ISEC 400 Homework 4 Name: Megan Leonard

Answer the following questions based on your reading of the textbooks, any supplemental material, and the instructor’s presentation this week. If you use an external source (i.e. a web-page, the required textbook, or an additional book) to help you answer the questions then be sure to cite that source. Hint: you should probably always be citing a source.

## Questions

1. **[10 points]** Recalling your previous experience writing web applications (either in WEBD 236 or another similar course), identify and explain (preferably with source code examples) five vulnerabilities in your own code or instructor sample code that correspond to the current weaknesses identified in the OWASP top 10. Also, explain how those weaknesses could have been corrected.

As I do not have the best memory, I was able to find coding examples on line that cover the OWASP top 10. The first code is an SQL injection:

uName = getRequestString(“username”);

uPass = getRequestString(“userpassword”);

sql = ‘SELECT \* FROM Users WHERE Name =”‘ + uName + ‘” AND Pass =”‘ + uPass + ‘”‘

This code lets the person obtain the user information and is vulnerable as the person can use the admin name to get the records that are accessible to admins. A quick fix to this would be to blacklist symbols needed for sql within the user side.

Another example is for authentication failure.

Text

Description automatically generated

The problem with this code is that the token that is created using the user id is saved within the application and available to the frontend. A fix for this would be to specifically save it to the backend using cookies for access, and set up a specific expiration date.

An example of server-side request forgery is as follows:

//Regex validation for a data having a simple format

if(Pattern.matches("[a-zA-Z0-9\\s\\-]{1,50}", userInput)){

//Continue the processing because the input data is valid

}else{

//Stop the processing and reject the request

}

This code uses validation for the request and forgery would be able to match the pattern with their input. A way to solve this would be to set up a domain validator to check the domain and a whitelist of allowed domains.

The next code is for a security misconfiguration that covers a XXE attack.

|  |
| --- |
| <!DOCTYPE Response [  <!ENTITY message SYSTEM "file:///etc/passwd" >  ]>  <Response>&message;</Response> |

This attack means that the system can read the passwd file on the server so the best course for this one would be to disable external entities or change their XML parsers to not be accepting of any custom document definitions.

The final code type is a broken access control.

|  |
| --- |
| pstmt.setString(1, request.getParameter(“acctNo”)); ResultSet results = pstmt.executeQuery( ); |

This lets the person alter the acctNo within the browser so they can send any account number and get the user’s information. Some ways to prevent this would be adding privilege functions so everyone but legitimate users have the permissions or putting in place some RBAC so that the permissions are limited to those who are supposed to have them.

1. **[5 points]** Compare and contrast blacklist and whitelist input validation. Which is more reasonable to implement in web applications and why?

Blacklist and whitelist validation are both lists that an item will be compared with to see if it is contained within the lists. The blacklist will refuse entry for any matching party while whitelist will allow only matching parties. The whitelist would be more reasonable to implement as it limits what is allowed to pass helping block different attacks. The blacklist would specifically stop any attacks that are on its list so you can end up missing some if it is not already on the list. The whitelist is a good means of limiting access to the web application.

1. **[5 points]** The solution for many vulnerabilities is proper encoding of user-supplied data. Compare and contrast early (i.e. as it is stored in the database) vs. late (i.e. as the data is reflected back to the user) encoding of data in web applications. How is this concept related to input validation?

Early and late encoding are both methods of sanitizing data. The early encoding will sanitize the data prior to it entering the database while the late encoding will sanitize the data once we are planning to use it. The concept of encoding the data using the two methods relate to input validation as it can be used to transform the data into a more friendly form for validation. The validation can also be set up to require encoding before accepting into the database for early and being used for late encoding as a means of protection.

1. **[5 points]** Use the Common Vulnerability Scoring System to rate each of the five vulnerabilities you identified in question 1 above (provide a screenshot and explanations for your choices). Perform the same classification using the DREAD approach. Were there any discrepancies in the relative ratings of risk between the two approaches? If so, why?
2. SQL Injection

Cvss= 4.5

Dread= 6

This score is decided by how the injection is used to get the admin records but are not the hardest to fix.

1. Authentication Failure
2. Cvss= 4
3. Dread= 6

Like the injection, this one involves a token being saved to the browser which can cause trouble through sharing of the browser but can be fixed without too much difficulty.

1. Server-side Request Forgery
2. Cvss= 5
3. Dread= 6.2

This is a bit more of a problem as it is made to create requests but could be helped with a domain validator.

1. Security Misconfiguration (XXE Attack)
2. Cvss= 7.5
3. Dread= 8

The security Misconfiguration is the highest as it deals directly with passwords and sensitive information, it also gives access to a file of passwords.

1. Broken Access Control
2. Cvss= 5
3. Dread= 6.5

This is the second highest as we are dealing with the use of account numbers to retrieve sensitive information and need permissions put into place to stop it.

1. **[5 points]** In approximately 300 to 400 of prose (i.e. sentences, not bullet lists) using APA style citations if needed, summarize and interact with the content that was covered this week in class. In your summary, you should highlight the major topics, theories, practices, and knowledge that were covered. Your summary should also interact with the material through personal observations, reflections, and applications to the field of study. In particular, highlight what surprised, enlightened, or otherwise engaged you. Make sure to include at least one thing that you’re still confused about. In other words, you should think and write critically not just about what was presented but also what you have learned through the session. Ask at least one question that your instructor can answer in the returned assignment or class discussion.

This week we looked at the OWASP top ten and how to handle the different threats and vulnerabilities. I remember going over OWASP before and I find it quite interesting as what people consider to be the biggest threats and how they change over the years. Looking at the differences between the OWASP top ten of 2017 and 2021 showed some new one’s bur also seemed to combine some of the previous vulnerabilities like XXE being added under security misconfiguration. The validation was also nice to learn about as so much of what we do needs validation like the blacklist and whitelist. My question for this week is with the OWASP top ten do you agree with the changes they made, or do you think something else should be added or switched?

Citation:

*C5: Validate all inputs¶*. C5: Validate All Inputs - OWASP Proactive Controls documentation. (n.d.). Retrieved February 19, 2023, from <https://owasp-top-10-proactive-controls-2018.readthedocs.io/en/latest/c5-validate-all-inputs.html#:~:text=Even%20though%20blacklisting%20can%20often,and%20potentially%20stop%20obvious%20attacks>.

Chachak, E. (2021, November 2). *Real-world examples for Owasp top 10 vulnerabilities*. CyberDB. Retrieved February 19, 2023, from <https://www.cyberdb.co/real-world-examples-for-owasp-top-10-vulnerabilities/>

diptokarmakar47. (2020, October 12). *The OWASP top 10 – a technical deep-dive into web security*. freeCodeCamp.org. Retrieved February 19, 2023, from <https://www.freecodecamp.org/news/technical-dive-into-owasp/>

*Owasp Broken Access Control Attack and its prevention*. Crashtest Security. (2022, November 21). Retrieved February 19, 2023, from <https://crashtest-security.com/broken-access-control-prevention/>

*Server-side request forgery prevention cheat sheet¶*. Server Side Request Forgery Prevention - OWASP Cheat Sheet Series. (n.d.). Retrieved February 19, 2023, from <https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html#example>

Vicente, G. (2022, May 11). *Prevention of XML external entity (XXE) attacks*. Hdiv Security. Retrieved February 19, 2023, from <https://hdivsecurity.com/bornsecure/prevention-of-xml-external-entity-xxe-attacks/>

Whittaker, T., &amp; Whitney, T. (2022). Week04 Power Point. Retrieved February 19, 2023, from

<https://franklin.instructure.com/courses/11070/files/1516718?module_item_id=623551>