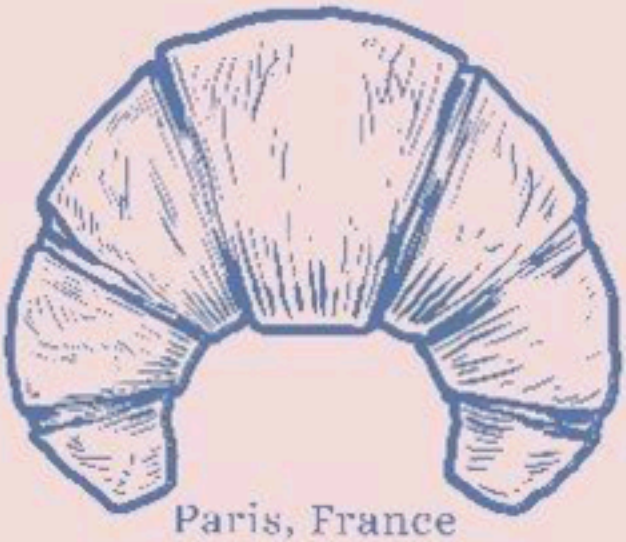
2022

LE BONBON CROISSANT PENETRATION TEST REPORT



Paris, France

LE BONBON CROISSANT

CPTC 2021

1/9/2022

Contents

Executive Summary

On January 8th, 2022, [redacted] completed a security assessment of the company Le BonBon Croissant (LBC) based in France. Per the requirements from Le BonBon Croissant, [redacted]s scope focused on the security within the customer experience, warehouse industrial controls systems, cardholder data environment, and payment processing applications across Le BonBon Croissant's network. [redacted] assessed these findings based upon the potential risk they posed to LBC's business operations from the standpoint of financial impact, productivity cost, and the likelihood of occurrence.

The assessment was completed as a follow up to a previous assessment completed in November of 2021. [redacted] members noted several improvements by LBC by removing previously vulnerable machines and services while also increasing the security of their communications. However, several vulnerabilities still remained which were exploited and documented.

During the assessment, [redacted] identified several weaknesses throughout Le BonBon Croissant in both the technology and the policies of the company. These weaknesses allowed [redacted] to compromise the Le BonBon Croissant system, finding flaws that could result in legal action and heavy financial penalties due to compliance with PCI-DSS and GDPR. Therefore, the overall rating of SEVERE was given to Le BonBon Croissant. In accordance with our risk classification, SEVERE is defined as:

"Customer or company data is being exfiltrated or significantly altered. The business may not be operational. Staff should be working 24/7 to remediate the impact. There is a potential for catastrophic financial and reputational impact to Le BonBon Croissant."

The vulnerabilities discovered include credential reuse, payment manipulation, and insecure storage of customer information. These vulnerabilities have the potential to severely impact the reputation of Le BonBon Croissant, the customer experience, and penalties in excess of 20 million euros.

To secure the company and align with regulations, [redacted] recommends Le BonBon Croissant take the following actions.

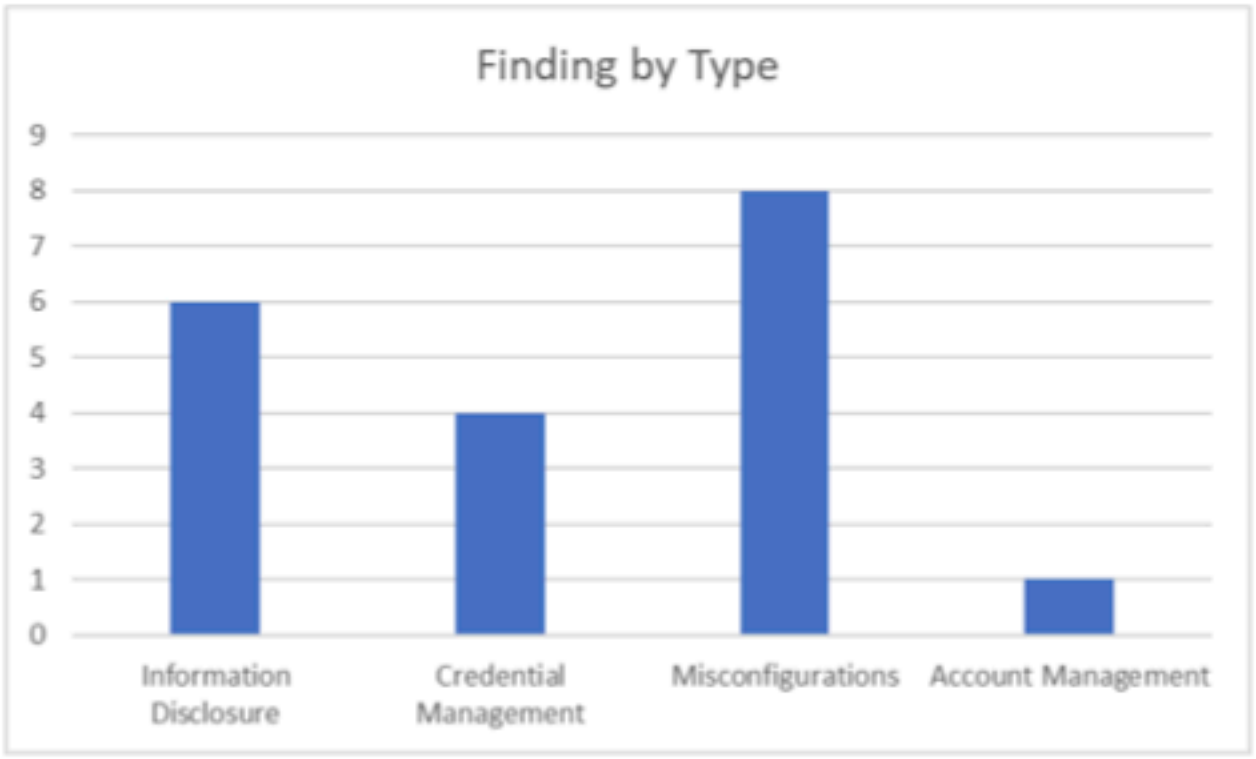
1. Implement an organization-wide password policy discouraging password reuse
2. Store and transfer information in compliance with PCI-DSS
3. Implement stringent authentication controls in compliance with PCI-DSS for payment processes

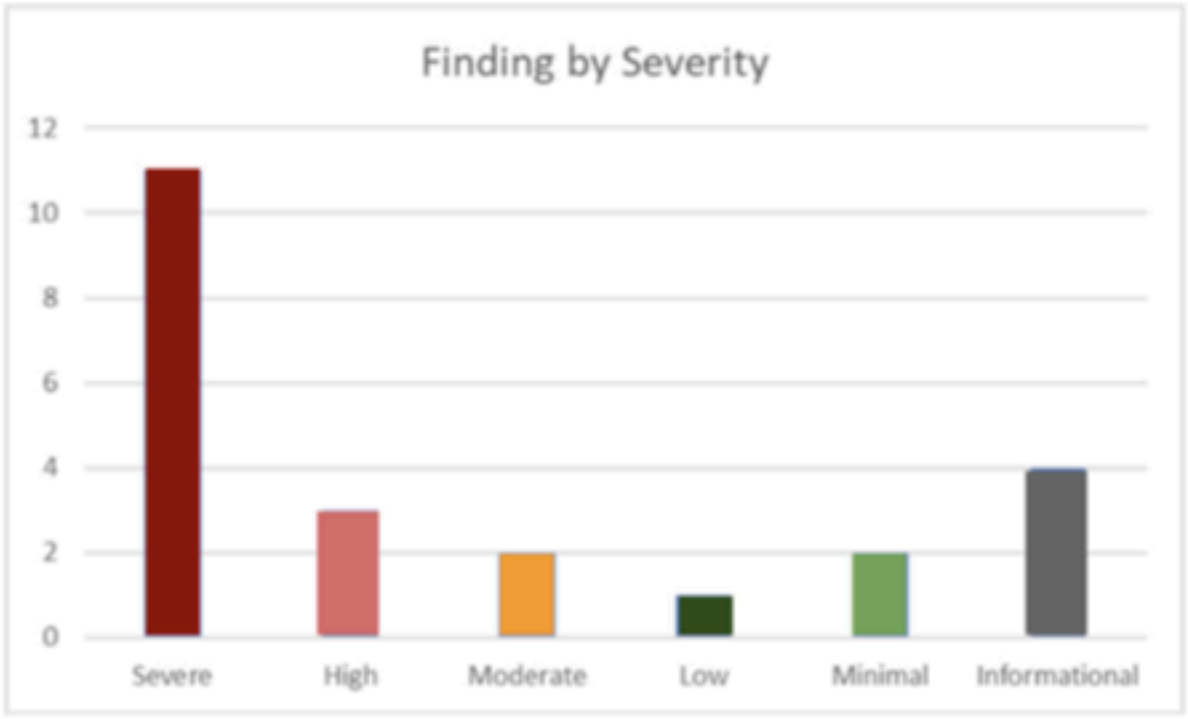
Scope

The scope of the assessment was determined by the parameters set forth by Le BonBon Croissant and the PCI-DSS standards to which LBC must be compliant. Therefore, over the course of a two-day assessment period [redacted] assessed the network of LBC, initially excluding the industrial control systems of their shipping operation. On the second day though the shipping systems were added to the assessment. Throughout the entire assessment, social engineering was out of scope.

In accordance with PCI-DSS standards, the penetration test was conducted with a focus on the cardholder data environment (CDE). This included all internal and external network and application components relating to the transmission, storage, and use of cardholder data. People were excluded from the process per the requirements from LBC, but any employee data or workstation that interacted with the CDE was included within the scope.

Findings Summary





Strategic Recommendations

Password Policy

Based upon the assessment [redacted] recommends the following organizational level improvements to increase the security of LBC. The flaws that could lead to the most critical business impact involve how passwords are managed by LBC. It is recommended that LBC adopts a strong password policy requiring that passwords be complex and not reused organization-wide. Most importantly, credentials should be specific to a person even if used for the purpose. A password should not be reused for administrators across machines since it can lead to catastrophic security breaches. It is recommended that password reuse be a cornerstone of the new password policy in addition to password strength. Examples of password standards can be found at the following link: https://pages.nist.gov/800-63-3/sp800-63b.html. Passwords should be at least 8 characters long and reject repeated characters. Additionally, NIST recommends that information about the username should not be included in the password.

Encryption

Furthermore, while services were improved to use encryption and authentication before revealing sensitive information, some critical information was still stored at rest in cleartext. It is recommended that the company store all critical customer information in an encrypted form even to be compliant with PCI-DSS. Credit card information specifically should be stored in a secure form when not being used and only unencrypted when being used.

Account Management

Finally, it is recommended that an overall access control system be created to separate user privileges. There are several services and resources that are available to all users by default as there are no other roles besides root or administrator. In essence, accounts need to be separated by purpose, giving each user the least amount of access necessary to perform their tasks.

Organization-wide implementation of these recommendations will help to reduce many of the attack vectors presented below.

Risk Classification

Business Impact Classification

Severe (>20):

Customer or company data is being exfiltrated or significantly altered. The business may not be operational. Staff should be working 24/7 to remediate the impact. There is a potential for catastrophic financial and reputational impact to Le BonBon Croissant.

High (15-19):

Customer or company data is being affected. Systems or data may be unavailable, and customers cannot complete business due to disruptions. Staff needs to be called in past business hours to remediate the impact. There are significant financial or reputational impacts to Le BonBon Croissant.

Moderate (10-14):

Customer or company data may be unavailable but not exfiltrated or altered. Production systems are affected, and customers are affected by the disruption. Staff may be required to work past business hours to remediate the impact. There is a noticeable financial or reputational impact to Le BonBon Croissant.

Low (5-9):

Customer and company data are not affected. Production systems may be impacted, but the disruptions do not impact customers. Day to day staff should be able to remediate any disruptions within the workday. There is a minimal financial or reputational impact to Le BonBon Croissant.

