

Name: *Benjamin J. Clemas*
EID: *CLEBJ002*
SID: *110348297*

Word Count: *3456*
Due Date: *30/10/2023*

Guardian Store Vulnerability Report

Table of Contents

Table of Figures	2
Executive Summary	3
Overview	5
Introduction	5
Vulnerability Classes Identified	5
Chosen Vulnerability: Broken Authorization/Broken Access Control: Password Strength ...	5
Introduction of Vulnerability Class and Chosen Vulnerability	5
Vulnerability Instance	5
Research	6
Reconnaissance	7
Exploitation	8
Patch	9
Security Opportunities	11
Defence-In-Depth	11
Definition	11
Opportunities	11
Secure Software Development Mitigations	12
Definition	12
Opportunities	12
Automated Password Strength Testing	13
Conclusion	13
Bibliography & References	14

Table of Figures

Figure 1	6
Figure 2	8
Figure 3	9
Figure 4	10
Figure 5	10
Figure 6	11
Figure 7	13

Executive Summary

Guardian has contacted us to penetration test GuardianStore, by identifying vulnerabilities and attempting to resolve them. For this report we will be focusing on the Broken Authentication/Broken Access Controls (BA/BAC) vulnerability class and the chosen vulnerability within said class, Password Strength. BA/BAC is a major vulnerability class which could compromise the confidentiality, integrity and accessibility of clients and admins alike.

We will be focusing mainly on its current weakness to brute force attacks. Being a surprisingly common vulnerability, finding a fix for this issue will mean similar cases and issues can also be resolved.

To exploit this vulnerability, we'll be using a brute force payload sniping attack. By combing a list of common admin passwords found online ([8] Wilson, T. 2007), and user passwords ([9] Annie. 2022), we made a small trial password payload of 60 items.

Quick enough to run and with some variations easily expandable to 72 items.

During this time we also found a patch for the password strength vulnerability, which was efficient. only requiring the RegularExpressions ([10] RegExp. 2023) module and it's built-in test function. Which, when passed a pattern will test a sting to find if it matches that pattern. After finding a sufficiently strong password strength pattern and modifying it to be more all inclusive we had found a potential fix.

To test this fix and exploit the vulnerability we first performed reconnaissance, by using the Confidential Information Leak vulnerability, found in the reviews of products, we found many accounts. Including but not limited to: jim@guardian.com, bender@guardian.com, uvogin@guardian.com, stan@guardian.com, bjoern@owasp.org, morty@guardian.com, mc_safesearch@guardian.com, and accountant@guardian.com. As well as the admin account admin@guardian.com, which would be key target, and highlight the potential damages of a Password Strength Vulnerability Exploitation. During this process we also found out that Jim's account belonged to the fictional character Captain Kirk who's answer to his security question was 'Samuel' the middle name of his brother. This would be used to test the reset patch.

Using the password payload designed earlier, and a brute force payload sniping attack on the admin accounts password, we found that the password 'admin123' was correct. This highlights the current insecurity and need for a patch, as within a few hours a third-party could access the admins account. Allowing them to locking us out of the account, as well as create false products, reviews, and otherwise compromise the security, confidentiality, integrity, and availability, of the Guardian Store System.

To fix this exploit we found the createPassword and resetPassword function and added a function and if conditions which would fail the password if it wasn't strong enough, using the RegEx string test designed earlier. This ensured that the only passwords to make it though would have at least; 1 upper and 1 lower case letter, 1 number, 1 symbol and be a minimum of 5 characters long. After some testing the fix was complete and a weak admin password like 'admin123' would never exist again. We chose not to force a site wide password reset, as being forced to change their password can cause users to use less secure passwords like Ca\$h1. To automatically check for this vulnerability we implemented a login strengthFailedLog to log all the emails with weak passwords, so they could be notified and recommend to change their passwords in the future. We would manually change administrative passwords to adhere to this new strength test. Using this method, we can

slowly phase out the weak password, and make them stronger. Attacks in the future would use a payload consisting of upper, lower, number and symbol passwords. In the future multi-factor authentication should be added to the reset password and change password sections and potentially the login page as well to provide defence in depth and protect our system.

