilize any password they previously had. It should notify them, for instance, with an error message (ex. “Password cannot be one that was used previously). This will ensure that attackers don’t wait for the right moment to guess a password when a user brings back an old one.

Some additional, general recommendations for password, database, and registration security would be to:

1. Encrypt all passwords and data: Encryption makes an attacker’s job more complicated and less enticing.
2. Install a firewall: Firewalls keep data protected from attackers, and finding ways to get past them is usually time-consuming.

**Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)**

The HTTPS protocol is one of the means often used in transferring data through encrypted channels, to avoid potential interception by malicious users. It uses TLS/SSL to ensure that a user is taken to the correct website, rather than an infected or dangerous one (OWASP Testing Guide Version 4.0, 2011). The HTTPS implementation process can be a complicated one, and the website Mindstick.com lists several best practices to make it easier. These practices are as follows:

1. Selecting the Right Type of SSL/TSL Certificate:

This will depend on the requirements put forth by the website. Types of certificates include Domain Validating (DV) Certificates (for confirming ownership of a domain), Organization Validation (OV) Certificates (which include organizational facts to strengthen domain validation), and External Validation (EV) Certificates, which directly show the employee’s name within a browser, being the highest form of validation available. It is up to the user’s judgement which certificate to use.

1. Implementing 301 Redirects:

This is an important practice in HTTPS implementation, as a 301 redirect informs the search engine that the website or app has been permanently moved to a steady protocol. This allows HTTP users to be redirected to HTTPS automatically, which prevents duplicate issues, and maintains search engine marketing.

1. Updating All Internal Resources and Links:

This involves making sure that all links are HTTPS-compatible, and, if not, replaced with proper ones. This could involve anything from inner scripts, to hyperlinks, to navigation methods. This practice will keep surfing level stable and consistent.

1. Updating All External Links:

External links are those that lead users to the application from outside sources. They also need to be regularly updated, as this both secures the system and makes for a better customer experience. Updating these links requires reaching out to external webmasters, and requesting that the links be replaced.

1. Updating Third-Party Services and Content Delivery Networks (CDNs).

Any delivery networks must be made HTTPS-compliant, and connection of all third-party scripts must be secure. This can help avoid capability protection vulnerabilities, and blended content material warnings.

1. Enabling HTTP Strict Transport Security (HSTS):

Implementing this practice can prevent man-in-the-center attacks and improve general application security. However, developers must configure HSTS very carefully, to avoid ability issues.

1. Implementing Perfect Forward Secrecy (PFS):

Even if a piece of personal information is exposed, this technique allows confidentiality of beyond periods, providing additional security to the HTTPS implementation.

1. Updating SSL/TLS Protocols:

All related protocols must be constantly updated as well, since hackers are always finding new ways to breach older protocols. Up-to-date protocols would mean the latest protection against attacks, within a stable, encrypted environment.

1. Monitoring Website Performance:

Because the performance of a website or application is often impacted by HTTPS implementation (due to the encryption it entails), it must be monitored constantly. Some tools used for monitoring include Google Page Speed Insights, which helps perceive development possibilities, and ensures that webpage load instances still function after transition to HTTPS.

1. Configure Robots.txt File:

Just as is the case with various links, the Robots.txt file must be replaced with one that is compatible with HTTPS. It must also be ensured that search engines can move slowly and index a secure model of the app. All directives must be made HTTPS-compatible.

1. Updating Google Analytics and Webmaster Tools:

All settings within Google Analytics and Webmaster tools must be replaced with HTTPS-compliant ones. In this way, information will be accurately tracked, and data provided about ranking of secure content by search engines.

1. Consistently Checking for SSL/TLS Vulnerabilities:

Checking for such vulnerabilities with tools like Qualys SSL Labs will examine the strength of the configuration. Any vulnerabilities detected must be addressed as soon as possible, to maintain a consistently high security record.

1. Create a Custom 404 Error Page:

A custom-made 404 Error page would be very useful in case of an HTTPS transition error, especially if it allows users to explore the website/application.

(MindStrick.com, 2023).

As for other ways to authenticate users, Johnson (2023), presents the following five alternatives to HTTPS:

1. Password-Based Authentication:

This one is not really an alternative, since passwords are used with HTTPS as well. It is the simplest method, yet also the least safe. As mentioned earlier within this document, hackers can always initiate password-guessing attacks, while the passwords themselves can be forgotten, reused, or be too simple. Most of this can be prevented with password reuse company policies and proper password storage.