Minimal (<5):

Customer data, company data, and production systems are not affected. Day-to-day staff can remediate the disruptions within their regular work duties without extra time allocated. There is little to no financial or reputational impact to Le BonBon Croissant.

Governance and Compliance

Payment Card Industry Data Security Standard (PCI-DSS) is required by law for those handling cardholder data. Companies must be annually evaluated for their compliance with PCI-DSS and must be found compliant to accept payment via card. Companies are evaluated on how they store, transmit, and secure financial information relating to customer transactions. Failure to comply could result in fines up to $100,000 per month and legal action. Actions include, but are not limited to, further audits and assessments that must be paid for by the violating company.

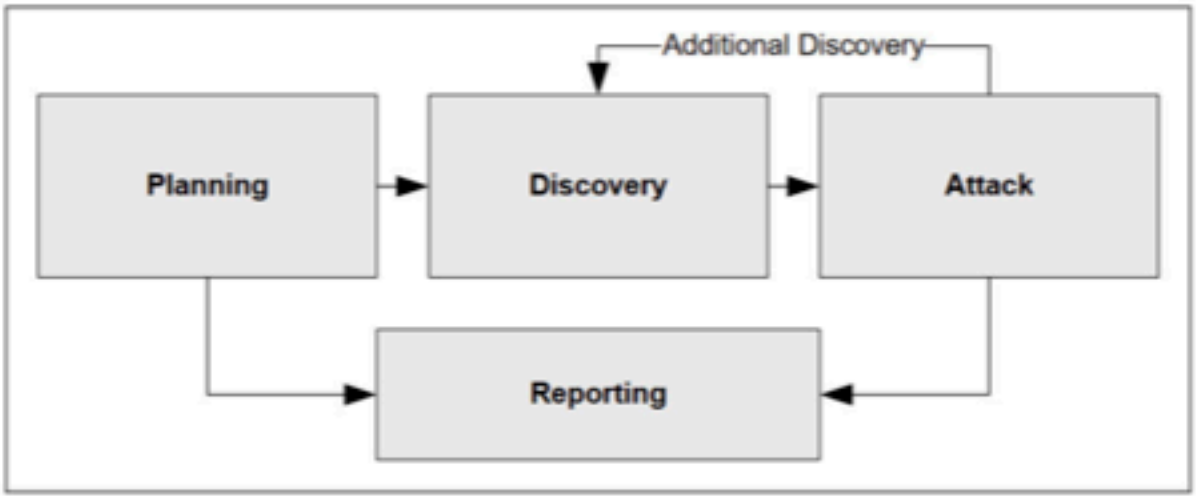
General Data Protection Regulation (GDPR) was drafted and passed by the European Union (EU) and applies to anyone who has customers from the EU. Failure to comply could result in fines up to 20 million euros or 4% of the company's revenue, whichever is greater. Further legal action can be taken which in the most extreme case can result in the suspension of operations.

Since these standards serve as the basis for transaction processing that the organization operates in, [redacted] has aligned findings from the engagement with PCI-DSS and GDPR standards to provide insight into the financial and legal implications of not complying with these international standards. The following chart provides a breakdown of where LBC was found to be out of compliance with PCI-DSS and GDPR.

PCI Compliance Findings	
Violation	Compliance
Plaintext Storage of Personal Information	PCI-DSS - Protect Stored Cardholder Data
Plaintext Storage of Credit Cards	PCI-DSS - Protect Stored Cardholder Data
Weak Logins	PCI-DSS - Passwords must be a minimum of seven characters and change every 90 days
Out of Date Systems	PCI-DSS - Develop and Maintain Secure Systems and Apps
Overprovisioned Access	PCI-DSS - Restrict Access by Business Need to Know
Unauthenticated Memcached and PLC	PCI-DSS - Identify and Authenticate Access to System
Website does not include right to be forgotten	GDPR
No appointed Data Protection Officer	GDPR

Penetration Test Methodology

[redacted] followed a custom methodology framework incorporating aspects of Penetration Testing Execution Standard (PTES) and NIST 800-115. The final framework had four phases incorporating the seven phases of PTES into the NIST framework. The four phases are shown, which spanned the pre-assessment and assessment phases of the engagement.

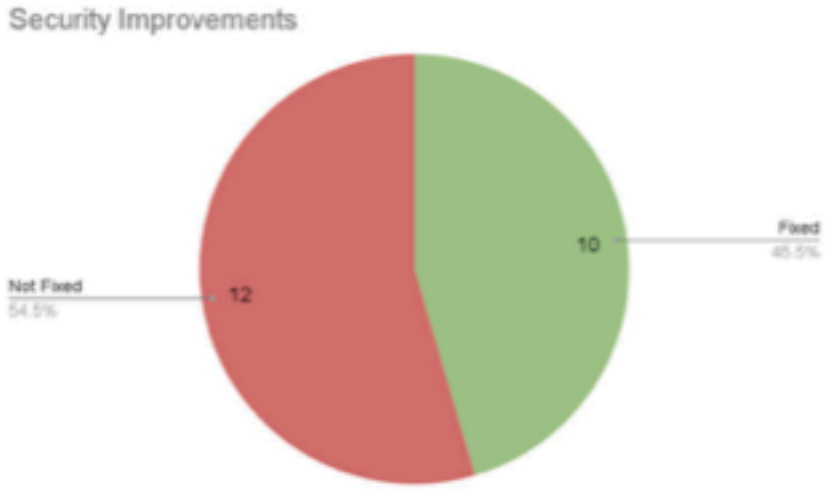


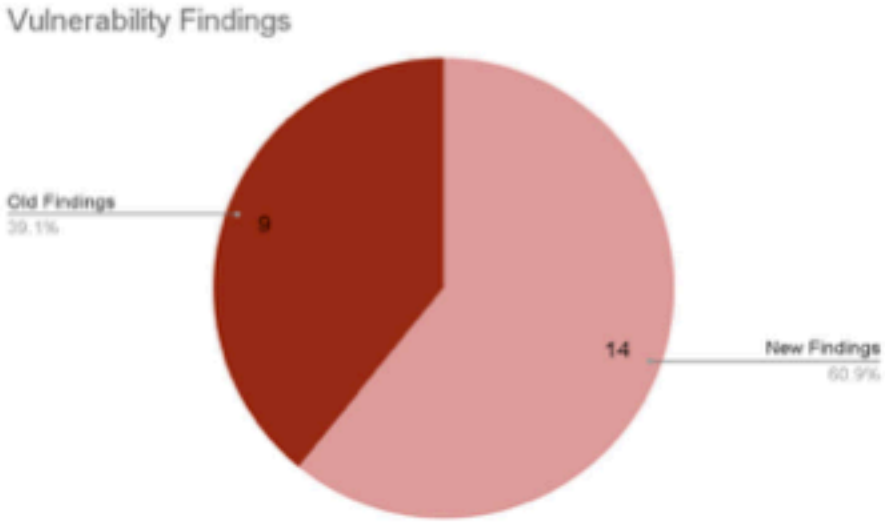
Phases of testing adapted from NIST 800 - 115

The approach began with the team conducting open-source research prior to starting the engagement on Le BonBon Croissant's network and forming a plan of attack based upon information discovered. When the engagement began on the network, the team began a discovery phase which had the goal of gathering information and mapping the network of Le BonBon Croissant. When information was gathered, the team proceeded to the attack phase based on the information gathered. The discovery and attack phases were repeated as more information was collected and leveraged for exploitation. All results were logged from the attack and planning phase to be compiled during the final reporting phase. The report phase consolidated all of the information into a report for Le BonBon Croissant.

Improvements

From the previous assessment conducted in November of 2021, several improvements were implemented to LBC's network. The most noted improvements were the removal of previous vulnerable machines, encryption of network communications, and the removal of vulnerable services.





Vulnerable Machines

Previously, LBC's network hosted a vulnerable windows server running VNC, a remote desktop service. Since the previous assessment this machine has been removed along with any instances of VNC. This is a substantial improvement as the machine was previously an initial foothold for the assessment team into the network. Without the machine present, LBC has removed an easy way for attackers to gain initial access to the network and collect credentials to move laterally.

Network Communications

During the assessment in November, it was discovered that a majority of network communications were operating using HTTP rather than HTTPS. This had the effect of allowing the communications to continue in cleartext leaving them vulnerable to sniffing. Now, LBC's web services redirect to HTTPS and make use of SSL encryption. This has reduced the likelihood of traffic being eavesdropped on substantially.

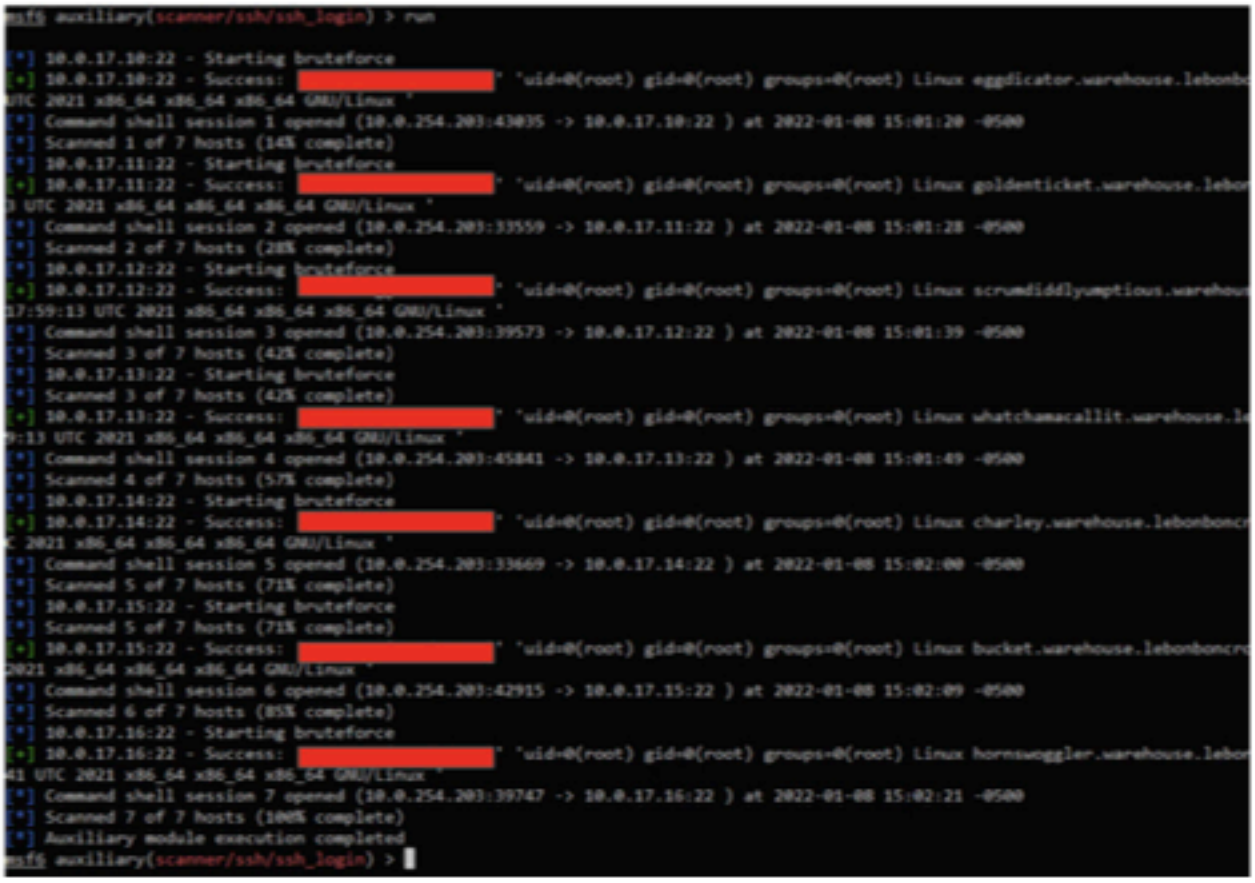
Vulnerable Services

Finally, during the November assessment a number of vulnerable web services were found running throughout the LBC network. For example, a vulnerable version of OpenCart was found running previously that left users susceptible to a CSRF attack. Now, these services have been replaced by the development of the new e-commerce site and APIs, which do not leave customers as exposed as the previous development.