By implementing the Password Strength Patch and Automated Password Strength Testing, we can start to fix the broken authentication issues present on Guardian Store and patch out the Password Strength Vulnerability.

Through making use of opportunities presented to us by Defence-In-Depth methods; Implementing Multi-factor Authentication, Backend Encryption, Internet Security Awareness and Physical Defences. As well as Secure Software Development Mitigation Techniques; Least Privileges privilege decompression, Least Common Mechanism reducing commonalities, Common Mediation again with Multi-Factor Authentication, Separation of Privilege with decompression and a task system, and finally Physiological Acceptability, making our system usable and preventing admins creating or using backdoors. We can further defend our system from future attacks and make guardian store the safest shop on the web.

Overview

Introduction

Guardian has contacted us to help penetration test GuardianStore, identify vulnerabilities and attempt to resolve them. For this report we will be focusing on a single vulnerability class and a chosen vulnerability within said class.

Vulnerability Classes Identified

The following vulnerabilities classes have been identified:

- | | |
|--|---------------------------|
| x Broken Authentication/Broken Access Controls | x Cross-site scripting |
| x Broken authorization | x Cryptography issues |
| x Business logic flaws | x Insecure file upload |
| x Confidential Information Leaks | x NoSQL injection |
| x Cross-site request forgery | x Sensitive data exposure |
| | x SQL injection |

Chosen Vulnerability: Broken Authorization/Broken Access Control: Password Strength

Introduction of Vulnerability Class and Chosen Vulnerability

The chosen vulnerability class for this penetration report is Broken Authorization, also known as Broken Access Control. This is a major vulnerability class which could compromise the account security and confidentiality of clients and admins alike. Potentially leading to a data breach, or a third party using administrative privileges to compromise the availability, integrity or confidentiality of the site.

The chosen vulnerability is password strength and current weakness to brute force attacks. Being a surprisingly common vulnerability, finding a fix for this issue will mean similar cases and issues can also be resolved.

Vulnerability Instance

This vulnerability exists because no password strength validation is present, only ensuring that a password is present, and it matches the repeated password.

The servers main method, Sever.ts handles account creation and directly pushes the users new account information to the database, leaving no room for validation. This framework is problematic as, not only does it leave the database vulnerable to exploitation though unvalidated information, but it also prevents validation. Decompressing this into a validated createAccount method, would fix this, but is out of scope for this penetration test and report.

```
routes > TS changePassword.ts > changePassword > <function>
18 module.exports = function changePassword () {
19   return ({ query, headers, connection }: Request, res: Response, next: NextFunction) => {
20     const currentPassword = query.current
21     const newPassword = query.new
22     const newPasswordInString = newPassword?.toString()
23     const repeatPassword = query.repeat
24     if (!newPassword || newPassword === 'undefined') {
25       res.status(401).send(res.__('Password cannot be empty.'))
26     } else if (newPassword !== repeatPassword) {
27       res.status(401).send(res.__('New and repeated password do not match.'))
28     } else {
```

Research

To develop a password strength patch, we must first exploit the vulnerability. For demonstrating the weakness of the vulnerability, and its potential ramifications, we will focus breaching an admin account.

For the creation of a password payload to perform a brute force sniper attack with, we first gathered the most common admin passwords ([8] Wilson, T. 2007), and user passwords ([9] Annie. 2022). This made a trial password payload of 60 items. With some variations, easily expandable to 72 items while still being quick to trial. Due to the insecurity of this admin password, and the weak password strength requirements, it was all we needed to gain access. This highlights the extent of this flaw and the potential damage it can do in a short time.