1. Two-Factor and Multi-Factor Authentication:

Sometimes, a system may ask its users to use two or more authentication methods, such as a text message with a code in addition to the username and password. When the second factor is on a different device from the one used during the login, this is called out-of-band authentication. Since it involves more than just the original user credentials, this method is more secure, requiring hackers to obtain extra information. A second factor like biometrics or push notifications is even more reliable, since it cannot be breached as easily as email or text. This method’s only drawback is that it sometimes creates friction for the UX.

1. Biometric Authentication:

The effectiveness of this method, briefly touched upon above, lies in its unique nature. Rather than relying on information that can be obtained and copied, it relies on the user themselves, involving procedures like palm and fingerprint scanning, facial and iris recognition, and behavioral patterns. A benefit of this technique is that the user does not need to remember personal information, and the data they give is not easily obtainable by attackers. A drawback, on the other hand, is that not all technology has yet evolved enough to effectively handle biometrics. Additionally, not all users may feel comfortable with sharing biometric information.

1. Single Sign-On (SSO):

This technique involves using the same set of credentials for every application or website that the user logs into. The user’s account has an identity provider (IdP), which notifies the application that the user has been verified. The reduction in the number of credentials improves security, as the user now does not need to remember a lot of information. They also do not need to login multiple times, thanks to the UX, and will spend less time with the help desk should a problem arise with their password.

There are two drawbacks to this method. The first is how much time it takes to connect to all the applications needed. This can be disappointing, time consuming, and financially inefficient, especially if the application belongs to a large company. The second is that hackers could easily access a multitude of accounts if the IdP is somehow breached.

1. Token-Based Authentication:

This involves using a physical device, like a tablet or smartphone, to access the app. It can potentially offer a “password-less experience”, or allow users to verify credentials only a single time, within a specific timeframe, for login reduction. This method can be very effective against attackers, as, to bypass tokens, they would have to access the physical device, and have the correct credentials of the user. To effectively utilize token-based authentication, companies must trust employees to keep watch of their tokens, and design a reenrollment process if a token is forgotten or lost.

(Johnson, 2023).

**Testing for Default Credentials (OTG-AUTHN-002)**

Because of the frequent use of open-source and commercial software in current times, which requires minimum configuration, passwords and other credentials are often left on default. Due to the existence of special lists and patterns, default credentials are widely known within the development community, and thus can fall into the hands of cybercriminals (OWASP Testing Guide Version 4.0, 2011). There are many ways to go about guessing user credentials when perpetrating a password-guessing attack. The author will list those he would personally think about using below:

1. Trying Popular Default Usernames:

There are many usernames that are used as default in software with limited configuration. Most of them are based on the role of each user. For instance, in an administrator account, the username may be something like “administrator”, “admin”, and “root”. A tester may use “tester”, “test”, “testing”, and other variations. It is also very common to add numbers at the end of a username, such as “admin123”, “tester1”, “guest0”, “operator2”, and others. There could be many variations of the above usernames, but they are all commonly used when not replaced. Therefore, the author would look for default credential lists on the web, and try common usernames. To make the process easier, it would be automated, and use password-guessing software to expedite the process. The same value can also be used for both username and password.

1. Searching for Hard-Coded Passwords:

In some cases, though it is a frowned-upon practice, credentials are hardcoded into the application. Therefore, the author would view and examine the app’s code, to see if it contains references to a username or password. If not hardcoded, the credentials could also be written as comments in the program.

1. Attempt Using the Names of the Educational Institution, or Application, as Credentials:

Employees and employers sometimes use the name of their organization as a username or password. Therefore, the author would assume that the students and tutors using the app may do the same.

1. Guess the Organization’s Email Structure:

Many organizations, and especially schools, base the structure of their email addresses on a [studentname@institutionname.edu](mailto:studentname@institutionname.edu) pattern. The author would first visit the institution’s website to find a list of random student and tutor names (available anywhere, from a “Meet the Faculty” page to a “Student of the Month” announcement). Then, he would attempt to structure the names he selected following the format of the email, and multiple variations of it. Once this was figured out, brute force attack software would be used to crack the passwords. The attack would use common potential variables, such as dates of birth, years of graduation, etc.

(OWASP Testing Guide Version 4.0, 2011).