Attack Narrative

1. SSH with known password

After performing initial reconnaissance it was found that open SSH services were running on several hosts within the network. The team attempted to log into the hosts using a list of passwords generated from previous engagements and leaked passwords on the internet. The team found that one password within the list allowed the team to gain a session on every host within the system given the initial scope. This was the initial foothold of the attack and the team continued onto the host at the IP 10.0.17.10 for the rest of the attack.



2. Review source on box to find hard coded and environment secrets

Within the host at 10.0.17.10 a docker container was found running a payment service for LBC. The team was able to enter the container and examine the source code of the application. Within the source code, credentials for the service were found hard coded in clear text. These credentials were recorded for later use. Additionally, environment variables were examined for other secrets and recorded.

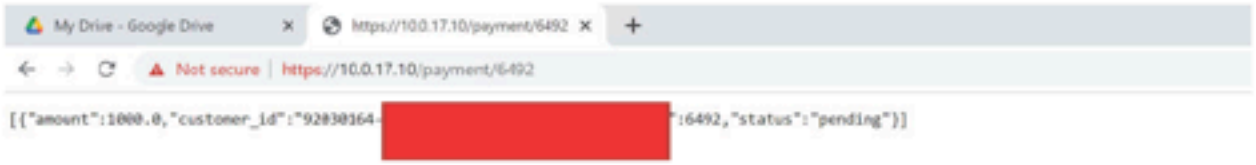
```
def get_current_username(credentials: HTTPBasicCredentials = Depends(security)):
    correct_username = secrets.compare_digest(credentials.username, [redacted]
    correct_password = secrets.compare_digest(credentials.password, [redacted]
    if not (correct_username and correct_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect email or password",
            headers={"WWW-Authenticate": "Basic"},
        )
    return credentials.username
```

3. Authenticate to services

Using the credentials found in the source code, the team authenticated to the service hosted on 10.0.17.10. It was determined that the application managed payments and transactions for LBC. The team then began testing the functionality of the payment system.

4. Manipulate Payments

After examining the API source code hosted on 10.0.17.10, [redacted] was able to identify multiple security vulnerabilities. Manipulation of payment transactions was the most significant vulnerability that was discovered. [redacted] was able to make post requests to the /payment API and change the amount of money a customer owed. Furthermore, [redacted] was able to artificially create new transactions for customers. Furthermore, [redacted] learned the other API function calls and was able to retrieve and view other customer data without authentication.



5. Gain further access to databases

From investigating the API and the environment variables within the docker container, [redacted] determined that a database was being used as well to store some of the information for the payment system. Using credentials previously obtained, [redacted] moved to the host at 10.0.17.14 to then find a postgres database.

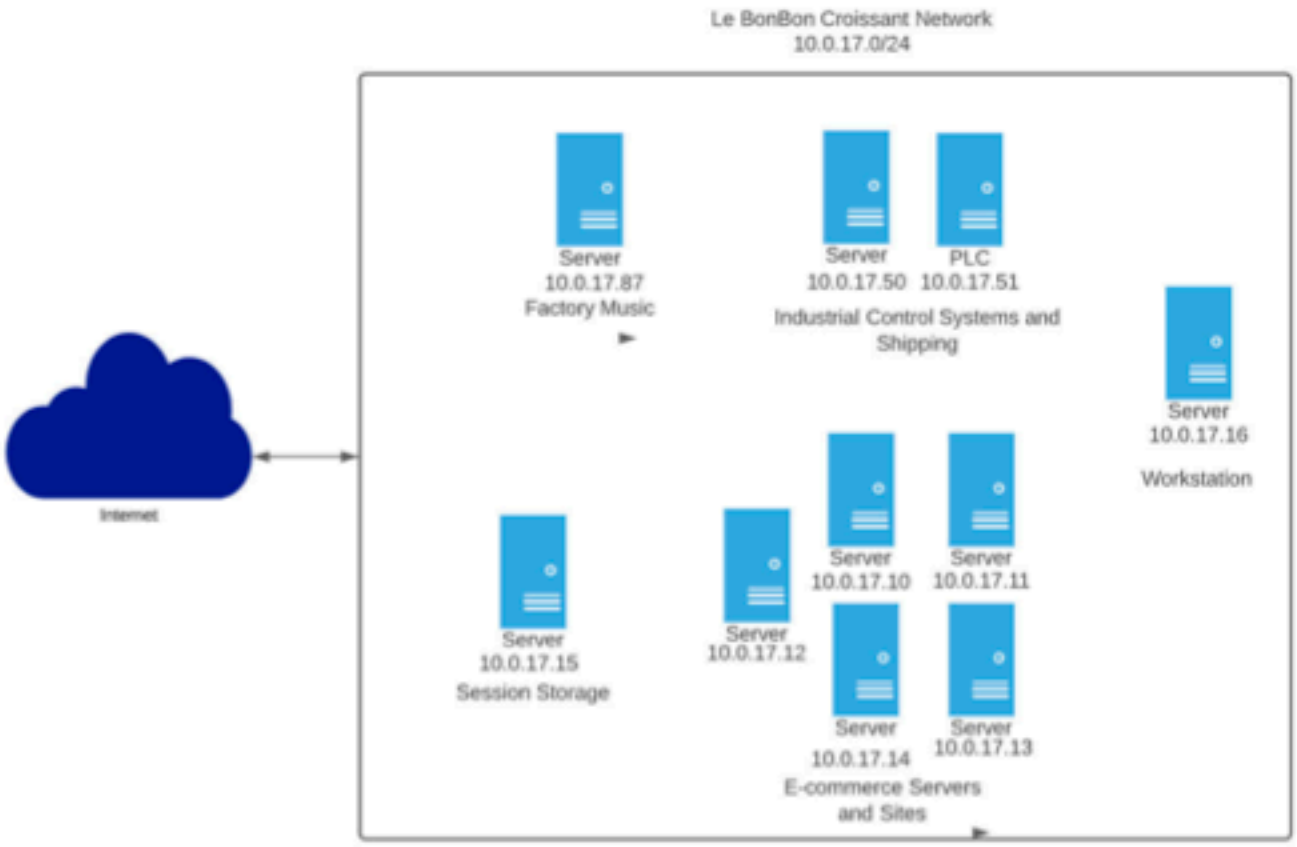
```
root@f9043e7bb8e7:/api# printenv
POSTGRESQL_USERNAME=postgres
JAWBREAKER_DB_APP_NAME=jawbreaker-db
HOSTNAME=f9043e7bb8e7
JAWBREAKER_DB_DATABASE=jawbreaker
PWD=/api
OS_FLAVOUR=debian-10
JAWBREAKER_DB_PORT=5432
POSTGRESQL_DATABASE=jawbreaker
JAWBREAKER_APP_HOST=jawbreaker-api
HOME=/root
JAWBREAKER_APP_NAME=
JAWBREAKER_DB_USER=postgres
TERM=xterm
BITNAMI_IMAGE_VERSION=3.7.12-debian-10-r125
JAWBREAKER_DB_PASSWORD=[redacted]
SHLVL=1
BITNAMI_APP_NAME=jawbreaker
POSTGRESQL_PASSWORD=[redacted]
JAWBREAKER_DB_HOST=charley.warehouse.lebonboncroissant.com
OS_NAME=linux
PATH=/opt/bitnami/python/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
JAWBREAKER_DB_ROOT_PASSWORD=[redacted]
JAWBREAKER_APP_PORT=8000
JAWBREAKER_DB_ROOT_USER=root
OS_ARCH=amd64
_=/usr/bin/printenv
```

6. Gain tokens and customer credit cards

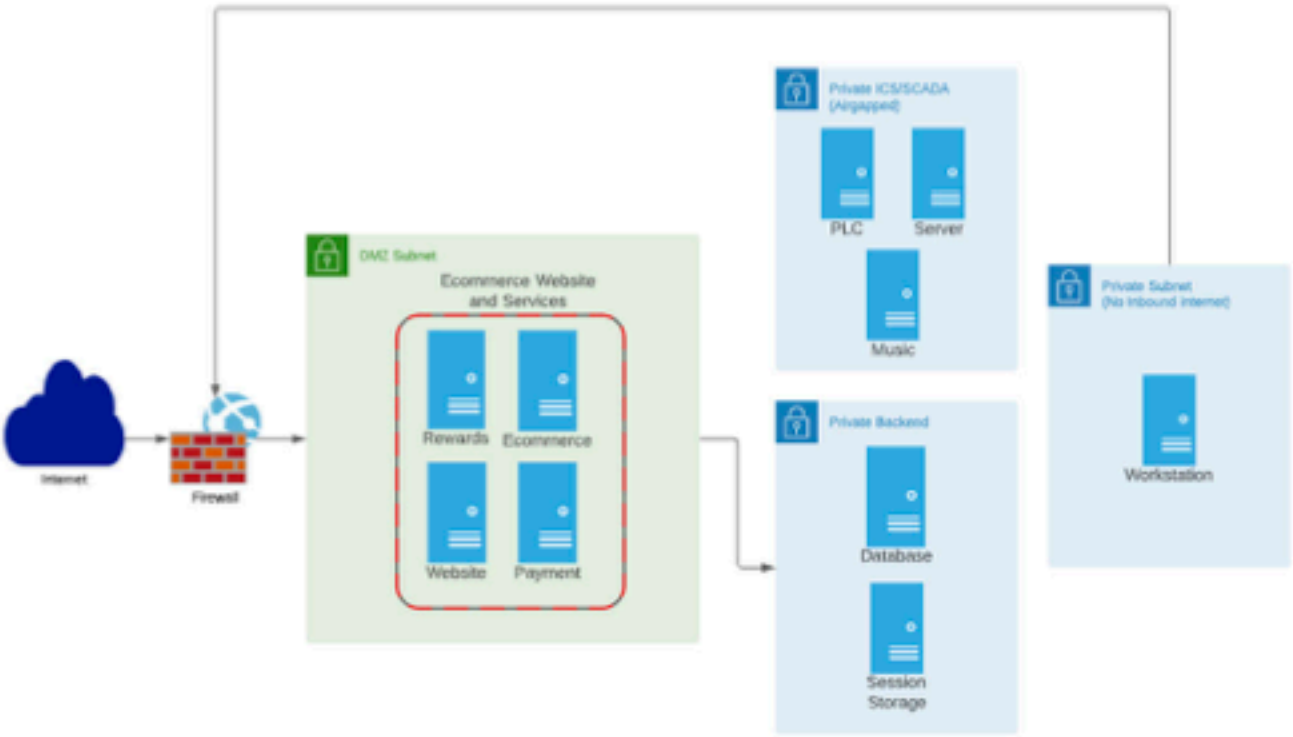
The postgres database could be dumped without authentication despite having the password to the database. The database was revealed to have customer credit card information. At this point [redacted] also browsed to a hidden endpoint in the system which revealed access tokens. With the combination of

access tokens, customer data, and having a root session on nearly every host in the system, [redacted] concluded their main attack scenario.

Network Map


Le BonBon Croissant Network
10.0.17.0/24
Server
10.0.17.87
Factory Music
Server
10.0.17.50
PLC
10.0.17.51
Industrial Control Systems and
Shipping
Server
10.0.17.16
Workstation
Internet
Server
10.0.17.15
Session Storage
Server
10.0.17.12
Server
10.0.17.10
Server
10.0.17.11
Server
10.0.17.14
Server
10.0.17.13
E-commerce Servers
and Sites

Proposed Network Map



[redacted] proposes that LBC adopt the above logical network map. Much of the risk discovered could have been reduced if LBCs network was properly segmented. The above diagram represents some basic network segmentation that LBC can immediately implement to improve its security posture.

