**Vulnerabilities Phase 2**

Peter Colburn - 30017907

Thomas O’Brien - 30021359

Kavell Long - 30004171

Billy Hodges – 30021362

Contents

[Injection (SQL) – Peter 1](#_Toc96631380)

[Insecure design – Tom 2](#_Toc96631381)

[Broken Access Control – Kavell 2](#_Toc96631382)

[Identification and Authentication Failures – Kavell 3](#_Toc96631383)

[Vulnerable and Outdated Components – Tom 3](#_Toc96631384)

[Memory Safety – Peter 4](#_Toc96631385)

[Privilege-confusion – Billy 4](#_Toc96631386)

[Our choices – Kavell 5](#_Toc96631387)

[References 5](#_Toc96631388)

# Injection (SQL) – Peter

SQL injections are one of the most serious vulnerabilities for applications as said by Halfond, Viegas and Orso (2006, p. 1). They cause vulnerabilities due to hackers having the ability to gain complete access to things like sensitive data like the information of customers and consumers and they also allow attackers to access any kind of database. Because databases hold unbelievably valuable information, they are a big target for attackers and that is why it is so vulnerable, and it should not be overlooked when using secure software. SQL Injections can also be used for hackers to gain control over a system or application causing lots of damage. This obviously should be prevented at all costs as it comes under the legal acts. So how do SQL attacks work and why are they so dangerous? So, SQL injections are a class of code-injection I am which data provided by the user is included in an SQL query in such a way that part of the users input is treated as SQL code. So, by doing this an attacker can start to enter SQL commands into the database meaning that they can receive any data they want from the system as said by Halfond, Viegas and Orso (2006, p. 1).

How are SQL injections prevented then? SQL injections can be easily prevented by the simplest way which is to ensure to use defensive coding practices. The main cause for SQL injections is due to insufficient input validation. One way of fixing this is by using input type checking. This is where inputs into SQL can be checked, for example the developer can reject any input that contains characters other than digits. Another thing that can be used is Identification of all input sources. This means that every time something is introduced with SQL it must be checked before it can be inputted as said by Halfond, Viegas and Orso (2006, p. 6) This is the safest way to prevent SQL injections, but it is also not very efficient. Lastly, there is a way that testers can prevent SQL injections before the database is even released. This is done by Black Box Testing. This is where a web crawler is introduced to try and find any vulnerable points so that these problems can be fixed before it is released and starts holding data as said by Halfond, Viegas and Orso (2006, p. 6).

# Insecure design – Tom

According to OWASP (2021) insecure design includes: Error messages containing sensitive information, unprotected storage of credentials, trust boundary violations, and insufficiently protected credentials. These tell us some of the common pitfalls we must avoid as a group. For example, this research has brought upon other avenues of research such as password encryption. According to OWASP (2021)- “Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods.” This means that secure design is not just about individual issues in a code but instead is the “culture and methodology” that we use to design it. This means in our project we should adopt this methodology to ensure that our project is protected from the pitfalls of insecure design.

As said by Janusz Kulon (2022) “You can test for the presence of bugs in your code, but you cannot test for the absence of bugs in your code” this further backs up the importance of secure design principles as we need to avoid implementing ‘spaghetti’ code and other poorly implemented designs. This also means our program should undergo regular and thorough testing to eradicate as much faulty code as possible and to find as many common issues as possible while we can. It is also important to consider a support model after the release of our project to the client and the support network we can implement to allow fixing of bugs that may be discovered by them. This could include the implementation of error reports that could be sent to us to help us find the issues easier.

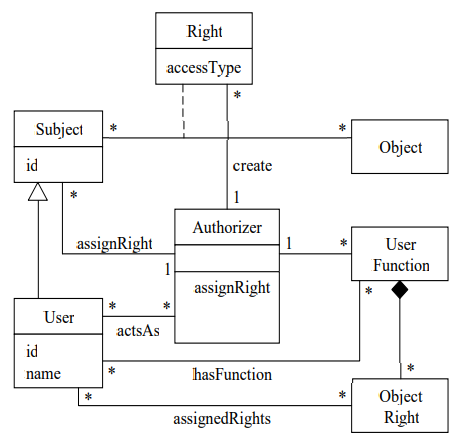
# Broken Access Control – Kavell

Broken access control is a policy where users have certain areas of access within an application. For example, in a retail setting a manager may have full control over a payroll system – adding/amending employees etc., but an associate will only be able to view their own payslip or holidays. If the associate were to be given the same control as the manager, they may misuse this privilege and give themselves a higher wage or more holiday entitlement.

