**Guardian Store Vulnerability Report**

**Table of Contents**

[**Table of Figures** 2](#_Toc148883347)

[**Executive Summary** 3](#_Toc148883348)

[**Overview** 4](#_Toc148883349)

[**Introduction** 4](#_Toc148883350)

[**Vulnerabilities Identified** 4](#_Toc148883351)

[**Chosen Vulnerability:** Broken Authorization/Broken Access Control: Password Strength 5](#_Toc148883352)

[**Introduction** 5](#_Toc148883353)

[**Research** 5](#_Toc148883354)

[**Reconnaissance** 6](#_Toc148883355)

[**Exploitation** 6](#_Toc148883356)

[**Patch** 7](#_Toc148883357)

[**Security Opportunities** 8](#_Toc148883358)

[**Defence-In-Depth** 8](#_Toc148883359)

[Definition 8](#_Toc148883360)

[Opportunities 8](#_Toc148883361)

[**Secure Software Development Mitigations** 9](#_Toc148883362)

[Definition 9](#_Toc148883363)

[Opportunities 9](#_Toc148883364)

[**Conclusion** 10](#_Toc148883365)

[**Bibliography & References** 11](#_Toc148883366)

# **Table of Figures**

Figure 1………………………………………………………..………………………………………6

Figure 2………………………………………………………..………………………………………8

Figure 3…………………………………………………………..……………………………………9

Figure 4……………………………………………………………..…………………………………9

Figure 5………………………………………………………………..……………………………..10

# **Executive Summary**

Guardian has contacted us to penetration test GuardianStore, by identifying vulnerabilities and attempt to resolve them. For this report we will be focusing on the Broken Authentication/Broken Access Controls vulnerability class and the chosen vulnerability within said class, Password Strength. wet is a major vulnerability class which could compromise the account security and confidentiality of clients and admins alike. We will be focusing mainly on it’s 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, this will require a payload of common user and administrator passwords. By combing a list of common admin passwords found online ([8] Wilson, T. 2007), and user passwords ([9] Annie. 2022). This 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 better exploit the vulnerability we first performed reconnisence, 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](mailto:jim@guardian.com), [bender@guardian.com](mailto:bender@guardian.com), [uvogin@guardian.com](mailto:uvogin@guardian.com), [stan@guardian.com](mailto:stan@guardian.com), [bjoern@owasp.org](mailto:bjoern@owasp.org), [morty@guardian.com](mailto:morty@guardian.com), [mc\_safesearch@guardian.com](mailto:mc_safesearch@guardian.com), and [accountant@guardian.com](mailto:accountant@guardian.com). As well as the admin account [admin@guardian.com](mailto: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 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 to the else if conditions which would fail the password, a RegEx string test. 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. We could manually change the administer passwords to more secure variants and in the future put a notification advising users to update their passwords. Using this method, we can slowly phase out the weak passwords and make them stronger as attacks 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.

# **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.

## **Vulnerabilities Identified**

The following vulnerabilities have been identified:

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

A screen shot of a computer program

Description automatically generated

## **Research**

|  |  |
| --- | --- |
| 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 |
| passwrd | 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 |

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 coniditons 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.