Among the potential resources to be used for this operation, aside from lists of default credentials (which could perhaps even be extracted from the program itself), are two categories of tools. Brief lists of them are shown below:

Password Cracking Tools (Most Popular):

1. iMobie AnyUnlock
2. CrackStation
3. Password Cracker
4. Brutus Password Cracker
5. Aircrack
6. Rainbow Crack
7. THC Hydra
8. Cain and Abel
9. Medusa
10. John the Ripper
11. ophCrack
12. WFuzz

(Software Testing Help, n.d.).

Brute Force Attack Software:

While most of the above tools can also be used for brute force attacks, there are a few extras that could also be useful. They include:

1. Aircrack-ng
2. L0phtCrack
3. Hashcat
4. DaveGrohl
5. Ncrack

(Shankdhar, 2020).

**Testing for Weak Lock Out Mechanism (OTG-AUTHN-003)**

Having a lockout mechanism on a website or application is an effective means against brute-force attacks. Most websites usually have a rule of locking the user out after 3-5 incorrect attempts, and keeping them out until either an administrator unlocks them, or the system does. When implementing lockout mechanisms, it is very important to ensure that they not only keep unauthorized users out, but also successfully let authorized users in (OWASP Testing Guide Version 4.0, 2011). To deal with a user logging in multiple times with incorrect credentials, the OWASP Foundation recommends a mechanism that locks the user out after 3-5 attempts, and then unlocks the account eventually. How and when the account will be unlocked depends on various things. In general, the OWASP Foundation proposes four different unlock mechanisms to deal with this:

1. Time-based lockout and unlock: As the name implies, this mechanism automatically unlocks the user’s account a predetermined amount of time after lockout. In most cases, this takes about 15 minutes.
2. Self-Service Unlock: In this mechanism, the user is given the ability to unlock the account themselves. The system sends an email with an unlock code to the email address provided by the student, and they enter it on the website to log back in.
3. Manual Administrator Unlock: In this approach, an administrator unlocks the account manually after some time passes.
4. Manual Administrator Unlock with Positive User Identification: This method is almost the same as the previous one. However, this time, the user needs to confirm their identity in various ways (Ex. Confirming name, answering security questions, etc.) before their account can be unlocked.

To handle logins with incorrect credentials for this particular application, the author would suggest using the time-based lockout and unlock mechanism. It is a common method, ensuring safety without being time-consuming. Below is a sample Java pseudocode for it. If we wanted to make it work in real life, we would need to add HttpServlet mechanisms, as well as HashMap:

import javax.servlet.https.HttpServletRequest;

import javax.servlet.https.HttpServletResponse;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpSession;

public class LoginServlet extends javax.servlet.http.HttpServlet {

private static final int MAXIMUM\_LOGIN\_ATTEMPTS = 3;

private static final long LOCKOUT\_DURATION = 900000

private static HashMap<String, Integer> attempts = new HashMap<> ( );

private static HashMap<String, Long> lockout = new HashMap<>( );

protected void studentCredentials(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String studentUsername = request.getParameter(“studentUsername”);

String studentPassword = request.getParameter(“studentPassword”);

User student = Database.getUser(studentUsername);

If (user != null && user.getPassword().equals(studentPassword)) {

if (lockout.containsKey(studentUsername) && lockout.get(studentUsername) > System.currentTimeMillisec( ) ) {

response.sendRedirect(“lockout.jsp”)

return;

}

if (attempts.containsKey(studentUsername) && attempts.get(studentUsername) >= MAXIMUM\_LOGIN\_ATTEMPTS) {

lockout.put(studentUsername, System.currentTimeMillisec( ) + LOCKOUT\_DURATION);

attempts.remove(studentUsername);

response.sendRedirect(“lockout.jsp”);

return;

}

HttpSession session = request.getSession( );

session.setAttribute(“student”, student);

response.sendRedirect(“home.jsp”);

} else {

attempts.put(studentUsername, attempts.getOrDefault(studentUsername, 0) + 1);

Response.sendRedirect(systemError.jsp”);

}

}

The above code sets the maximum number of attempts before lockout to 3, and the lockout duration to 900,000 milliseconds (15 minutes). It then uses HashMaps to track the attempts, and requests the student credentials as parameters. If the student enters the correct credentials, but the number of attempts has run out, then the lockout mechanism is activated, using a Servlet in a “.jsp” file. The system is notified, and the maxim lockout time is implemented. Via an HTTP session, the user is redirected to the application’s homepage, written as another “.jsp” file. If the user enters invalid credentials, they will be shown an error message, also as a “.jsp” file.