In the OWASP the Top 10 Web Application Security Risks (2021), broken access control was the one that came up more than any other category. It was in the 2017 list as well but lower down the list. Out of all the applications that were tested, 94% of them showed broken access control. There are a few ways in which broken access control can be done from the user. Some examples of this are Cross Site Scripting, Cross Site Request Forgery and Local File Inclusion. According to Hassan et al. (2018), these are often the result of the implantation not being done properly when inputting user authentication among other reasons.

OWASP (2021) explains the ways to prevent these things happening include using access control mechanisms more than once within an application, disabling the web server directory listing and ensuring backup files are not within the web roots.

Another aspect which must be considered is the “principle of least privilege.” This is discipline built from integrity, availability and confidentiality which are objectives for information security programmes. This principle means that a user should only be given the bare minimum of access needed to do their job on a need-to-know basis. Fernandez, Mujica and Valenzuela (2011, p. 2) say in their paper “Two security patterns: Least Privilege and Security Logger and Auditor” how to code specifically to stop the misuse of the application by users. They go on to suggest a security pattern called “Need-to-Know Authorizer” – this makes “Rights” which are then assigned to “Subjects” which will have determined sets of access levels. Below is a screenshot of the UML diagram of their pattern.



# Identification and Authentication Failures – Kavell

Identification and authentication failures are where an unauthorised user gains access to valid usernames and passwords through an attack or the company uses weak commonly used usernames and passwords – which could be easily guessed, e.g., admin/admin or admin/password1 etc. according to OWASP (2021). These failures moved from second place in 2017 to seventh place in 2021 in the Top 10 Web Application Security Risks from OWASP.

OWASP (2021) then goes onto say how these vulnerabilities could be helped. The first thing which could be implemented (where possible) is a multi-factor authentication – this will help with stolen username and passwords as well as attacks forcing their way in. Next would be checking if a password is weak by comparing it to the top 10,000 worst passwords list. These passwords should then be made purposely long and complicated to prevent guessing.

# Vulnerable and Outdated Components – Tom

When we design and implement our program, we must consider that the components that we use are up to date, but why is this important? Vaniea Kami, (2014) “Betrayed by Updates: How Negative Experiences Affect Future Security” tells us that due to frustration from updating many users will choose to ignore safety updates, this is a problem as many updates to software or devices are patches to new security risks. For example, after a company learns about a vulnerability to their system, they can then send out an update to increase security. Users who therefore choose to ignore updates expose themselves to these security risks. We can also apply this to how we develop our program. The first step we may take to reduce this is by ensuring libraries we use while coding is the most recent and up to date ones so that we have all the latest security features and updates. We should also consider retroactively updating our program if a security patch or update is released for any libraries or software, we are using to ensure we are gaining the full benefit and security that comes with it. One of the key reasons why it is so critical to ensure we use update software is that while it is possible to test for presence of bugs it is impossible to test if there are none present at all. This means often that new bugs are picked up by users and new security weaknesses are discovered which these software updates can prevent.

# Memory Safety – Peter

Memory Safety is the state of being protected from various software bugs and security vulnerabilities when dealing with memory access. The two vulnerabilities that will be stated will be buffer overflows and dangling pointers.

Starting with dangling pointers they are pointers that are left pointing to deallocated memory after the object they used to point to has been freed. This can cause vulnerabilities because attackers may manipulate these pointers by the program assuming the pointer is operating on an object of the type formerly occupied by memory but will operate on any memory being occupied at that given time as said by Akritidis (2010, p. 1). These are considered just as dangerous as buffer overflows, but they are more easily prevented. Dangling pointers can be prevented very easily by just initialising the pointer to “NULL” after deallocating memory. This means that the pointer is no longer pointing to that memory location and therefore is no longer a dangling pointer as said by Akritidis (2010, p. 2).

Moving on to Buffer Overflow attacks. These attacks are the most common memory safety threat that comes up. Like dangling pointers, buffer overflow attacks are committed due to pointers being used inefficiently. An example of how the buffer overflow works is it will overwrite a buffer on the stack to replace the return address as said by Larochelle, Evans (2001, p. 1). When the function eventually returns instead of jumping to the return address control will jump to the address that was placed on the stack by the attacker. This then allows the attacker to execute arbitrary code which gifts the ability to run any commands or code of choice on a target machine or in a target process. One way of defending against such attacks is by using a compiler that generates special values on the stack next to the return address and this then checks that it has not been tampered with before jumping. Another way of preventing these attacks is by using checks and detections before there is deployment of code.

# Privilege-confusion – Billy

Privileges in a operating system is were an application is isolated to certain aspects of the operating system and can only have access to certain features. To Acquire extra capabilities, an application needs to specifically declare the permissions it requires. Per, Towards Taming Privilege-Escalation Attacks on Android (2012), A Confused Deputy attack is a type of privilege escalation attack. Confused Deputy attacks happen when a malicious application exploits vulnerable interfaces of another privileged application.  This can be abused in our software and can lead to catastrophic consequences as loss of data or stealing of data from a malicious third party, which would obviously be catastrophic in our case as our software holds sensitive personal information about employees.

To Stop this happening with our software we need to implement the right security measures to try and counteract the possibility of privilege confusion attacks. One obvious way too stop privilege attacks is to make sure that our program has the right permissions. This requires us to have a in depth look at what exactly our program does and what permissions does it need to do these tasks. As having a program with permissions that it does not need can leader to greater risks as more damage can be done with higher level permissions as it allows the application more access to the system.

# Our choices – Kavell

After considering our research we have decided to implement broken access control and identification and authentication failures, this is because they are the most effective to put within our code. These will be fully implemented within phase 3.

Broken access control will be implemented by having different levels of access, one being an admin and another being an associate. The admin will have full access and editing control of the whole application and the associate will only have viewing privileges. Both accounts will be password protected.

Identification and authentication failures will be implemented by using harder to guess and unique passwords. According to Security.org (2022), passwords should be more than 16 characters long, a combination of numbers, characters and letters, no personal information, and not “password”.

# References

Akritidis, P. (2010) Cling: A Memory Allocator to Mitigate Dangling Pointers. Pp. 1-16. Available at: usenix.org/legacy/event/sec10/tech/full\_papers/Akritidis.pdf (Accessed: 11/02/2022).

Fernandez, E.B., Mujica, S. and Valenzuela, F., (2011). ‘*Two security patterns: least privilege and security logger and auditor.’* pp. 1-6.

Hassan, M., Ali, M., Bhuiyan, T., Sharif, M. and Biswas, S. (2018) *Quantitative Assessment on Broken Access Control Vulnerability in Web Applications.* Available at: <https://www.researchgate.net/profile/Saikat-Biswas-10/publication/328956656_Quantitative_Assessment_on_Broken_Access_Control_Vulnerability_in_Web_Applications/links/5bed323d299bf1124fd38e46/Quantitative-Assessment-on-Broken-Access-Control-Vulnerability-in-Web-Applications.pdf> (Accessed: 11/02/2022).

Halfond, G. J. W., Viegas, J. and Orso, A. (2006) A Classification of SQL Injection Attacks and Countermeasures. Pp. 1-11. Available at: <https://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf> (Accessed: 10th February 2022).

Kulon, J. (2022) Lecture quote, 17/02/2022.

Larochelle D. and Evans, D. (2001) Statically Detecting Likely Buffer Overflow Vulnerabilities. Pp. 1. Available at: <https://www.usenix.org/legacy/events/sec01/full_papers/larochelle/larochelle_html/> (Accessed: 11/02/2022).

Security (2022) *How Secure Is My Password?.* Available at: <https://www.security.org/how-secure-is-my-password/> (Accessed: 11/02/2022).

OWASP (2021) *A04:2021 – Insecure Design.* Available at: <https://owasp.org/Top10/A04_2021-Insecure_Design/> (Accessed 17/02/2022).

OWASP (2021) *OWASP Top Ten.* Available at: <https://owasp.org/www-project-top-ten/> (Accessed: 11/02/2022).

Towards Taming Privilege-Escalation Attacks on Android (2012), Available at: <https://pdfs.semanticscholar.org/350d/9814ed771b1dd418190b7cd612c487730a91.pdf> [Accessed 17 February 2022].