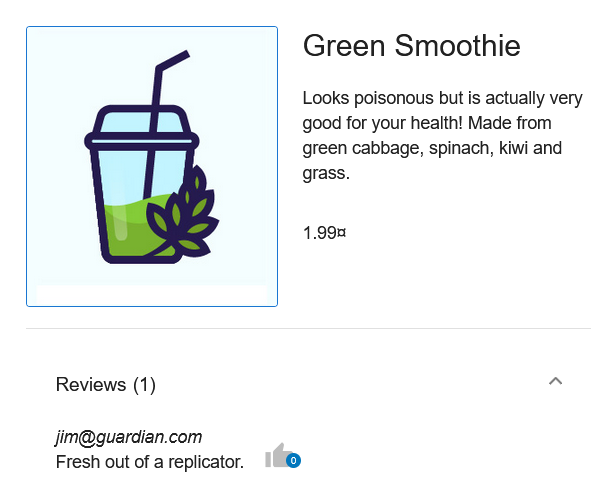
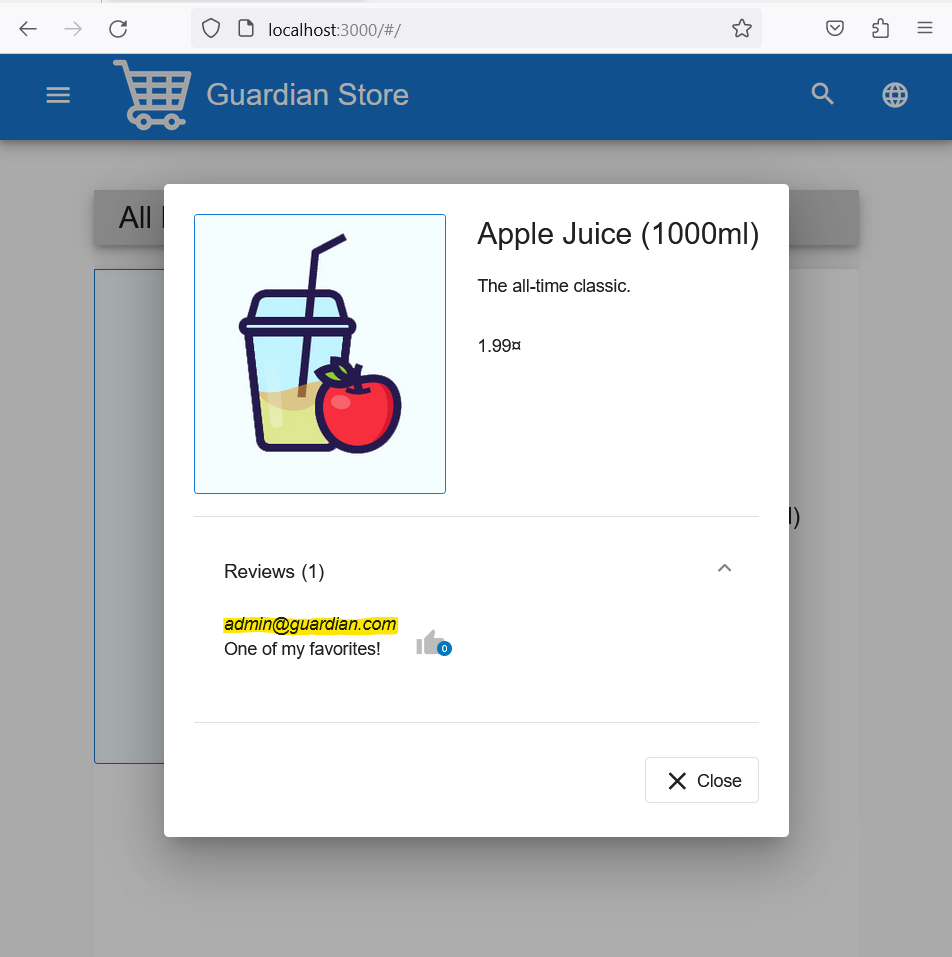
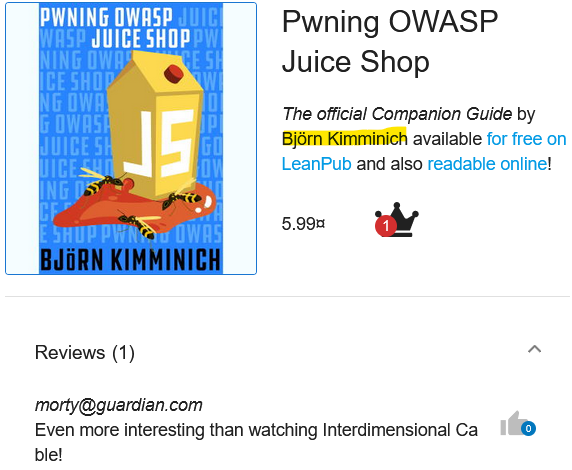
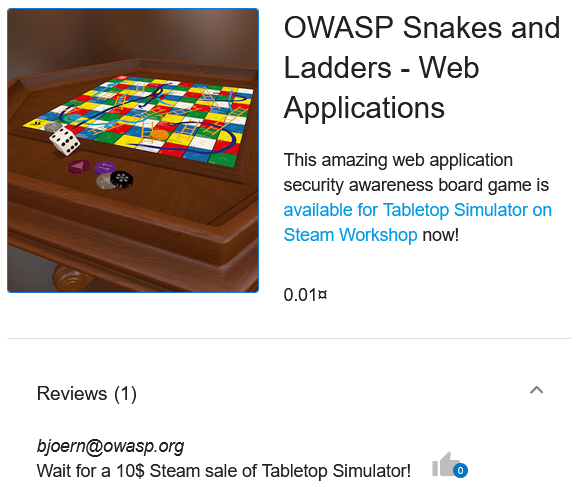
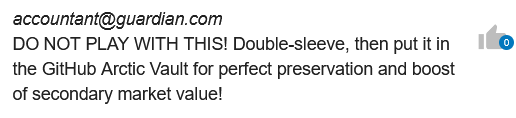