First, a DMZ subnet is a public-facing subnet that hosts publicly facing applications, such as LBCs website. Hosting publicly facing applications in a separate zone, allows LBC to mitigate the damage if a host in the zone is compromised. Hosts in the DMZ have limited access and communication to other LBC hosts mitigating the impact of a breach.

Second, the private backend adds additional security to LBCs publicly facing applications. These services such as the website database and session storage can only be accessed in a limited capacity by the applications in the DMZ. Therefore, it is harder to compromise the services hosted in the private backend in the event of a breach. This segmentation prevents malicious users from being able to directly access or attack the backend services.

Third, a private subnet for employee workstations improves LBCs security posture by only allowing outbound internet communications over a stateful firewall. This prevents malicious actors from being able to directly access or attack employee workstations.

Lastly, the SCADA/ICS network is private and air-gapped. As discussed with LBC, the PLC and ICS devices are extremely fragile. If these devices go down LBC will suffer catastrophic business impact. Since these

devices do not need internet communication to function in the warehouse, they should be air gapped. This will prevent any external or extraneous communication to the devices. Therefore, it would not be possible to compromise or disrupt these devices over the internet. Air gapping these devices will tremendously improve LBC's security posture.

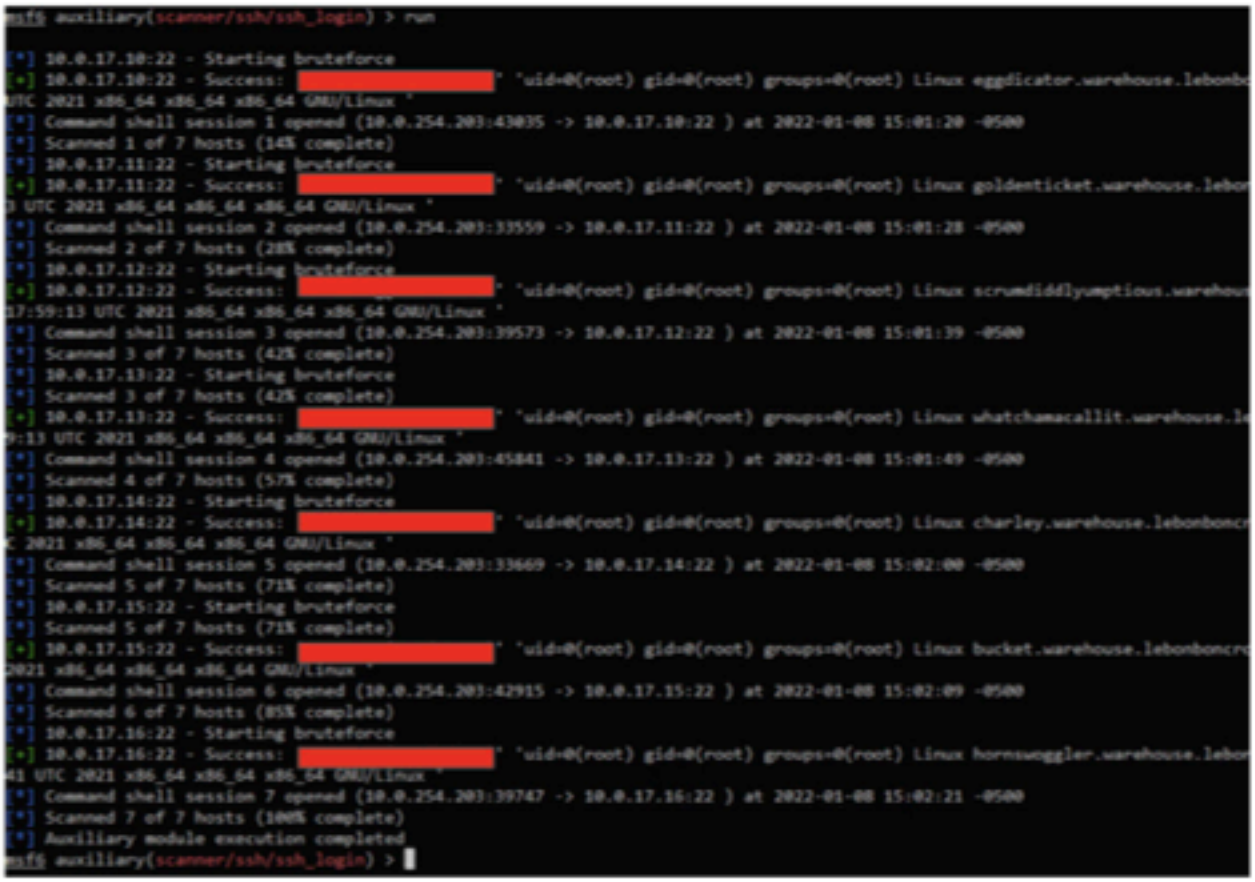
Technical Findings

Severe

Credential Reuse
Description
Hosts: 10.0.17.10-16, 10.0.17.87 These hosts used the same SSH password for the `root` account. As a result, an attacker that is able to compromise one host will be able to compromise all of the other hosts.
Impact
Business Impact Score: 25 **CVSS:** 9.8 **Explanation of Business Impact:** This is a severe vulnerability because a malicious actor could acquire one password and use it to comprise almost all of LBC's network. This has a high chance of occurring and execution of this attack would lead to catastrophic financial and reputational damage to LBC. An attacker could use this vulnerability to violate the confidentiality, integrity, and availability of LBC's entire business.
Tactical Remediation
As a tactical remediation, the SSH passwords for the affected hosts should be changed and ensured they are unique.
Strategic Remediation
In the long term, a password policy and password best practices training should be provided in order to prevent credentials from being reused on different hosts.
Short Term Remediation
In the short term, an SSH lockout policy should be implemented to make it more difficult for an attacker to discover passwords until the passwords can be made unique.

Reproduction:

Metasploit can be used to show all SSH servers have the same password:



Manipulation of Payment Transactions
Description:
Host: 10.0.17.10 The hosts contained an API that allowed unauthenticated users to insert or change payment data. This data contained the payment invoice and information. Therefore, any user is able to change the amount of data that a customer owes to LBC.
Impact
Business Impact Score: 25 **CVSS**: 9.1 **Explanation of Business Impact:** This is rated as a severe vulnerability because it presents immediate financial and reputational damage to LBC. Any user is able to arbitrarily change or add payment transactions to LBCs system. This means that a user could increase or decrease the amount of money that a customer pays LBC. This could lead to LBC overcharging or undercharging customers. These issues could lead to customers taking legal action against LBC.
Tactical Remediation
The payment transaction API should be rewritten such that only authentication and authorized employees from LBC can access the payment APIs.
Strategic Remediation
Implement a company wide audit program to validate transactions and ensure the security of payment processes. Audits should be performed on a quarterly basis focusing on payment security. Unit testing should be implemented as part of the CI/CD development, including tests that check if a user token is able to access information for another customer. Developers should also receive training on developing secure web applications.
Short Term Remediation
Immediately redact access so that the API can only be accessed from within the LBC internal network.

Reproduction:

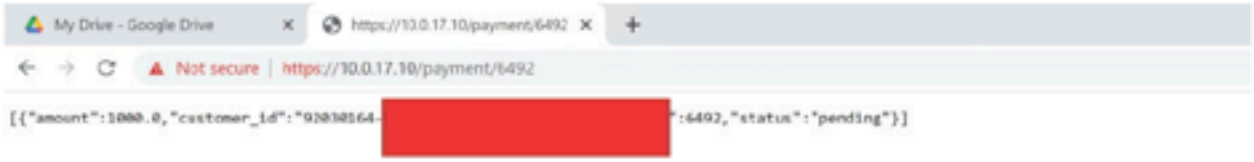
The source code of the API on the host 10.0.17.10 showed that any user could edit transactions by making a POST request:

```
@api.route('/payment', methods=['POST'])
def make_payment():
    content = request.get_json(silent=True)
    if content['customer_id'] is None:
        return "customer_id is required!", 400
    if not check_uuid(content['customer_id']):
        return "invalid customer_id", 400
    if content['amount'] is None:
        return "amount is required!", 400

    payment_data = {
        "customer_id": content['customer_id'],
        "amount": content['amount'],
        "status": "pending"
    }
```

Then a malicious actor can edit a previous transaction or make a new one. In the screenshot below, there is an empty transaction. The transaction id is 6492:

Using the source code pictured above, a malicious user can craft a post request to create a false transaction in the database as shown below. In this example, a transaction of 1000 is created for a customer:

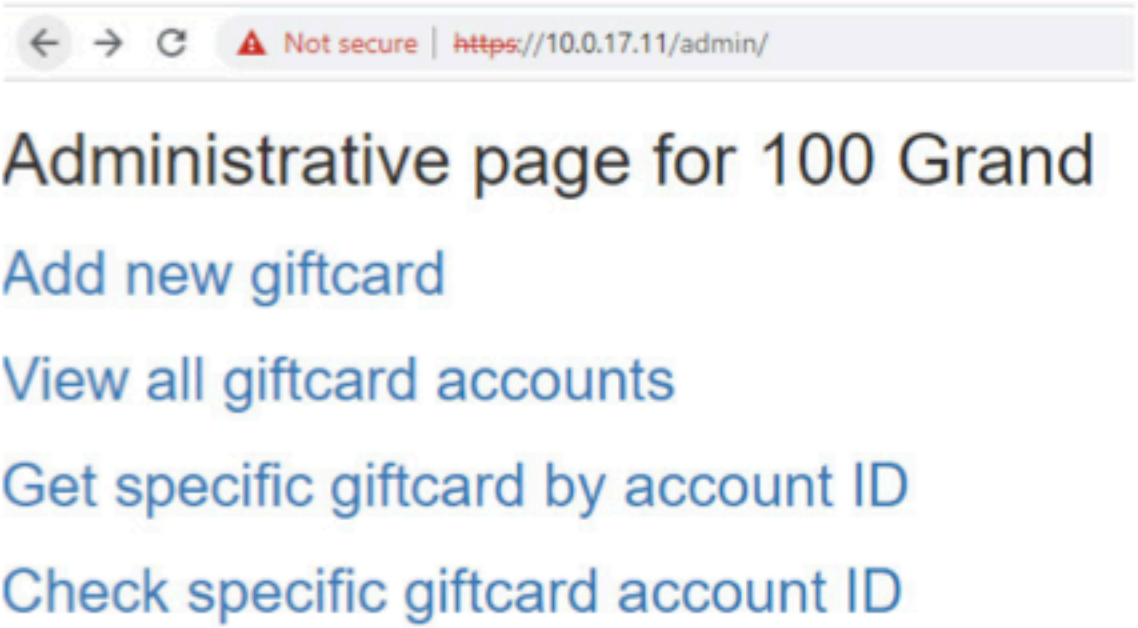
After making the post request to the `/payment` API and refreshing the transaction page for the transaction ID 6492, one can see that the transaction details sent in the post request now appear in the database.



Gift Card Manipulation
Description
Host: 10.0.17.11 The administrator service on the reward program does not have proper controls for adding and authenticating a gift card. Although only administrators can access the UI, there is no authentication required when using the `/add` endpoint.
Impact
Business Impact Score: 20 **CVSS**: 7.1 **Explanation of Business Impact:** The creation of arbitrary gift cards poses a large threat to Le BonBon Croissant. The arbitrary amount of gift cards can massively detract from Le BonBon Croissant's profits as they can result in free products to customers. Additionally, it can artificially create payments due through customers having negative values. This could result in customer complaints and customer loss.
Tactical Remediation
Require authentication for all endpoints on the gift card API.
Strategic Remediation
Unit tests in the CI/CD pipeline should test whether endpoints can be accessed with no or improper credentials.
Short Term Remediation
Take down the API service and manually edit the gift card database until proper authentication is added.