**Testing For Weak Password Policy (OTG-AUTHN-007):**

When it comes to creating a password for anything, there are usually three things that people want it to be: Complex, long enough to be secure, but short enough to remember. Personally, in terms of efficient password creation, the author would recommend what has been said in the paper before: A password at least 8 characters long, with at least 1 uppercase letter, lowercase letter, number, and special character. However, for academic certainty, we shall now look at the professional password and authentication recommendations given by the National Institute of Standards and Technology (NIST). The 15 guidelines and their explanations are listed below:

1. Using passwords at least 12 characters long.

The longer a password is, the more time it will take for any attacker to crack. The reasons behind this are quite obvious: Passwords of greater length are more chaotic, and contain more characters to guess.

1. Avoiding Commonly Used Passwords.

It was previously mentioned that default passwords, like “123”, “user”, or “password”, should never be used, as they are extremely weak, and among the first passwords that hackers would guess. Guessing them will be very easy, since there are lists of default passwords available for public access. NIST recommends combining random words or phrases, to create a unique and complex password.

1. Using Two-Factor Authentication:

As said before, an account authenticated by two factors is twice as secure. Usually, the second factor is a verification code sent by email or text message. Two-factor authentication lessens the possibility of a data breach.

1. Having a Password Manager:

The educational institution running our application could implement a password manager to make the work easier for both students and tutors. Not only do password managers safely store difficult-to-remember passwords, but they also can generate complex passwords on their own.

1. Updating Passwords Regularly:

The recommendation from NIST is to request a password update from the user at least every 6 months. This will decrease the chances of unauthorized access, keeping the account secure and safe from a breach. For our application, students and tutors would be asked to change their password every academic term (4 months).

1. Avoiding Personal Information as a Password Component

Any personal information can be very easily obtained by hackers by searching through social media, records, or other profiles of the student or tutor. It can also be extracted during a data breach. For this reason, students and tutors would be advised not to use any hints to personal information in their passwords.

1. Implementing Requirements for Password Complexity:

Having a complex password strengthens account security. The NIST recommendation for this is a mix of numbers, lowercase and uppercase letters, and special characters. This coincides with the author’s previous personal recommendations regarding password complexity.

1. Educating Employees on Password Policies:

It is essential that the employees of a company, or the students of a school, are properly educated on password requirements. This will ensure, to the extent possible, that all guidelines are followed, and decrease chances of password compromise. For our application, both the students and tutors will be required to watch an “account security video”, or a series of them, and then pass a test to demonstrate their knowledge. Failure to pass the test would mean being required to take it again.

1. Implementing Password Expiration Policies:

Users are recommended by NIST to update their passwords after a certain period of time. This will ensure that hackers do not have unlimited time to guess a single password. The time recommended by NIST is a maximum of 60 days. This is the timeframe our application will use, so that the feature keeps the application secure without becoming a nuisance to the user.

1. Monitoring for Password Breaches:

Keeping watch of suspicious activity will help prevent potential attacks on the software. Our application will notify its administrator of any suspicious logins or other actions.

1. Identifying Users Biometrically:

While it has already been mentioned that biometric identification can greatly improve application security, and NIST confidently recommends it, this is a feature that will most likely not be present in our program, as it is probably unnecessary for this type of application.

1. Securely Storing Passwords:

Having a secure password storage measure like encryption will keep them safe from malicious agents. Additionally, salting, key stretching, and hashing can be good protection techniques. Encryption is what our application will go for.

1. Encrypting Passwords in Transit:

Many attacks occur in the timespan when data is being transferred from one point to another. Our application will use HTTPS to encrypt all passwords.

1. Auditing Security of Passwords:

Performing a password security audit can expose flaws and weaknesses in the protection mechanisms, leading to improvements. With auditing, we will be able to improve our program in many of the ways examined before, such as setting complexity requirements and preventing reused passwords.

1. Constantly Updating Password Guidelines:

Our team will make sure to keep all guidelines up to date, just as it is done by NIST. This will ensure that we are following the latest standards and technologies, maximizing our protection of the software.

(Dhole, 2024).