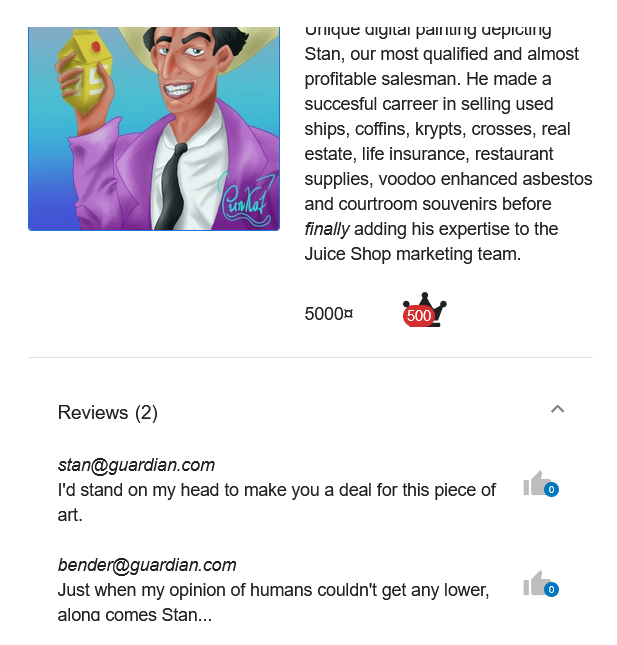
Figure 1. password payload, madeup from 10 of the most common admin and 50 of the most common user password. Aswell as 22 variations on those passwords.