Reproduction:

When browsing to 10.0.17.11 after supplying admin credentials, there are options within a menu to add a gift card:

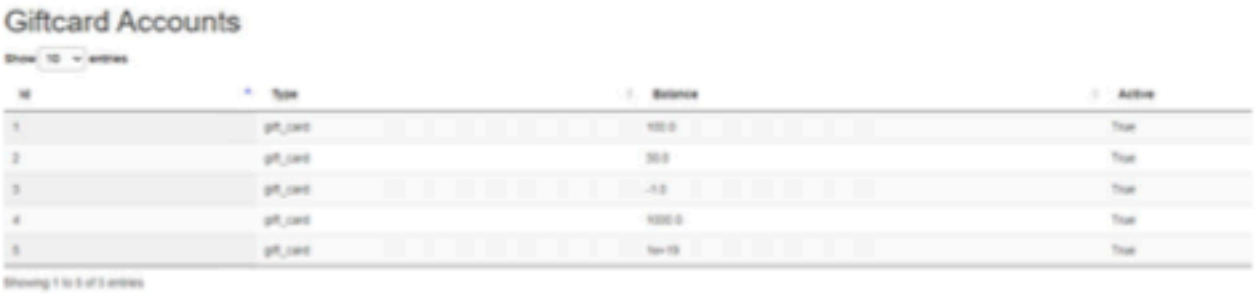


It then brings you to a screen that allows a new gift card to be added by supplying the account number, balance for the gift card, and the customer's name:

Once the information is completed and the submit button is clicked then the gift card is created. However, the `main.py` source code shows that the `/add` endpoint this form uses does not require authentication:

```
@app.get("/add/")
async def rewards(account: int, balance: float, account_type: str, first="None", last="None", card=0,active=1,test=0):
    db = next(get_db())
    dbdat = crud.add_account(db, account, balance, account_type, test, active, card, first, last)
```

[redacted] was able to create several gift cards with erroneous information which could then be viewed in an account summary section:



The `/accounts` endpoint is also available to unauthenticated users:

```
└─# curl https://10.0.17.11/accounts/ -k
[{"id": 1, "accnum": "2", "type": "gift_card", "balance": 1
odified": "2022-01-07", "is_test": false, "is_active": true
", "type": "gift_card", "balance": 1000.0, "date_created":
", "is_test": false, "is_active": true, "is_card": false}]
```

SSH Authentication Misconfiguration
Description:
Hosts: 10.0.17.10-16, 10.0.17.87 All SSH server configurations allow for root and password logins. root login allows for remote attackers to gain full access to a server. Password authentication allows for brute force attempts and the possibility for a rogue SSH server to steal passwords.
Impact
Business Impact Score: 20 **CVSS**: 8.8 **Explanation of Business Impact:** This is a severe finding because it could cause a disruption of LBC's entire business. With root access enabled on almost every host in the network, a malicious actor would be able to take any action they wanted on LBC's business and network.
Tactical Remediation
The team advises that the `sshd_config` be modified to disallow `root` and password logins. Only non-root accounts should be able to SSH login via public key authentication.
Strategic Remediation
The team advises that configurations for public facing services be periodically reviewed and compared to examples of secure configurations.
Short Term Remediation
The SSH server should be disabled if the configuration is not updated.

Reproduction:

The team was able to authenticate into any SSH server using the `root` account and a previously found password:

```
ssh root@10.0.17.10
```

```
root@10.0.17.10's password: [REDACTED]
```

This is because `/etc/ssh/sshd_config` contains the following lines:

```
PermitRootLogin yes

PasswordAuthentication yes
```

In a default configuration, both of these values are set to no.

Hardcoded Credentials
Description
Hosts: 10.0.17.10-11 The services running on the affected hosts use credentials that are hardcoded in clear text for authentication. This results in the passwords being easy to recover if an attacker gains access to the source code of the services. Passwords should also be changed periodically, and hardcoding credentials does not easily allow this.
Impact
Business Impact Score: 20 **CVSS**: 9.1 **Explanation of Business Impact:** The services that make use of hardcoded, cleartext credentials are related to the payment and rewards management programs. If an attacker recovers the credentials they have control over both of these systems. This allows them to create false payments, award gift cards without payment, and clear payments. Le BonBon Croissant could face lawsuits resulting from false payments or have profits cut by fraudulent transactions resulting in massive revenue loss.
Tactical Remediation
Change the passwords so that they are stored as a hash value that is compared to another that is not viewable by cleartext. Additionally, passwords should not be hardcoded so they can more easily be changed.
Strategic Remediation
Integrate credential management best practices into the development cycle and make sure it is reviewed before deploying services.
Short Term Remediation
Change the passwords on the hosts containing these services so the source code cannot be viewed.

Reproduction:

By browsing the source code, the credentials to services and applications could be found. Once an attacker was authenticated to a machine and entered into the docker container using `docker exec -it <docker ID> bash`, then the source could be viewed by printing out the file with the `cat` command. There was no other obstacle to viewing the credentials as they were in cleartext, hard coded within the programs.

`api.py` on 10.0.17.10 contains hardcoded admin credentials:

```
users = {
    "admin": generate_password_hash("[redacted]"),
    "jimbo": generate_password_hash("[redacted]"),
    "test": generate_password_hash("[redacted]")
}
```

As well as `main.py` on 10.0.17.11:

```
def get_current_username(credentials: HTTPBasicCredentials = Depends(security)):
    correct_username = secrets.compare_digest(credentials.username, "[redacted]")
    correct_password = secrets.compare_digest(credentials.password, "[redacted]")
```

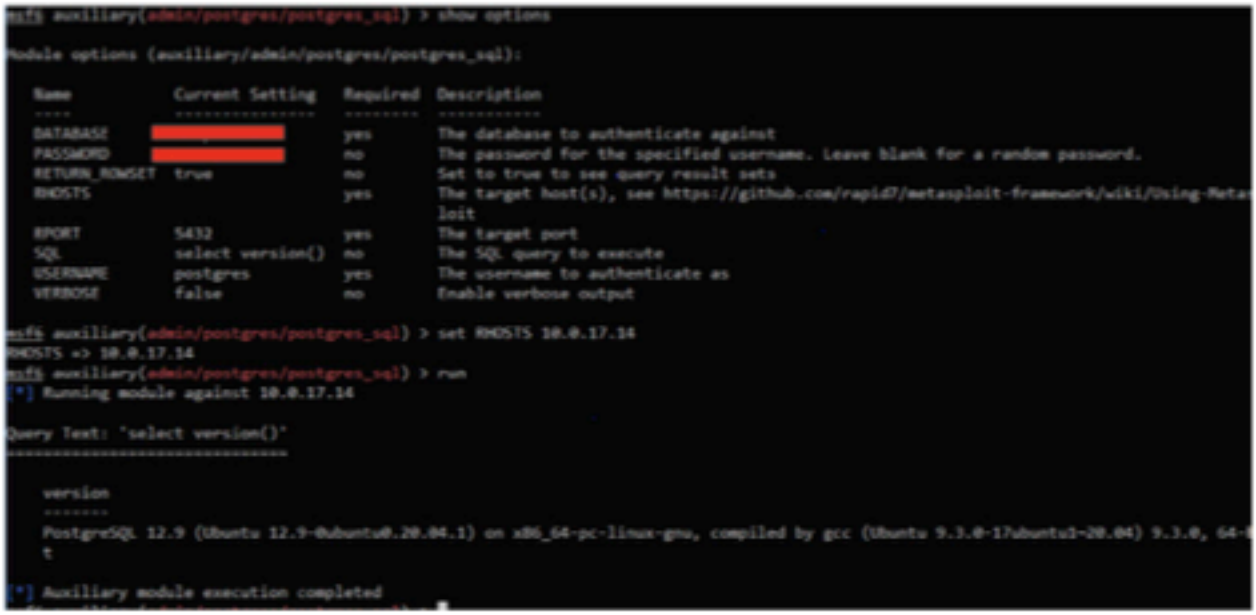
User Password Management
Description:
Hosts: 10.0.17.10-11, 10.0.17.14 The hosts contained weak passwords that would be easily guessable by a malicious actor. The passwords seen on these hosts are a part of common wordlists that attackers use to brute-force their way into systems.
Impact
Business Impact Score: 20 **CVSS**: 9.1 **Explanation of Business Impact:** The passwords protecting critical services on the affected hosts were easily guessable and found in common wordlists. This can lead to severe reputational and financial damage as the services control payment systems, reward programs, and contain customer data.
Tactical Remediation
Use secure password generation (password managers) such as LastPass or 1Pass to have users create and manage their passwords.
Strategic Remediation
Implement a robust password policy that requires users to meet PCI-DSS Password standards. For a complete password policy, [redacted] suggests that passwords meet the recommendations from Cylab which states that users should have passwords that are a minimum of twelve characters in length to ensure a balance between usability and security. Additionitially, implementing multi-factor authentication is recommended as a best practice whenever possible.
Short Term Remediation
Immediately change all passwords that do not meet the PCI-DSS password standards to passwords that do meet the PCI-DSS password standards.

Reproduction (10.0.17.10-11):

The hardcoded passwords on both of these APIs are easily guessable. They allow for administrator access.

Reproduction (10.0.17.14):

The PostgreSQL server could be accessed with a guessable password as can be seen in the screenshot below:



API Token Management
Description
Hosts: 10.0.17.12-13 The ecommerce site poorly implements API tokens, including not limiting access by user, hardcoding tokens, and hardcoding the expiration for tokens.
Impact
Business Impact Score: 20 **CVSS**: 8.3 **Explanation of Business Impact:** This vulnerability presents immediate financial and reputational damage to LBC. Usage of these tokens allow any user to edit inventory and view all other customers' data. This violates PCI-DSS and presents both legal and financial consequences to LBC. PCI-DSS penalties can be up to $500,000 per incident in addition to other remediation costs. Furthermore, modification, inaccuracy, and potentially defamation of LBC's publicly facing inventory and website can lead to a loss of customers.
Tactical Remediation
The ecommerce and client API should be rewritten such that user tokens can only access information about themselves, only admin tokens can modify the inventory, the client uses a different token for each user, and the token expiration is dynamic.
Strategic Remediation
Unit testing should be implemented as part of the CI/CD development, including tests that check if a user token is able to access information for another customer. Developers should also receive training on developing secure web applications.
Short Term Remediation
Fixing this issue cannot be completed in the short term because it requires significantly modifying the client and API code. In the meantime, the website should not be open to customers.