To patch this, we contemplated using a for loop and string enumeration to pull apart the string and check the password strength, however, we decided it would cause too much bloat and may slow execution. After some research, we found the RegularExpressions ([10] RegExp. 2023) module which allowed you to match any string with a pattern which items could be tested against. This was also a efficient module and wouldn't cause any slow down in execution. After some testing we found a built-in RegEx test function after module installation which we could just feed a pattern string to. Adding this to the else if string throwing errors with password creation and not passing it until conditions where met, made this the perfect fix.

```
^((?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])
(?=.*[-$*.{}?!"'@#%&^_><:;|_~^]\(\)\{\}]{5,})$
```

Regex pattern. Green is pattern formatting, Yellow sets requirements, first ensures it has a lowercase, second an uppercase, and third is a list of accepted symbols. Cyan states the minimum length of the string. Passwords are tested using this pattern through the regex.test() function to test its strength.

Test	123456a
Root	iloveyou
root	111111
test	7777777
1	112233
12	123123123
123	000000
1234	222222
12345	666666
123456	1q2w3e4r5t6y
1234567	987654321
12345678	admin
123456789	admin1
password	admin12
password1	admin123
Password	Admin
Password1	Admin1
Password	Admin12
passwr	Admin123
PassWrd	dragon
qwerty	zxcvbnm
qwertyuiop	monkey
Asdfghjkl	123abc
asdfghjkl	123654
1234qwer	111111
123321	123qwe
cash1	123Qwe
Ca\$h1	121212
Monkey	159753
Dragon	qazwsx
Zxcvbnm	qwe123
asdasd	Qwe123
Asdasd	QazWsx
654321	Qazwsx
aa12345678	Aa12345678

Figure 1. password payload, madeup from 10 of the most common admin and 50 of the most common user password. Aswell as 22 variations on those passwords.

Reconnaissance

To break the authorization and login, we require an email and password, as an attacker an admin email or an email with high privileges is preferred. However gaining access to as many accounts as possible is also beneficial because we could exploit broken authorization to elevate unprivileged accounts privilege. These will also allow us to test out patch.

To find accounts email addresses, we began combing through reviews of products on the store page. After reviewing them all, we found an admin account with the email address admin@guardian.com. As well as jim@guardian.com, bender@guardian.com, uvogin@guardian.com, stan@guardian.com, bjoern@owasp.org, morty@guardian.com, mc_safesearch@guardian.com, and accountant@guardian.com user accounts.

This made it easy to find accounts and left me with just having to get or reset the password. This could be gotten by using social engineering, or brute force attacks, to guess the password. As well as, resetting the password, by answering the security question.


This reconnaissance vulnerability could be patched with the username showing instead of the address i.e., "Jim" instead of jim@guardian.com.

Login

Email *

Password *


admin@guardian.com

One of my favorites! 


jim@guardian.com

Fresh out of a replicator. 


uvogin@guardian.com

0 st4rs f0r 7h3 h0rr1bl3 s3cur17y 


stan@guardian.com

I'd stand on my head to make you a deal for this piece of art. 


bender@guardian.com

Just when my opinion of humans couldn't get any lower, alona comes Stan... 


accountant@guardian.com

DO NOT PLAY WITH THIS! Double-sleeve, then put it in the GitHub Arctic Vault for perfect preservation and boost of secondary market value! 

morty@guardian.com

Even more interesting than watching Interdimensional Cable! 

bjoern@owasp.org

Wait for a 10\$ Steam sale of Tabletop Simulator! 

Exploitation

To gain access to the admin account, we used the previously designed payload to perform a, most common admin passwords brute force sniper payload attack, on the admin account, as well as a most common surnames brute force sniper payload attack. This is because having access to both the password, or the means to change the password, would give me access to the account. The more boarder the attack, the more avenues used, and more surfaces exploited, the more likely for a successful breach. Using the password payload, we found that the password 'admin123' was correct, this highlights the current insecurity, and need for a patch. This would within a few hours grate a third-party access to the admins account and allowing them to change the password, locking us out of the account aswell as create false products, reviews and otherwise compromise the security, confidentiality, integrity and availability of the Guardian Store System.