## **Reconnaissance**

A screenshot of a login form

Description automatically generatedTo 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](mailto:admin@guardian.com). As well as [jim@guardian.com](mailto:jim@guardian.com), [bender@guardian.com](mailto:bender@guardian.com), [uvogin@guardian.com](mailto:uvogin@guardian.com), [stan@guardian.com](mailto:stan@guardian.com), [bjoern@owasp.org](mailto:bjoern@owasp.org), [morty@guardian.com](mailto:morty@guardian.com), [mc\_safesearch@guardian.com](mailto:mc_safesearch@guardian.com), and [accountant@guardian.com](mailto: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](mailto:jim@guardian.com).

## **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.

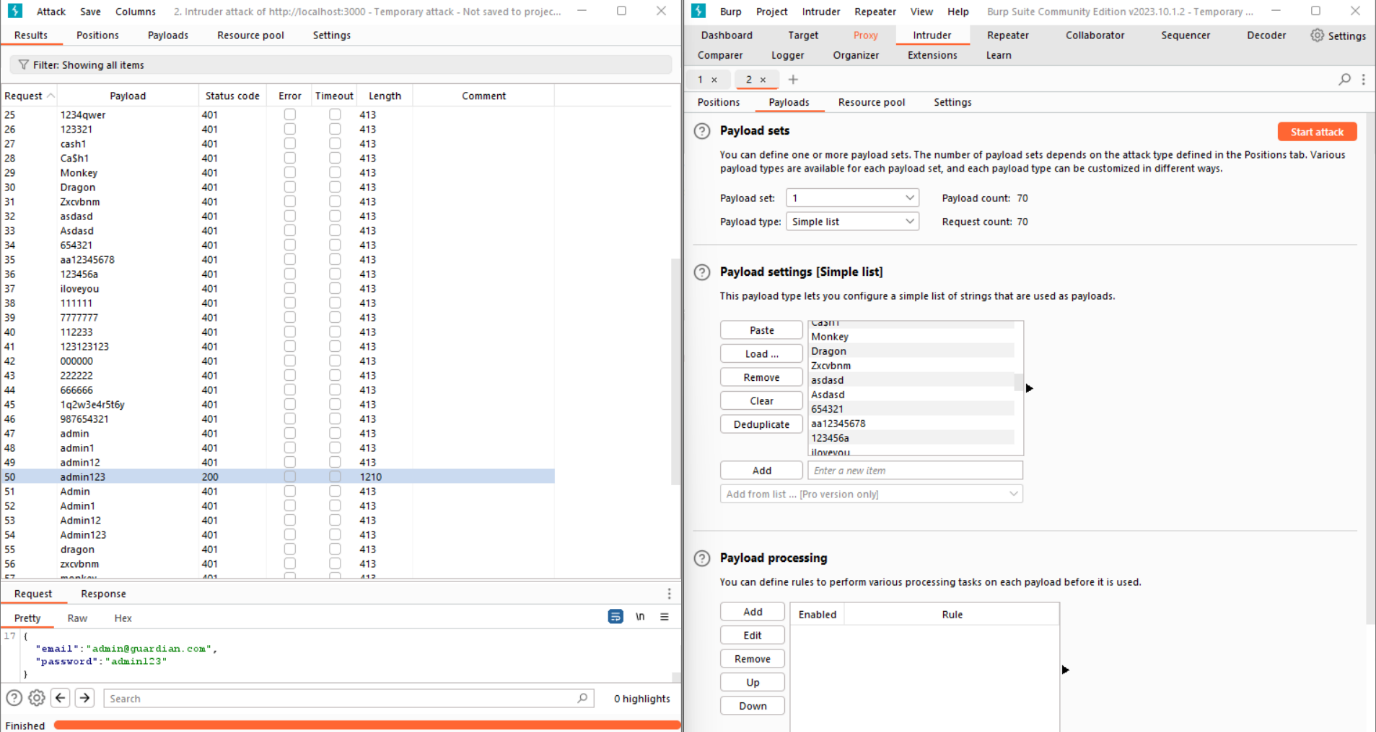


Figure 2. Burpsuite Brute Force Payload Sniping Attack, using the preestablished payload this bruteforce attack found the password of the admin account is ‘admin123’. This is shown by the successful status code 200, and the different length verses the failed unauthorized attempts.

## **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. Our pattern would ensure the password is atleast 5 characters long, with a lowercase, uppercase, number and symbol.  
This was added to the reset & change password function, ensuring that old insecure passwords are phased out with new secure ones in time. While simultaneously ensuring uses don’t use weaker passwords like Ca$h1, as they are wont to do, when forced to change their password.