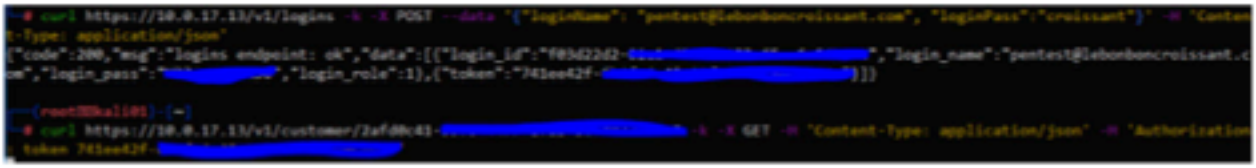
Reproduction:

The source code for the API client at 10.0.17.12 shows a hardcoded global API token that is used for all requests:

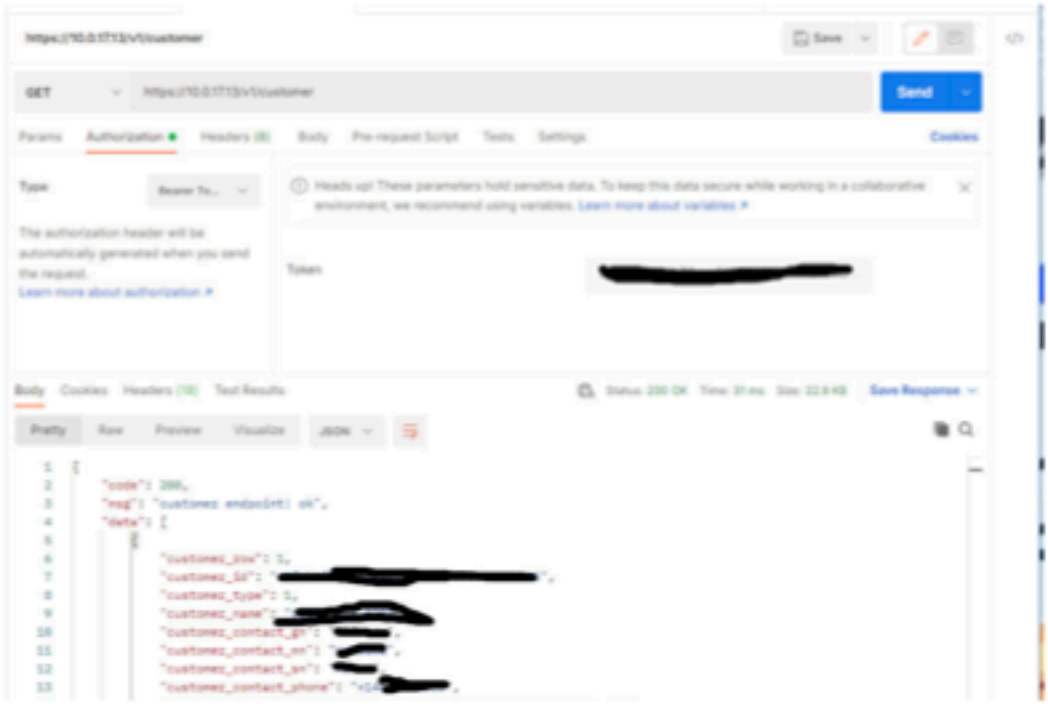
```
const WmciApiHeaders = {
  'Authorization': `token ${WmciApiKey}`,
  'Content-Type': 'application/json'
};
```

The client should save the token once a user logs in and use that for subsequent requests, since each user should have permission to access different resources.

Along the same lines, the source code for the API at 10.0.17.13 shows that, as long as a token is valid, it is valid for any request. For example, sending a request to `/v1/customer/id`, where `id` is a customer ID, will always return customer data, including their address, even if the token belongs to a different user:



Even more concerning, any token can be used for the `/v1/customer` endpoint, which retrieves information for all customers:



There should also be a differentiation between user and admin API tokens, since currently any token can be used to modify the inventory by sending POST, PUT, or DELETE requests to `/v1/inventory`.

Finally, when a user logs in (`/v1/logins`), a new token is added to the database. However, the same expiration is used for every token:

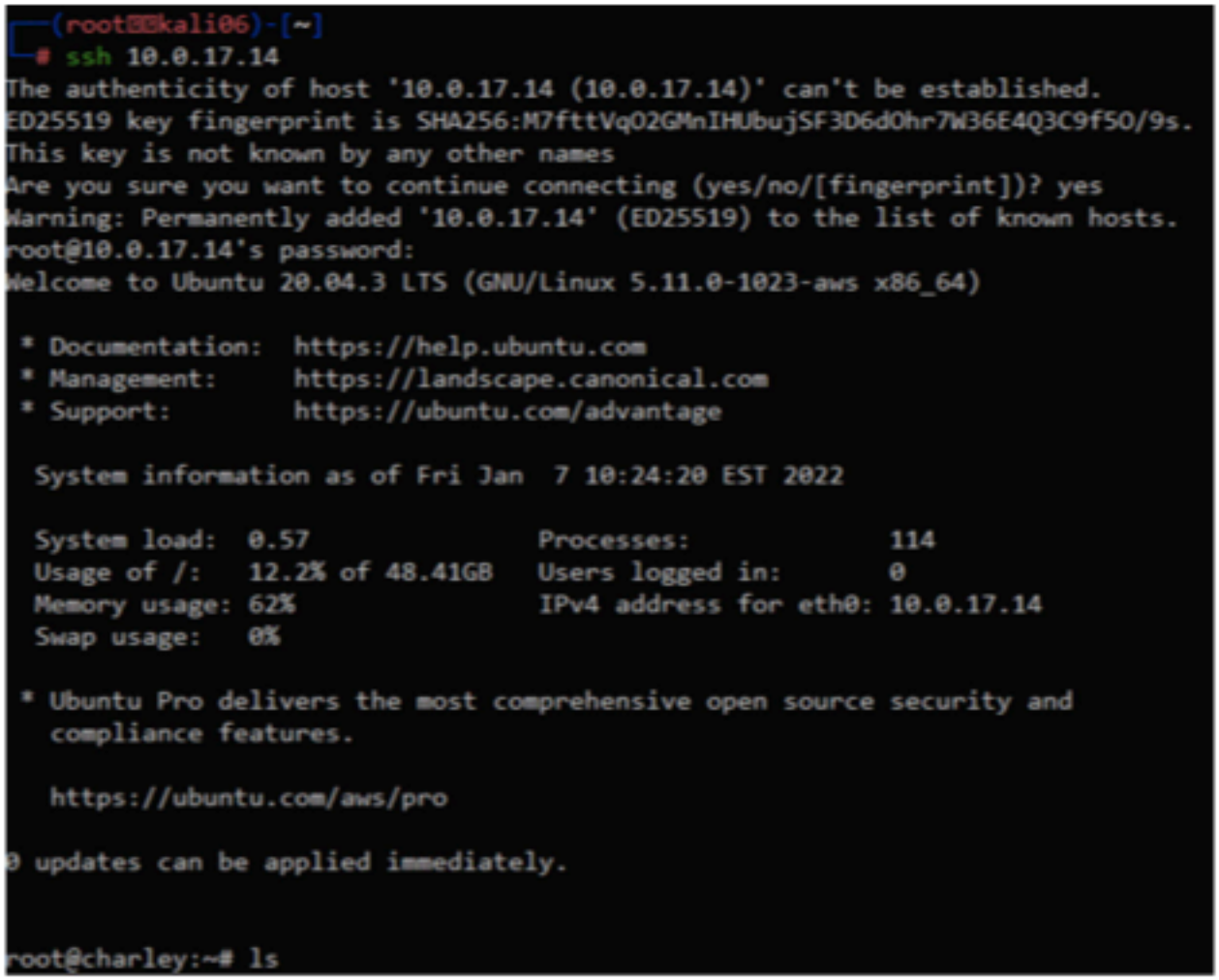
```
INSERT INTO tokens (token, customer_id, expires, allowed_methods)
VALUES (?, ?, "2022-10-10 11:09:02", "GET,POST,PUT,DELETE")
```

Additionally, the `isAuthN` middleware only checks if a token exists, not if a token has expired. Tokens should be short lived since they may be compromised by an attacker.

Restrict Administrator Account Usage
Description
Hosts: 10.0.17.10-16, 10.0.17.87 The above hosts used their administrator or root accounts by default. There were no other users found or separation of privileges found within the hosts and thus a compromise of the system instantly resulted in administrator privileges.
Impact
Business Impact Score: 20 **CVSS:** 7.8 **Explanation of Business Impact:** The lack of separation of privileges means that if an attacker gained access to any host they had the full ability to compromise business operations that the host performed. This could include deleting customer data, halting of operations, or exfiltrating and altering confidential data.
Tactical Remediation
Create a default user for login that does not have root access to the system by default for use in day-to-day operations.
Strategic Remediation
Implement a comprehensive access control policy organization wide based upon the least amount of privileges required for each user to complete the work on that specific system.
Short Term Remediation
Update the administrator passwords so that each is different.

Reproduction:

```
ssh root@{Affected IP}
```



Unencrypted Credit Card Information
Description
Hosts: 10.0.17.14 The host contained unencrypted credit card information in its database. This information is considered sensitive, and if leaked to the wrong hands, could result in damage to the reputation of the company leading to potential financial harm.
Impact
Business Impact Score: 20 **CVSS**: 7.5 **Explanation of Business Impact:** Unencrypted storage of credit card information is a major violation of the PCI-DSS standards. This vulnerability makes LBC susceptible to legal and financial consequences. Currently, PCI-DSS penalties are up to $500,000 per incident in addition to other remediation costs such as credit monitoring for affected users.
Tactical Remediation
Sensitive customer information should be encrypted with a secure algorithm such as AES-256 and only be decrypted when it is needed. Then if a data breach occurs, attackers cannot read the information without another secret key.
Strategic Remediation
Developers should receive training on what data is considered sensitive and should not be stored in cleartext.
Short Term Remediation
All database and API credentials should be changed to prevent access to attackers. Database logs should also be monitored in case of unauthorized access.

Reproduction:

After a thorough scan of the host, it was noticeable that a PostgreSQL database server was running. Accessing the data on the server was done by connecting to the host 10.0.17.14 and using the command `sudo -u postgres pg_dumpall < dump.sql`, which exported all of the databases to a file.

The table structure that contains the credit card information of clients can be seen below. The actual credentials are not included in this report for security reasons.

```
CREATE TABLE billing.credit_cards (
    id integer NOT NULL,
    name character varying(100) NOT NULL,
    number character varying(19) NOT NULL,
    expiration character varying(5) NOT NULL,
    ccv character varying(4) NOT NULL,
    zip character varying(5) NOT NULL
);
```

PostgreSQL RCE
Description:
Hosts: 10.0.17.14 The postgres service is running with weak credentials for the user `postgres`. Moreover, the `postgres` user has the superuser flag turned on.
Impact:
Business Impact Score: 20 **CVSS**: 8.3 **Explanation of Business Impact:** An attacker can completely compromise the system and gain control over the server. This allows the attacker to shutdown business critical services, steal customer data and compromise other applications on the network.
Tactical Remediation:
Change the credentials for the `postgres` user to a more complex one that can not be guessed by the attacker.
Strategic Remediation:
Add strict account management policies such as account lockout and password complexity requirements.
Short Term Remediation:
Remove the superuser privileges for the `postgres` user. These privileges are not required for the functioning of the applications.

Reproduction:

Postgresql service can be accessed using the command psql with default username and password.

```
# psql -h 10.0.17.14 -U postgres -W
```

The service version can be found using the following query.

```
> select version();
```

The postgres user has admin privileges that can be verified below:

```
> show is_superuser;
```

To exploit this misconfiguration, the pgexec exploit can be used.

To correctly compile the exploit, Docker images can be used. Since the server uses PostgreSQL v12, below commands can be used to correctly compile the exploit.

```
host# docker pull postgres:12

host# docker run -it postgres:12 bash

docker# apt update

docker# apt install postgresql-server-dev-12 git gcc

docker# git clone https://github.com/Dionach/pgexec/

docker# ./pg_exec.sh -U postgres -P <password> -h 10.0.17.14 -c
        "<arbitrary command to execute on server>"
```

The below screenshot show this functionality and successful reverse shell from the server:

PLC Exposed without Authentication
Description:
Hosts: 10.0.17.51 Sensitive information can be retrieved and altered on a critical PLC without any authentication.
Impact:
Business Impact Score: **CVSS**: **Explanation of Business Impact:** An attacker can completely compromise the system and gain control over the server. This allows the attacker to shutdown business critical services, steal customer data and compromise other applications on the network.
Tactical Remediation:
The PLC software should be altered to require sufficiently complex credentials to send any commands.
Strategic Remediation:
Custom developed embedded systems should be reviewed by a 3rd party to ensure it has proper security, such as authentication.
Short Term Remediation:
The PLC should not be accessible to the public internet, and in cases for remote management, a VPN server can be setup.