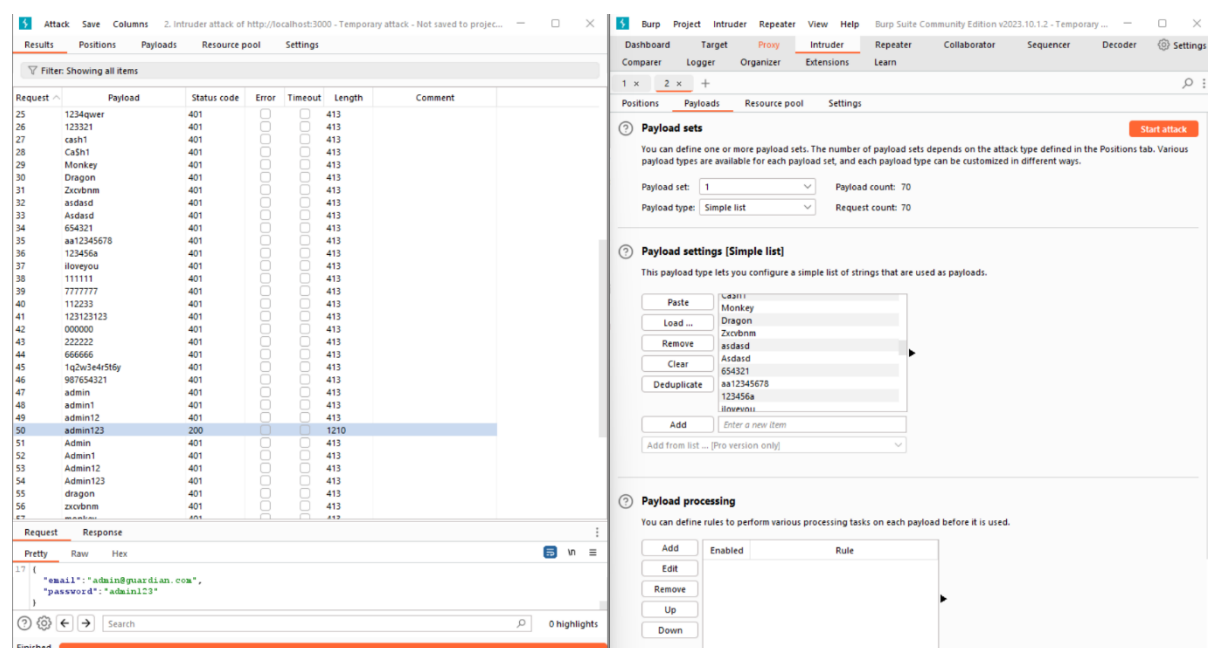


Figure 2. Burpsuite Brute Force Payload Sniping Attack, using the preestablished payload this bruteforce attack found the password of the admin account is 'admin123'. This is shown by the successful status code 200, and the different length verses the failed unauthorized attempts.

Patch

To mitigate against the password strength authentication vulnerability, we should implement a password strength test before allowing the resetting or changing of a password. This can be done rather simply using RegExp, which allows you to create a pattern to test a string with. By Applying the pattern we created previously, to a password using the test function we can ensure the password is at least 5 characters long, with a lowercase, uppercase, number and symbol.

This was added to the reset & change password function, though the check password strength function. This ensures that old insecure passwords are phased out with new secure ones in time. While simultaneously ensuring users don't use weaker passwords like Ca\$h1, as they are wont to do, when forced to change their password. Attackers would also assume that all passwords contained an uppercase, lowercase, number and symbol so this fix simultaneously strengthens weak passwords as they'll be less likely to be added to the payload. This is because attackers will use base their password payloads on password requirements. By keeping the error vague by adding 's' at the end of numbers and symbols, further protects against attackers using our requirements to make attacking easier.

```
lib > TS passwordStrength.ts > ...
1
2 // function to run a RegExp pattern check to check a passwords strength
3 export function checkStrength (password: any): boolean {
4   try {
5     // if password is strong return true, else return false
6     /*
7     regex pattern to check for password strength:
8     /^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#%&'\^&#x26;\/\|,;<.>;|_~^`\[\]\(\)]).{5,}/
9     RegExp Code for atleast 1 upper, lower, number & special character with min length 5
10    */
11    // eslint-disable-next-line no-useless-escape
12    if (/^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#%&'\^&#x26;\/\|,;<.>;|_~^`\[\]\(\)]).{5,}/.test(JSON.stringify(password))) {
13      return true
14    } else {
15      return false
16    }
17  } catch (err) {
18    console.log(err)
19  }
20  return false
21 }
22
```

```
routes > TS changePassword.ts > ...
27 // Runs the password checkStrength function
28 } else if (!checkStrength(newPassword)) {
29   res.status(401).send(res._('Ensure your new password has uppercase, and lowercase letters, as well as, numbers and symbols.'))
30 } else if (newPassword !== repeatPassword) {
```

```
routes > TS resetPassword.ts > ...
35 // Runs the password checkStrength function
36 } else if (!checkStrength(newPassword)) {
37   res.status(401).send(res._('Ensure your new password has uppercase, and lowercase letters, as well as, numbers and symbols.'))
38 } else if (newPassword !== repeatPassword) {
```

Figure 3. code snips of changePassword, resetPassword & checkPassword function highlighting the patch implementation.

Furthermore, by adding a notification informing users to update their passwords, manually updating the admins passwords and by utilising defence-in-depth to enforcing a strict internet security policy among our staff. we can ensure weak admin passwords like 'admin123' don't exist. In effect patching out the password strength vulnerability.

In the future we could further protect against brute force attacks, by implementing a max access attempt threshold, which when passed, would lock down the account. Stopping a brute force payload attack dead in its tracks.

We would also need to implement an unlock mechanism, to ensure users can contact support, have their password changed and their account unlocked.

Multi-Factor Authentication through an Email or SMS authentication request, requiring validation, would mitigate the attack. As third-parties would be required to have access to the request source, in order to get access to the account.

During testing it was found that the known 'Database Wallet' issue is preventing changed passwords from being permanent. It is recommended, that this common database issue is resolved, as it delays sight startup time and forces a database reboot. The only way around this would be keeping the site active 24/7 and that's not a feasible requirement as down time is always necessary for routine site maintenance.

Forgot Password

Ensure your new password has uppercase, and lowercase letters, as well as, numbers and symbols.

The form contains four input fields: 'Email *' with the value 'jim@guardian.com', 'Security Question *' with five dots, 'New Password *' with five dots, and 'Repeat New Password *' with five dots. Below the 'New Password' field, a message states 'Password must be 5-40 characters long.' with a '5/20' character count. At the bottom, there is a toggle switch for 'Show password advice' which is currently turned off.

Figure 4. Reset password attempt using weak password 'stars'. Demonstrating password strength patch working. Ambiguity added to the error, as only 5 characters 1 upper, 1 lower, 1 a symbol and 1 a number required. Keeping the error vague by adding s' at the end of numbers and symbols, protects against attackers using our requirements to make attacking easier.

Forgot Password

Your password was successfully changed.

The form contains four input fields: 'Email *' with the value 'jim@guardian.com', 'Security Question *' with five dots, 'New Password *' with six dots, and 'Repeat New Password *' with six dots. Below the 'New Password' field, a message states 'Password must be 5-40 characters long.' with a '6/20' character count. At the bottom, there is a toggle switch for 'Show password advice' which is currently turned off.

Figure 5. Reset password attempt using strong password 'Star\$1'. Demonstrating password strength patch working. This password shows it doesn't guarantee strong passwords. Regardless it does makes things harder for brute force attackers.

Security Opportunities

Defence-In-Depth

Definition

Defence-In-depth is a multi-layered security control methodology devised by the National Security Agency (NSA), and is a globally prevalent standard, for cyber security and data protection. It's aim is to supply redundancy, procedural, technical, personal, and physical security.