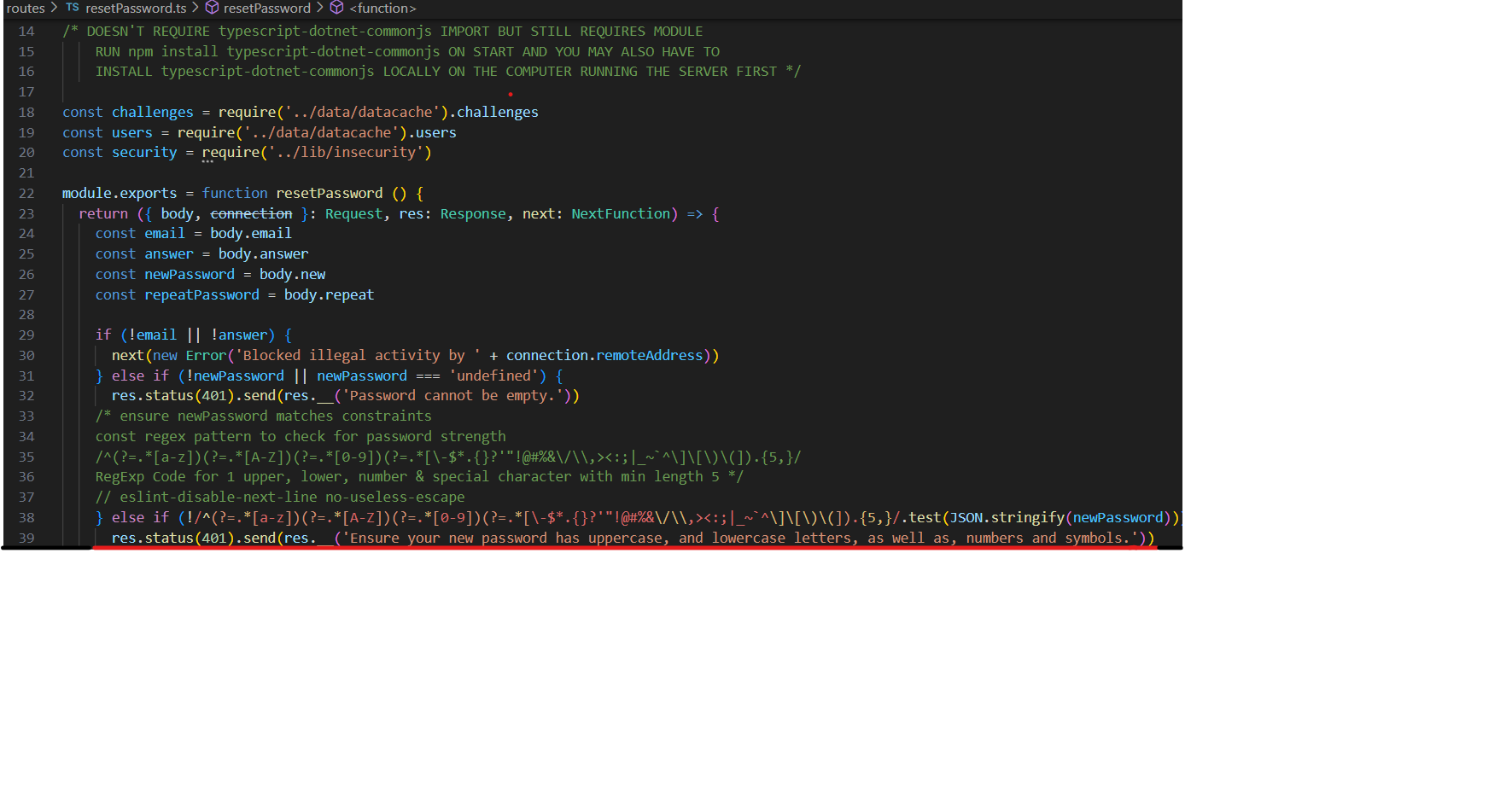
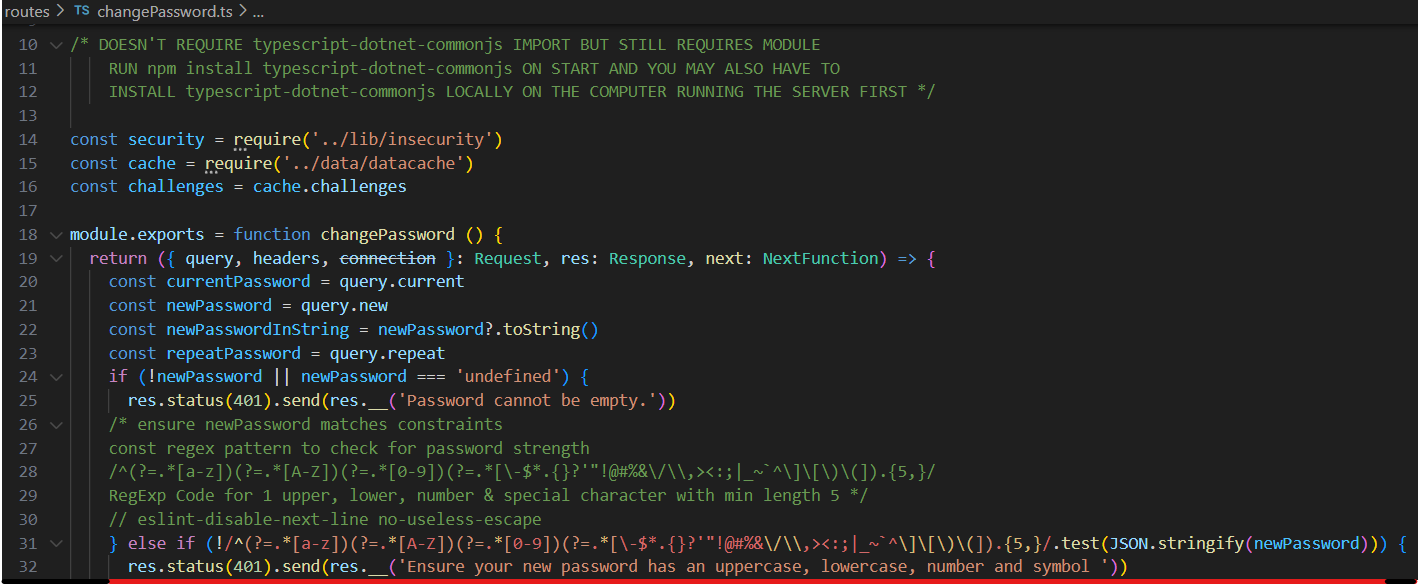
Furthermore, by adding a notification informing users to update their passwords, manually updating the admins passwords and by utilising defence-in-depth to enforcing a strict internet security policy among our staff. we can ensure weak admin passwords like ‘admin123’ don’t exist. In effect patching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