Reproduction:

One can connect to the PLC remotely with the following `netcat` command:

```
nc 10.0.17.51 2001
```

Sending a `?` command results in the following output:

```
G000,0000 TO RETREIVE VALUES FROM THE NETWORK

S000,0000,0000 TO SET VALUES IN THE NETWORK,

{DEVICE},{MEMID},[{VALUE}]

Note to Rick: Update Help Text to include more details and failure
modes and fix that memory leak!
```

This reveals sensitive information, such as the command format and a potential vulnerability in the PLC. Multiple G commands were used to attempt to find valid device IDs. Given more time, an attacker could find valid device and memory IDs, then start to change values.

High

Insecure Password Storage
Description
Hosts: 10.0.17.14 The MariaDB server stores user passwords as Base64, allowing for an attacker to easily recover the plaintext.
Impact
Business Impact Score: 16 **CVSS**: 7.5 **Explanation of Business Impact:** Unencrypted storage of user credentials presents both financial and reputational damage to LBC. Unencrypted storage of credentials can be seen as security negligence which would expose LBC to legal consequences. Furthermore, users may choose not to use LBC if they find out that their passwords are stored unencrypted.
Tactical Remediation
Passwords should be salted and hashed in a database, using a secure algorithm such as SHA-256.
Strategic Remediation
Developers should receive training on how to properly implement authentication and store sensitive information.
Short Term Remediation
All database and API credentials should be changed to prevent access to attackers. Database logs should also be monitored in case of unauthorized access.

Reproduction:

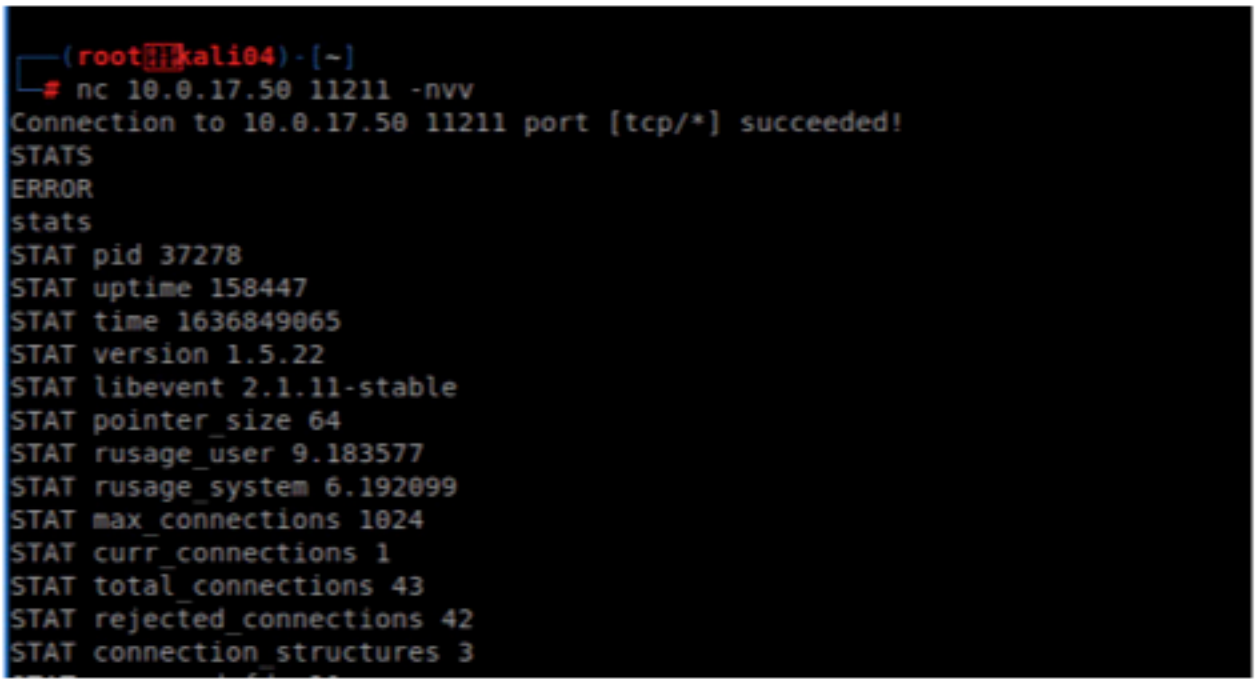
Querying the `wmci.logins` table shows that the information in the `login_pass` column is Base64 encoded:

A Base64 decoder can be used to recover the password plaintext without a key.

Memcached Authentication
Description
Host: 10.0.17.15 The key value datastore service memcached running on the port 11211 allows any remote user to execute commands without authentication.
Impact
Business Impact Score: 16 **CVSS:** 6.5 **Explanation of Business Impact:** Customer data and other sensitive credentials saved using the service may be exposed to the attacker. The attacker may also be able to modify the sensitive information and cause disruption of the services. This exploit leads to customer data being affected, which will cause major financial and reputational damage to LBC.
Tactical Remediation
At the time of the assessment, no data was stored using the service. The service can be disabled if that is not required.
Strategic Remediation
Memcached supports authenticated usage. The team recommends using the service with authentication.
Short Term Remediation
If the service can not immediately start using authentication, a firewall should be used in order to isolate the service communication and reduce exposure.

Reproduction:

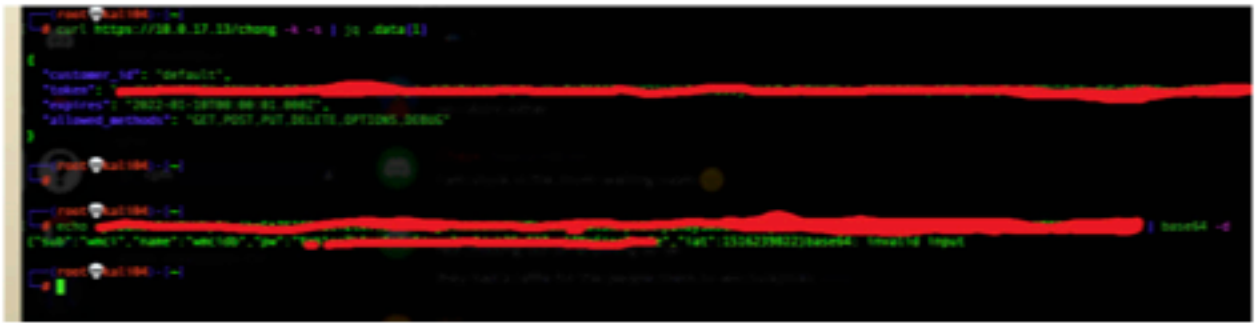
Connect to the memcached server using `netcat` on the port 11211. Arbitrary memcached commands can be executed over the socket as shown in the below screenshot. The below screenshot shows the execution of the "`stats`" command.



API Tokens Leak
Description
Host: 10.0.17.13 The ecommerce API endpoint `/chong` lists available tokens from the database. These tokens can be used to make authenticated requests against the API. No authentication is required to access the `/chong` endpoint. Moreover, the Base64 encoded token contains the plaintext password for a MySQL account. This leads to further compromise of other services potentially resulting in lateral movement.
Impact
Business Impact Score: 15 **CVSS**: 6.5 **Explanation of Business Impact:** The backdoor allows for information to be gathered that has a direct impact on LBC's business operations. The tokens allow for the manipulation of payment, inventory, and stealing customer information. These incidents could result in legal action against LBC from customers as well as the PCI-DSS standards which have penalties up to $500,000 for customer data being compromised in addition to other remediation costs. The reputational and financial fallout could gravely harm LBC's revenue if the leaked information is used maliciously to disrupt operations.
Tactical Remediation
Endpoints must not reveal tokens. Disable this endpoint and implement appropriate authentication service which accepts user credentials and issues tokens.
Strategic Remediation
Implement a company wide policy for securely storing secrets and setting an appropriate expiration time period on inuse tokens.

Reproduction:

Visit the `/chong` endpoint and obtain a list of tokens. No authentication is required to make this request:



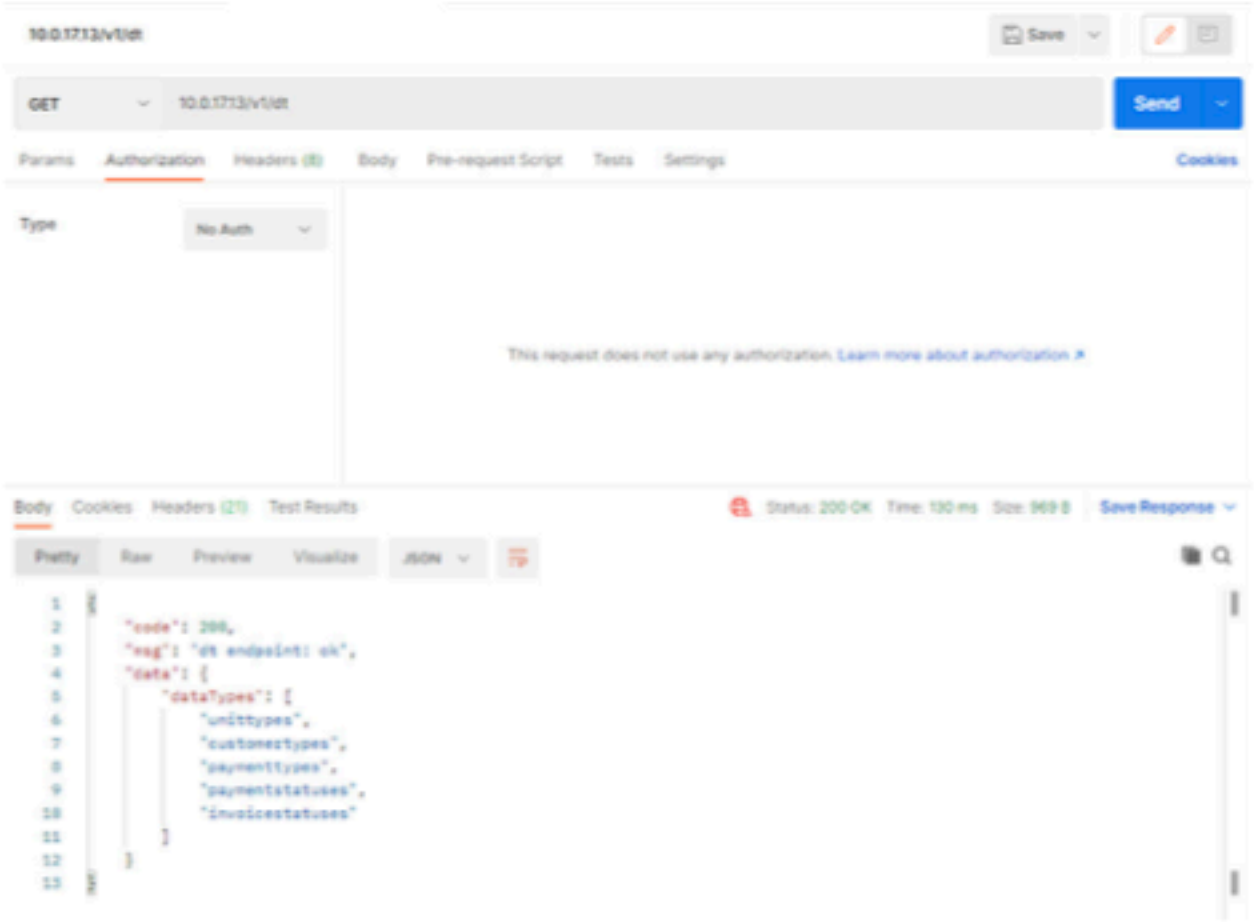
Use the obtained token to make authenticated queries to other endpoints on the same host or connect to the MariaDB server using the password contained in the token.