Defence-In-Depth is often segmented into physical, technical, and administrative controls.

Physical: physical controls, defending a system i.e., guards, locks, fences, security, etc.

Technical: technical controls, securing a system i.e., Firewalls, Authentication, etc.

Administrative: administrative controls, protecting a system i.e., policies, procedures, etc.

The layers that make up Defence-In-depth are System & Application, Network, and Physical. This includes but isn't limited to the following methods:

System and application:	Network:	Physical:
Antivirus software	Firewalls (hardware or software)	Biometrics
Multi-factor authentication	Demilitarized zones (DMZ)	Data-centric security
Encryption	Virtual private network (VPN)	Physical security
Hashing passwords		
Vulnerability scanners		
Timed access control		
Internet Security Awareness		

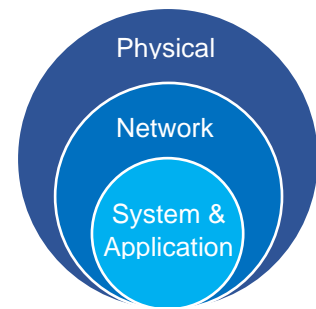


Figure 6. Layered Defense Diagram

Opportunities

Technical: System, Application and Network

Multi-factor Authentication would be a great opportunity to fix our broken authentication & authorisation, this is done by adding third-party authorisation like email or SMS. This makes attempting to break into the account or reset the password far harder.

Encryption and password hashing should also be used to secure our backend data and mitigate man-in-the-middle attack scouting.

Administrative: System, Application and Physical

An Administrative Defence-In-Depth opportunity would be teaching our administrators Internet Security Awareness. By learning common tactics, threats and red flags for social engineering or other attack methods, we can mitigate their effectiveness at tricking our staff into compromising system security. Intern further strengthening our system.

Physical: Network and Physical

To Physically protect our system, we would implement firewalls between our servers on the Network layer, and on the Physical layer we should implement site security protocols, with guards, and surveillance, to ensure no physical harm, or improper access can happen on our system.

([2] NSA. 2012).

Secure Software Development Mitigations

Definition

Secure Software Development Principles make recommendations on how to strengthen our cyber security and mitigate against attacks or intrusions..

Principles:

- Least Privilege
Minimum privileges required for proper function.
- Open Design
Security can't depend on secrecy.
- Fail-Safe Defaults
Defaults created that maintain security, in case of failure.
- Separation of Privilege
Checks should require multiple conditions to be met.
- Economy of Mechanism
Simple security design and controls, increased stability.
- Least Common Mechanism
Minimise commonalities and dependencies between users.
- Complete Mediation
Access must be authorised before granting access.
- Psychological Acceptability
Controls should be easy-to-use, understand, and control, avoiding bypass.

Opportunities

Least Privileges

We should decompress the admin@guardian.com account as it contains too many privileges. E.g. a store@guardian.com containing product creation privileges, reviews@guardian.com containing review privileges etc.

Least Common Mechanism

Reducing commonalities between users e.g., admin and user portals.
Usernames in reviews and not emails, to protect against recognisance.

Common Mediation

Multi-factor authentication should be implemented on change/reset password if not on the login itself. This could prevent future false logins and make the process of breaking into a user's account far harder.

Separation of Privilege

We should implement a task list handled by an overseer account, that requires there be a task set, to add a product, or do a action, before allowing accounts to perform said action. This would protect against malicious access and modification.

Physiological Acceptability

We should also create a Password Strength Policy to ensure that administrative passwords are never this weak again. This policy should be physiologically acceptable so that administrators aren't compelled to find work arounds, potentially compromising our security.

([1] OWASP. 2021, [3] A., Barrett. 2015)

Automated Password Strength Testing

To prevent this vulnerability from occurring in the future, an automated password strength tester was added to the login.ts file. This checked the login password of any given user, and if it failed the strength test, logged that the email failed on logs/weakPassword.log.txt as well as the time of failure. This log could be used in the future to send emails to these users informing them that they should update their passwords.