Figure x. code snips of changePassword and resetPassword functions highlighting the patch implementation.

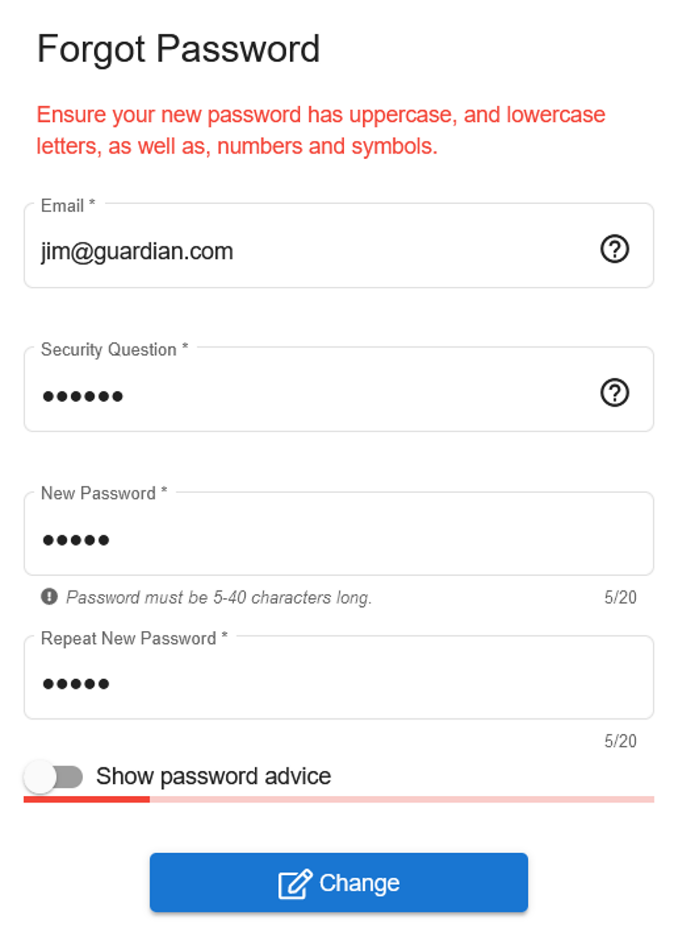
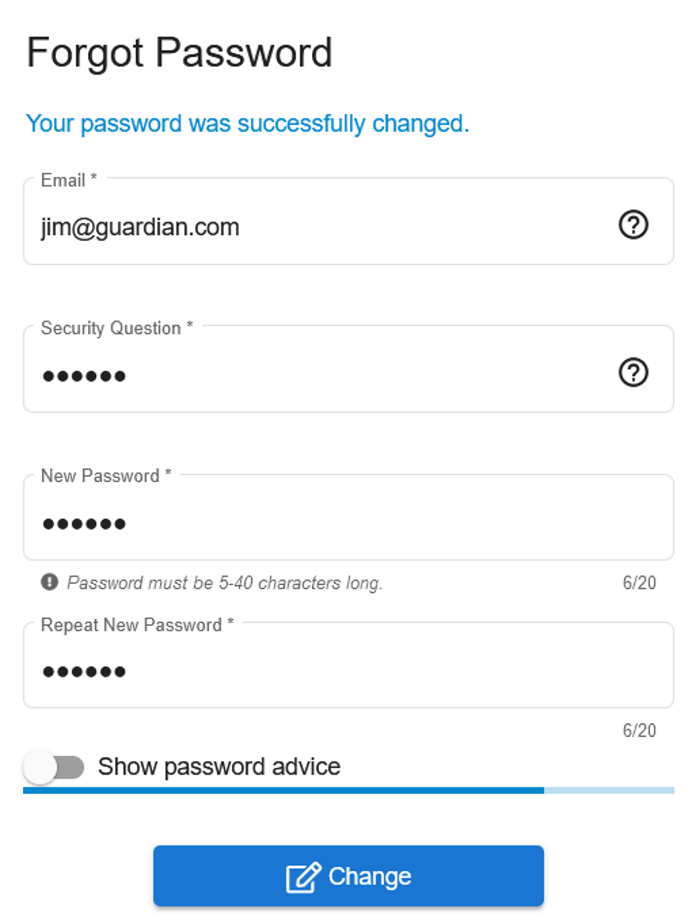


Figure 3. Reset password attempt using weak password ‘stars’. Demonstrating password strength patch working. Ambiguity added to the error, as only 5 characters 1 upper, 1 lower, 1 a symbol and 1 a number required. Keeping the error vague by adding s’ at the end of numbers and symbols, protects against attackers using our requirements to make attacking easier.

Figure 4. 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**

System & Application

Network

Physical

## **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.

Figure 5. Layered Defense Diagram

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

### Opportunities

**Technical:** System, Application and NetworkMulti-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.

* Fail-Safe Defaults

Defaults created that maintain security, in case of failure.

* Economy of Mechanism

Simple security design and controls, increased stablity.

* Complete Mediation

Access must be authorised before granting access.

* Open Design

Security can’t depend on secrecy.

* Separation of Privilege

Checks should require multiple conditions to be met.

* Least Common Mechanism

Minimise commonalities and dependencies between users.

* Psychological Acceptability

Controls should be easy-to-use, understand, and control, avoiding bypass.

### Opportunities

**Least Privileges**  
We should decompress the [admin@guardian.com](mailto:admin@guardian.com) account as it contains too many privileges. E.g. a [store@guardian.com](mailto:store@guardian.com) containing product creation privileges, [reviews@guardian.com](mailto: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 lib/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.

# **Conclusion**

By implementing the Password Strength Patch and 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, and Secure Software Development Mitigation Techniques 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](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.](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](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp).