UNIVERSITY of TASMANIA

KIT725 - Cybersecurity and eForensics

Assignment 2

Cybersecurity vulnerability analysis - Group Component

Group Number :25

Group member Name and Student ID:

Name	Student ID
Parth Patel	684138
Ravi Savani	685167
Prince Patel	681355

Table of Contents

1.Summary of identified vulnerabilities:

There were many vulnerabilities in the provided code of TheWeakestBankOfAntarctic, which could lead to financial system inconsistencies, loss of user interests, wrong account Balance, security issues, and even system crashes when not handled properly. We have discussed these weaknesses below:

One major flaw in the TransactionController class was the **repeated updating of the account balance when performing a transaction**. This vulnerability violates an important principle of **ACID** (Atomicity, Consistency, Isolation, and Durability) transactions, specifically Atomicity, which ensures that transactions are all or nothing. The balance was updated BEFORE the success of the transaction serialization was confirmed. However, this created a chance for fiscal inconsistencies if the serialization process crashed after the balance had already been altered, possibly showing an inaccurate representation of the account's state. For example, an account could show a completed transaction in the balance when the transaction wasn't fully processed. This type of error is another type of software vulnerability in which a program writes outside the bounds of an allocated area of memory, potentially leading to a crash or arbitrary code execution **Editorial Staff [4]**.To fix that problem, we ensure that the account balances are not updated until the transaction serialization has been completed successfully. The updated code is placed inside the try-catch block so that the account balances are only adjusted if the transaction is serialized successfully. This ensures that the balance is truly representative of the state of the transaction, that is, that the system never records partial transactions, and thus preserves the integrity of the data as well as the atomicity, and consistency of the transaction, significantly reducing the chance of misrepresentation of account states.

The other major flaw was the **improper withdrawal system**. This type of vulnerability is a dangerous software vulnerability that occurs when a system does not safely handle input data **Martin, R. A. (2012). [3]**. No validation was in place to verify that the withdrawal amount was greater than zero and enough balance to create a successful transaction. The money withdrawal may be unauthorized and the account balance might become negative or go above available cash. Such operational lapses like unchecked withdrawal processes and unauthorized access to funds can affect banking operations' stability and serious financial risks. Validation checks were added to resolve this. Now after updating the code, it checks if the amount to be withdrawn is more than

zero (positive) and whether the account has sufficient balance to withdraw. If these conditions are not met an exception is thrown, which makes the transaction halt. Unauthorized or erroneous withdrawals will be avoided, account balances will be maintained, and can always follow the financial principle. It would thereby strengthen security and usability. Future developments could be linked with fraud detection systems that pinpoint suspicious withdrawal patterns, either by user behavior or patterns outside of the norm.

Another critical flaw was the **lack of a rollback strategy** if anything went wrong during the transaction. **Braz et al. (2021) [2]** in his study showed that the software that does not check the return value from a function, can prevent them from detecting unexpected states and conditions. If the serialization fails after updating account balances, then those changes will remain and the money will not be reverted which will cause incorrect account balance data. This caused major problems for transfer operations, as the sender and receiver's balance had to be debited correctly. A rollback mechanism was introduced to solve this problem. If the transaction serialization is unsuccessful, then it is ensured that the balance is reverted to the sender's account. For instance, in a transfer operation if serialization fails after debiting from the sender, it is ensured to credit back the deducted amount of the sender, and no changes are made to the receiver's account balance. This error handling ensures that when the failures happen, things are kept consistent in your application and you can have a guaranteed state of these accounts. It makes sure that there are no financial miscalculations that lead to frustrated users. This mechanism of error handling ensures the data remains invariant and consistent during any transactional activities, matching up to all standards associated with transactional integrity in this industry. In addition, the introduced alteration would enable the extension of the rollback mechanism to probably solve more complex transactions such as multi-party transfer, where the system may demand an advanced rollback in future implementations.

There were many weaknesses in the CloseAccount function and lacked a lot of attention. However, there was a potentially **null reference exception** that could have arisen if the account in question could not be located when the method was invoked. If there is no account with the specified number, the account variable would be null which would cause a NullReferenceException to occur when trying to read Account Number. Such a vulnerability could cause a runtime error and eventually, the application would crash as a result of a faulty experience to the users trying to close

accounts that don’t even exist. Such failures could be disastrous in the sense that there would be instability and risk of unsaved data being lost due to the presence of such a serious bug in the functionality. Most null pointer issues result in general software reliability problems **The OWASP® Foundation. (n.d.)[8]**. Null references are the main bugs in the software system that cause instability and crashes or unhandled exceptions that degrade user experience.

The practical implication of this problem was implemented that if the account that is currently being searched is missing or doesn't exist, then the system will display a statement “account does not exist”. This improvement thus allows the application to handle these bad practices more effectively and not crash but display the message “account was not found”. The account closure feature was therefore improved, and the stability of the software increased by reducing its unexpected runtime errors turned out to be quite satisfying for the users. The other enhancement is that the scalability implications are improved, as now the code can gracefully handle a broader range of error scenarios, thus having an overall improving effect on system resiliency. This could be carried out even further by logging the events so that developers can track issues and anomalies in real-time.

Transaction handling flaws in the XMLAdapter class were also found that were related to transaction handling and serialization errors, in addition to the null reference problem. In particular, there was no way to reverse partial or corrupted transaction records in the event of a transaction serialization fault (either by disk space problems or access rights). This constituted a serious risk to the system for maintaining data consistency, especially in the area of applications related to finance, where, an accounting transaction history is required to be precise A solution incorporating transactional file writes was put into place to lessen this risk. The system can avoid possibly incorrect data overwriting in existing records by writing transactions to a temporary file first and replacing the main file only after successful serialization. This minimizes the possibility of partial or failed transactions from polluting other records that already exist, thus enhancing the dependability of the transaction processing system considerably and also protecting against data corruption. Moreover, it provides some degree of fault tolerance to the system and can even extend this to include further data validation checks to protect against serialization errors even more.

Additionally, during **transaction deserialization, inefficiencies in file handling** were addressed. When working with huge datasets, the old method might be slow and inefficient because it reads

each XML file separately. It processed files one at a time; this was slow and also memory-intensive. Such inefficiencies caused in any environment have high volumes of transaction delays and system bottlenecks. Batch processing was added to increase performance, enabling the code to read and process files in groups. According to **Martin et al. [5]** when a resource performs an activity on several cases simultaneously, sequentially, or concurrently, this is referred to as batch processing and it is less time-consuming. This modification increases efficiency and lowers memory strain, especially in situations with a large number of transaction files. Furthermore, although project restrictions at first caused the code to have problems with asynchronous file operations, a synchronous batch processing strategy was used. By reducing the possibility of performance snags during file I/O operations, this tweak made sure the system stayed responsive even when processing bigger transaction datasets. All things considered, these improvements greatly enhanced the dependability, effectiveness, and user experience. Furthermore, a future improvement could involve the use of asynchronous file processing or parallel I/O operations to further optimize performance in real-time transaction environments.

Hard-coded credentials were a serious security concern because the original implementation in AccessController class kept passwords and usernames directly in the code. The use of a hard-coded password increases the possibility of password guessing tremendously **The OWASP® Foundation. (n.d.)[9]**. Unauthorized access can result from the extraction of hardcoded credentials. It is against recommended security practices to store sensitive data in the source code, as this makes it simpler for hackers to access the data and jeopardizes the integrity of the system. To prevent sensitive data from being directly exposed in the source code, the credentials were moved to environment variables (APP_USERNAME and APP_PASSWORD) improving security through the instance of best practice secure credential management. This reduces the possibility of unwanted access and adheres to the secure credential storage principle. Future work will involve expanding this system into implementing encrypted credential storage mechanisms like integrating into AWS Secrets Manager or HashiCorp Vault.

Lack of Account persistence was another main issue. When the program exited, the accounts established by the CreateNewSavingsAccount and CreateNewCheckingAccount methods were gone since they were not saved or persistent in a file or database. The users would have a bad user experience since they would have to create their accounts again if account persistence was not

implemented. Any account created during runtime would be lost after the application ended. Despite the data adapter's lack of a SaveAccount method, a placeholder comment was included to indicate the location of the persistence logic that may be implemented later. This update foresees the requirement for persistence to guarantee that account data is preserved and not deleted. This is crucial for long-term scalability, and future versions can consider database integration for persistent accounts and transactions so that the data can be continued between sessions of the application.

There was a **lack of Clarity in the Return Value** in the AccountClosure class. The CloseAccount function has been debated to provide a return value of true irrespective of the account status which made the calling function work under the wrong assumption of a successful operation. It may affect the functioning of the application in other areas where it will be incorrectly anticipated that the account has indeed been closed. Such issues have been removed by modifying the method to make a call to CloseAccount, which will either return true or false, depending on whether the closing operation was indeed a success or not. This correction lets the system provide more accurate information in its feedback but protects subsequent errors from proliferating into the application logic. Further improvements can be made about error-reporting details with actual reasons for failure, like account not found or pending transactions returned to the end-users for better clarity.

Inefficient Searching was another potential issue that was due to Poor Implementation of the Search Method UserController class was also a Weakness. In performing this process, the SearchByAccount method employed a linear search in conjunction with the Contains method on a List of customers with a complexity time of O (n ∗ m) where n is the number of customers and m is the number of account owners. This inefficient search mechanism as the size of the dataset increases will lead to slow response times or delays as the user would have to wait longer than anticipated. This has more adverse effects when large datasets are being dealt with. The search mechanism was changed from List to HashSet which offers an average time of O (1) to look through the fills **Saeed Anabtawi. (2023, June 13)[7]**. This would improve performance for large datasets. Future work might include indexing to make searches even faster and to accommodate queries on multiple attributes like transaction history or account type.

The already identified vulnerabilities that were addressed in the system have improved the stability, security, and performance of the application. In addition to eliminating the risk of serious threats

in these core functionalities, this system enhanced other safety nets such as transaction rollbacks and error handling. Improving the search algorithm through file processing and refactoring the whole system while moving sensitive data like credentials into environment variables has reduced security risks to near-nothingness.

Further development may be the implementation of more elaborate logging or error-catching systems that allow real-time tracking of both errors and system performance. Additional implementation of automated testing, especially where it counts most transaction processing and the close-out of accounts will also help ensure the dependability of the system in subsequent generations of development.

All these improvements created a good groundwork for any future growth and scaling, so the system is better at handling larger data sets, more users, and more complex transaction scenarios.

2. Scope of work:

Parth Patel was able to point out key weaknesses related to security and performance issues. He pointed out that the AccessController class held hardcoded credentials thus increasing the system to be burdensome in terms of maintenance since sensitive information can be leaked. Also, he identified a missing Account persistence feature in the AccountController class, thus there is a high probability of losing data whenever the application is stopped. He pointed out missing authorization checks and inefficient search implementation that may lead to logical errors and slow up the performance.

Ravi Savani, in his report, focused on the vulnerabilities of the DataAdapter and XMLAdapter classes. He discovered that when trying to get accounts, there may be a possible null reference exception which likely causes an application crash. The XMLAdapter class lacks rollbacks on failed transactions as well as inefficient operations over files which in one way may be a blow to performance systems. His analysis proved to be very great in helping improve the reliability and efficiency of the system.

Prince Patel outlined three important vulnerabilities that they found in the TransactionController class: inappropriate handling of withdrawal, no guarantee of managing errors in a transaction, and updating balances redundantly. All of this would give a system a tendency to result in negative

balances or wrongly accounted information in the accounts. A fantastic investigation like that led to important findings concerning issues that could affect the system in terms of the integrity of financial operation

3. Presentation of the scope of work:

Group Members	Work Division
Prince Patel	1. Improper Withdraw. 2. Incorrect Transaction Handling. 3. Redundant Account Balance Update.
Ravi Savani	1. Potential Null Reference Exception. 2. Lack of transaction rollback mechanism. 3. Inefficient file handling for transactions. 4. Not leveraging async/await for file IO operations.
Parth Patel	1. Hardcoded Credentials. 2. Lack of Account Persistence. 3. Missing Authorization. 4. Inefficient Search Implementation.

4. Trend Analysis:

(Note that all the recurring weaknesses are highlighted in red color.)

Sam Ransbotham et al. [6] in their study stated many common software vulnerabilities, especially in open-source projects which are listed as input validation errors; design flaws; access control weaknesses, and configuration issues all captured within their study. Vulnerabilities in open-source systems are more likely to be exploited as attackers can see those vulnerabilities and easily exploit them. One of the main reasons such vulnerabilities are recurrent is that despite many eyes on open-source code, not all issues are detected before release leading to unpatched legacy systems. The availability of source code makes it easier and less time-consuming for attackers to come up with a good set of exploit tools. In the future, such vulnerabilities might result in more

extensive exploitation attempts that are also faster because the use of automatic tools become a norm for attackers. These weaknesses may persist longer if no remedial action is taken, thus posing increased threats to the software system and bringing the need for more robust security practices along with fast patching to mitigate the possible damages.

The research by **Taleby Ahvanooey et al. [1]** presents several key vulnerabilities, especially within Android and iOS-based smartphone operating systems (OS). Careful analysis of these incidents reveals that the same vulnerabilities are targeted over and over again, typically revolving around Malware infections, App permissions, social engineering, and Backdoor exploits. The prevalence of these vulnerabilities is due to Android being open-source where anyone can develop apps, including malicious ones. Additionally, impulse downloads from untrustworthy third-party markets by unaware users highlight a security risk at the same time. According to the research, there is a good chance that these vulnerabilities will increase in frequency as more people use smartphones and malware becomes increasingly sophisticated. As a result, this trend may lead to larger data breaches, increased unauthorized access or infiltration against smartphones with sensitive information, and higher rates of cyber-attacks. The implications for the future highlight the necessity to enhance security and educate users as well as implement improved malware detection systems on mobile devices.

This research by **Peng Zeng et al. [10]** explores the weaknesses that involve the disclosure of information, with a focus on vulnerabilities targeting the ability to detect specific threats employing deep learning processes. It has been established that repeated vulnerabilities in software systems typically include: buffer overflow; memory errors; and code dependencies. These problems are usually caused by the complex nature of current software where the developers can't account for every possible security breach and traditional approaches to feature extraction in machine learning can create problems themselves since they are subjective. Recurring vulnerability reasons also include the inability to capture the deep semantic and contextual relationships in the code as well as the constraints that are imposed by the existing detection methods. Automated feature extraction has been enabled by the use of deep learning models such as LSTMs, and CNNs which helped in the enhancement of detection accuracy.

However, the prospects should raise concern: because of the anticipated escalation in complexity for future software systems, vulnerabilities may be difficult to identify without relevant advanced

measures such as employing a deep learning-based model. There is expected to be increased reliance upon automated vulnerability detection measures, however, issues of data scarcity, the problem of 'transparency' in models, model interpretability, and fine-grained detection granularity must be addressed. If these challenges are not resolved, there shall remain a threat to software systems that are already at risk of exploitation.

5. REFERENCES:

(Note that All the references are highlighted in yellow color.)

[1] Ahvanooey, M. T., Li, Q., Rabbani, M., & Rajput, A. R. (2017). A Survey on Smartphones Security: Software Vulnerabilities, Malware, and Attacks. In *IJACSA) International Journal of Advanced Computer Science and Applications* (Vol. 8, Issue 10). www.ijacsa.thesai.org.

[2] Braz, L., Fregnan, E., Calikli, G., & Bacchelli, A. (2021). Why don't developers detect improper input validation? '; DROP TABLE Papers; -. *Proceedings - International Conference on Software Engineering*, 499–511. https://doi.org/10.1109/ICSE43902.2021.00054.

[3] Christey, S. M., Kenderdine, J. E., Mazella, J. M., Miles, B., & Martin, R. A. (2012). *CWE Version 2.2*.

[4] Editorial Staff. (2024, June 1). *What Is An Out-of-Bounds Read and Out-of-Bounds Write Error?* Https://Securityboulevard.Com/2022/06/What-Is-an-out-of-Bounds-Read-and-out-of-Bounds-Write-Error/.

[5] Martin, N., Swennen, M., & Jans, M. J. (2015). *Batch processing: definition and event log identification*. https://www.researchgate.net/publication/287198292.

[6] Ransbotham, S. (2010). *WORKSHOP ON THE ECONOMICS OF INFORMATION SECURITY An Empirical Analysis of Exploitation Attempts based on Vulnerabilities in Open Source Software*.

[7] Saeed Anabtawi. (2023, June 13). *Understanding HashSet, LinkedHashSet, and TreeSet in Java*. Https://Www.Linkedin.Com/Pulse/Understanding-Hashset-Linkedhashset-Treeset-Java-Saeed-Anabtawi-/.

[8] The OWASP® Foundation. (n.d.). *A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.* Https://Owasp.Org/Www-Community/Vulnerabilities/Null_Dereference.

[9] The OWASP® Foundation. (n.d.). *Use of hard-coded password*. Https://Owasp.Org/Www-Community/Vulnerabilities/Use_of_hard-Coded_password.

[10] Zeng, P., Lin, G., Pan, L., Tai, Y., & Zhang, J. (2020). Software vulnerability analysis and discovery using deep learning techniques: A survey. In *IEEE Access* (Vol. 8, pp. 197158–197172). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2020.3034766.