**Testing for Logout Functionality (OTG-SESS-006):**

Logging out of an application safely is just as important as logging into it. Attacks like cross-site forgery can be prevented by shortening the lifetime of session tokens (OWASP Testing Guide Version 4.0, 2011). To ensure a proper session termination and logout procedure, several things are needed:

1. The user must be able to see the logout button on a page clearly, without needing to scroll.
2. The user must be able to logout manually and have control over ending their session.
3. The session must be automatically terminated after a set amount of time, if a user is inactive on the website. This is to prevent attackers from taking advantage of that inactivity period.
4. The session state on the side of the server must be properly invalidated. This means that the application must properly handle the logout from the server.

(OWASP Testing Guide Version 4.0, 2011).

Because ours is a tutoring application, where students would probably not spend all day, our automatic lockout time in case of inactivity would be 15 minutes. The “Login” button would always be at the center of the homepage, while the “Logout” button would be at the top right corner of each page. Both positions would allow students and tutors to see it as soon as they access the app. Except for the automatic session termination, lockout would require manual initiation from the user. The server would them complete the remaining work.

Below is sample code that shows a Java program with a logout timer for a period of inactivity, using Java Swing and Java AWT:

public class logoutSession {

private Timer inactivityPeriodSeconds = 900;

private final int timeoutTimer = 900000;

public sessionTermination (int timeoutSecords) {

this.timeoutTimer = timeoutSeconds \* 1000;

InactivityTimerSetup( );

}

private void inactivityTimer( ) {

ActionListener loggingOut = new ActionListener( ) {

public void actionPerformed(ActionEvent e) {

System.out.println (“Your session has been terminated due to inactivity. Please login again.”);

}

};

inactivityPeriodSeconds = new Timer (timeoutTimer, loggingOut);

inactivityPeriodSeconds.setRepeats(false);

}

public void timerReset( ) {

if (inactivityPeriodSeconds.isRunning ( )) {

inactivityPeriodSeconds.restart ( );

} else {

inactivityPeriodSeconds.start( );

}

public void userIsActive( ) {

timerReset( );

}

public void endTimer( ) {

if ( inactivityPeriodSeconds.isRunning( )) {

inactivityPeriodSeconds.stop( );

}

}

}

In the above code, a timer for session duration is set for 900 seconds, while a timer for duration of the logout is set for 900,000 milliseconds (which both equal to a period of 15 minutes). The timer for the timeout duration is eventually also converted to milliseconds when it is multiplied by 1,000. Then, an action listener is set up to display a message for the user when their session is terminated due to inactivity. The inactivity timer is prevented from running again by the “.setRepeats(false)” method. The code then includes a method to restart the inactivity timer every time a user becomes inactive for a second time, and start it when they first show inactivity. Finally, the code has a method to reset the timer when the user begins to show activity, as well as to end the timer.

References

*11 Password Cracker Tools (Password Hacking Software 2024)*. (n.d.). Software Testing Help. Retrieved April 22, 2024, from <https://www.softwaretestinghelp.com/password-cracker-tools/>

Dhole, S. (2024, April 22). *NIST password guidelines 2024: 15 rules to follow*. TrustCommunity. <https://community.trustcloud.ai/article/nist-password-guidelines-2024-15-rules-to-follow/>

Johnson, K. (2023, October 18). *Use these 6 user authentication types to secure networks*. Security. <https://www.techtarget.com/searchsecurity/tip/Use-these-6-user-authentication-types-to-secure-networks>

Kime, C. (2023, May 17). *How to prevent SQL injection: 5 key prevention methods*. eSecurity Planet. <https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks/>

*OWASP Testing Guide* (4.0). (2011). Open Web Application Security Project.

P., H. (2023, November 20). *Best practices for HTTPS implementation: A comprehensive guide - MindStick*. MindStick-Unleash Your Imagination | Software Product Development Company. <https://www.mindstick.com/articles/334147/best-practices-for-https-implementation-a-comprehensive-guide>

Shankdhar, P. (2020, September 24). *Popular tools for brute-force attacks [updated for 2020]*. Cybersecurity Training & Certifications | Infosec. Retrieved April 22, 2024, from <https://www.infosecinstitute.com/resources/hacking/popular-tools-for-brute-force-attacks/>

*WSTG - Latest*. (n.d.). OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/03-Testing_for_Weak_Lock_Out_Mechanism#Remediation>