This would allow us to track which users need to update their passwords, without compromising their account security by logging their passwords. Intern providing an automated method for testing and eliminating password strength vulnerability present in the website.

```
44  ✓    if (!checkStrength(req.body.password)) {  
45      ✓        console.log('Writing')  
46      ✓        try {  
47      ✓            writeFileSync('logs/weakPasswordLog.txt', req.body.email +  
48      ✓                (' failed automated password strength test ' + (new Date()).toString() + '\n').toUpperCase(), {  
49      ✓                flag: 'w'  
50      ✓            })  
51      ✓        } catch (err) {  
52      ✓            console.log(err)  
53      ✓        }  
54      ✓        console.log('Stop Writing')  
55      ✓    }
```

Figure 7. automated vulnerable password checking on login, this sends emails associated with failed/weak passwords to a WeakPasswordLog so in the future they can be notified and suggested to change their passwords.

Conclusion

By implementing the Password Strength Patch and Automated Password Strength Testing, we can start to fix the broken authentication issues present on Guardian Store and patch out the Password Strength Vulnerability.

Through making use of opportunities presented to us by Defence-In-Depth methods; Implementing Multi-factor Authentication, Backend Encryption, Internet Security Awareness and Physical Defences. As well as Secure Software Development Mitigation Techniques; Least Privileges privilege decompression, Least Common Mechanism reducing commonalities, Common Mediation again with Multi-Factor Authentication, Separation of Privilege with decompression and a task system, and finally Physiological Acceptability, making out system usable and preventing admins creating or using backdoors. We can further defend our system from future attacks and make guardian store the safest shop on the web.

Bibliography & References

- [1] OWASP. (2021). Secure Product Design - OWASP Cheat Sheet Series. [online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Secure_Product_Design_Cheat_Sheet.html#2-the-principle-of-defense-in-depth.
- [2] NSA .(2012). Defence In Depth. [online] Available at: <https://web.archive.org/web/20121002051613/https://www.nsa.gov/ia/ files/support/defenseindepth.pdf>.
- [3] A., Barrett. (2015). Some principles of secure design – Designing Secure System module Autumn 2015. [online] Available at: <https://docplayer.net/17241198-Some-principles-of-secure-design-designing-secure-systems-module-autumn-2015.html>
- [4] OWASP. (2021). A01 Broken Access Control - OWASP Top 10:2021. [online] owasp.org. Available at: https://owasp.org/Top10/A01_2021-Broken_Access_Control/.
- [5] Miessler, D. (2022). Daniel Miessler SecLists Top 10000 Passwords. [online] GitHub. Available at: <https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-10000.txt>.
- [6] PowerShell Gallery (2023). PowerShell Gallery | Surnames.txt 0.1.1. [online] Available at: <https://www.powershellgallery.com/packages/TelligentCommunitySample/0.1.1/Content/Surnames.txt>.
- [7] OWASP (2021). OWASP Top Ten. [online] Owasp.org. Available at: <https://owasp.org/www-project-top-ten/>.
- [8] Wilson, T., Chief, E. (2007). Top 10 Admin Passwords to Avoid. [online] Dark Reading. Available at: <https://www.darkreading.com/perimeter/top-10-admin-passwords-to-avoid>.
- [9] Annie. (2022). Top 50 Most Common Passwords & Why You Shouldn't Use Them - PPWP. [online] Available at: <https://passwordprotectwp.com/most-common-passwords/>.
- [10] RegExp. (2023). RegExp - JavaScript | MDN. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp.
- [11] Sebastian, N. (2023). Top Password Strengths and Vulnerabilities: Threats, Preventive Measures, and Recoveries. [online] Available at: <https://www.goodfirms.co/resources/top-password-strengths-and-vulnerabilities>.
- [12] OWASP (2017). Authentication · OWASP Cheat Sheet Series. [online] Owasp.org. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html.