Moderate

API Data Leakage
Description
Host: 10.0.17.13 The ecommerce API allowed unauthenticated access to information. This exposed information about types of customers, payment, and payment or invoice statuses. This information was accessible via the Internet and on Le BonBon Croissant's systems.
Impact
Business Impact Score: 12 **CVSS**: 5.3 **Explanation of Business Impact:** The leakage of customer information puts LBC out of compliance with PCI-DSS which can result in financial penalties. Furthermore, customer data being leaked as the result of a breach can result in substantial reputational damage following a public breach disclosure. This can result in customer loss due to reputational damage.
Tactical Remediation
The API should not allow unauthorized access and thus require a form of authentication for any queries.
Strategic Remediation
The information can be inserted into a secure system that requires user authentication rather than an API token. This system should also be compliant with PCI-DSS and GDPR.
Short Term Remediation
If the API structure proves challenging to change, it is recommended that the API paths lacking authentication be removed from publicly available sources.

Reproduction:

The basepath `/v1/dt` was used for the host 10.0.17.13, which gave the host information about other possible paths, including types of customers, payments, payment and invoice statuses, which could be used to see their data structure:

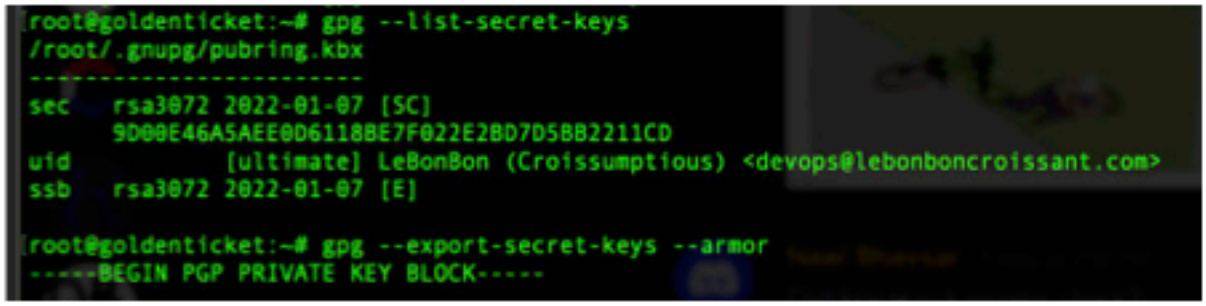


Unprotected GPG Private Keys
Description
Hosts: 10.0.17.10-13 Several PGP private keys were found on multiple hosts at `/root/.gnupg/`. These private keys are not protected with a password.
Impact
Business Impact Score: 10 **CVSS**: 5.3 **Explanation of Business Impact:** Compromise of private keys can lead to various types of business damage depending upon how they are used by applications. For example, GPG keys used for secure email configuration can lead to complete confidentiality compromise if they are leaked.
Tactical Remediation
Secure private keys using a strong password only known to appropriate employees who require it.
Strategic Remediation
Implement secure storage for various secrets used in the organization. For example, make use of an enterprise keyvault service.
Short Term Remediation
Limit the exposure of these keys by implementing firewalls and other mechanisms like account lockout periods to keep attackers at bay.

Reproduction:

```
ssh root@<Affected-Host>
# gpg --list-secret-keys
# gpg --export-secret-keys --armor
```

The below screenshot shows an example:



Low

Lower Versions of TLS Allowed
Description
Hosts: 10.0.17.10-11, 10.0.17.13 Insecure versions of TLS protocol are allowed by the proxy service configured on various hosts. Older versions of TLS are vulnerable to various attacks like BEAST (Browser Exploit Against SSL/TLS) and man in the middle attacks.
Impact
Business Impact Score: 3 **CVSS**: 5.3 **Explanation of Business Impact:** An attack resulting from this vulnerability can disrupt customers' experiences while using LBC's services. It is more likely that customers will be victims of scams and attacks while trying to shop on LBCs websites. This could result in customer loss if malicious actors are able to intercept and compromise customer data.
Tactical Remediation
Disallow old versions of TLS. Most libraries used by the services are compatible with the latest version of TLS v1.3.
Strategic Remediation
Implement an organizational patch management system that checks and updates out of date software and libraries.
Short Term Remediation
Add a web application firewall to block TLSv1 and TLSv1.1.

Reproduction:

The `proxy.conf` file in the Bitnami containers allows this insecure configuration as shown in the below snippet:

```
keepalive_timeout  65;

ssl_protocols    TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;

ssl_ciphers      HIGH:!aNULL:!MD5;

client_max_body_size 80M;
```

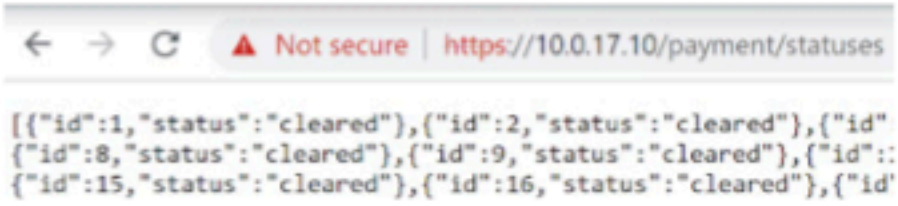
Minimal

Payment Information API Leakage
Description
Host: 10.0.17.10 The affected host's API can leak customer information without authentication. The result is that a user can view others' payment statuses and their related invoices. No PII was leaked as only ID numbers associated with transactions were shown.
Impact
Business Impact Score: 2 **CVSS**: 5.3 **Explanation of Business Impact:** The disclosure of customer information to unauthorized parties places LBC out of compliance with PCI-DSS. However, this leak did not contain PII or other sensitive data. Customer data should only be accessible to the customer and entities within LBC that have legitimate business reasons. If LBC is found out of compliance then they could receive a financial penalty due to these violations.
Tactical Remediation
The payment transaction API should be rewritten such that only authentication and authorized employees from LBC can access the payment APIs.
Strategic Remediation
Implement a company wide audit program to validate transactions and ensure the security of payment processes. Audits should be performed on a quarterly basis focusing on payment security. Unit testing should be implemented as part of the CI/CD development, including tests that check if a user token is able to access information for another customer. Developers should also receive training on developing secure web applications.
Short Term Remediation

Immediately redact access so that the API can only be accessed from within the LBC internal network.

Reproduction:

Navigate to https://10.0.17.10/payment/statuses:



Publicly Advertised Company Vulnerabilities
Description
Host: 10.0.17.13 Information about the company structure was publicly available due to an employee asking a question on the popular technical platform StackOverflow. The post contained sensitive information that could be traced back to the company, such as due to the employee's username on the platform.
Impact
Business Impact Score: 1 **CVSS**: 4.0 **Explanation of Business Impact:** The public availability of the company's data structure makes it significantly easier for attackers to exploit. While no impact can come directly from it existing it increases the likelihood that an incident can occur later against LBC. Malicious actors could use the gained information to identify similar API patterns inside LBC.
Tactical Remediation
The StackOverflow post should be removed. Afterwards, host 10.0.17.13 should be updated to enforce authentication for users to access the information on the API. Information about API tokens can be found in the following link: https://swagger.io/docs/specification/2-0/authentication/api-keys/.
Strategic Remediation
Employees should be provided security training to understand the potential catastrophe if sensitive data is leaked to unauthorized users.
Short Term Remediation
The removal of the StackOverflow post would provide the most help in the short term.

Reproduction:

A StackOverflow post was found that contained information about a publicly accessible API that the company contained. It was simple to link it back to Le BonBon Croissant's name due to the username of the employee on the post, as well as information in the post itself.

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

Informational

Music Player Daemon Running as Root
Description
Host: 10.0.17.87 The Music Player Daemon is running as `root`, which could potentially allow an attacker to have full privileges on the host.
Tactical Remediation
Change the configuration file at `/etc/mpd.conf` so the `user` and `group` settings are not privileged users and groups.
Strategic Remediation
Security teams should routinely ensure that minimal processes are running as `root`.

Reproduction:

The configuration file at `/etc/mpd.conf` contains the following content:

```
# General music daemon options ########################################################
#
# This setting specifies the user that MPD will run as. MPD should never run as
# root and you may use this setting to make MPD change its user ID after
# initialization. This setting is disabled by default and MPD is run as the
# current user.
#
user                            "root"
#
# This setting specifies the group that MPD will run as. If not specified
# primary group of user specified with "user" setting will be used (if set).
# This is useful if MPD needs to be a member of group such as "audio" to
# have permission to use sound card.
#
group                           "root"
```

Poor SQL Client Development
Description
Hosts: 10.0.17.10 The payment API does not prepare SQL queries and contains a SQL syntax error.
Tactical Remediation
The API code should be modified such that all SQL queries are prepared and there are no syntax errors.
Strategic Remediation
Unit testing should be completed during the CI/CD pipeline to ensure the API will not throw errors for intended functionality. Developers should receive training on how to prevent SQL injections.

Reproduction:

The source code for the API at 10.0.17.10 shows that there are SQL queries where user input is concatenated instead of using a prepared statement:

```
payment_query = f"SELECT * from billing.payments WHERE id = {payment_id}"
```

```
cc_data = query_db(f"""
SELECT billing.credit_cards.id,
billing.credit_cards.name,
billing.credit_cards.number,
billing.credit_cards.expiration,
billing.credit_cards.ccv,
billing.credit_cards.zip
FROM billing.credit_cards
JOIN billing.payment_methods
ON (billing.payment_methods.payment_ref = billing.credit_cards.id)
WHERE billing.payment_methods.customer_id = '{content['customer_id']}'
```

Thankfully, these queries require the user input to be an integer or UUID, so a SQL injection is likely not possible. However, prepared queries should always be used to prevent any possibility of SQL injection and because it is standard security practice.

Furthermore, there is a syntax error in the remove payment method SQL query:

```
delete_payment_methods_query = f"DELETE * from billing.payment_methods WHERE id = {args['id']}"
```

Attempting to use the endpoint results in an internal server error:

```
┌──# curl https://10.0.17.10/payment_method?id=1 -k -X DELETE
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>500 Internal Server Error</title>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error and was unable to
 the application.</p>
```

Outdated Docker Base Image
Description
Hosts: 10.0.17.10-13 The base Docker image `bitnami/minideb:buster` is outdated and known to have multiple vulnerabilities due to the outdated Linux dependencies used within the image.
Tactical Remediation
To remediate this vulnerability, update the base image to the more recent version `bitnami/minideb:bullseye`.
Strategic Remediation
As a long-term remediation, a Docker patching and update policy should be implemented to ensure that secure and up-to-date Docker images are used for all containers.

Reproduction:

```
docker ps
```

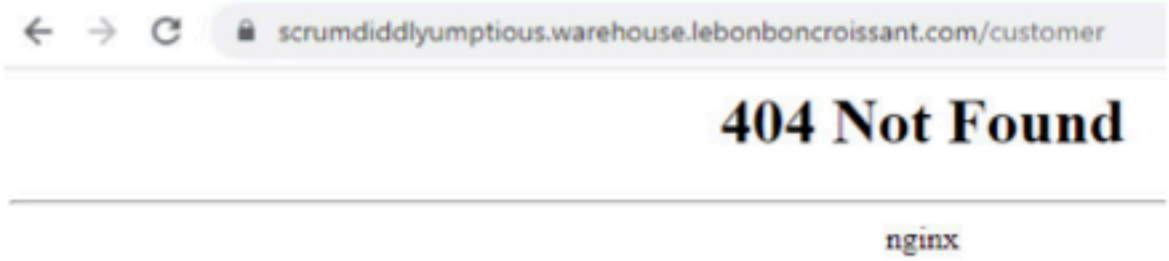
```
docker scan mrossja/cptc-2021-twix
```

```
Package manager:   deb
Project name:      docker-image|rossja/cptc-2021-twix
Docker image:      rossja/cptc-2021-twix
Platform:          linux/amd64
Base image:        bitnami/minideb:buster

Tested 108 dependencies for known vulnerabilities, found 79 vulnerabilities.

Base Image              Vulnerabilities  Severity
bitnami/minideb:buster  62               2 critical, 11 high, 5 medium, 44 low

Recommendations for base image upgrade:

Major upgrades
Base Image                 Vulnerabilities  Severity
bitnami/minideb:bullseye   37               1 critical, 2 high, 0 medium, 34 low
```

Unfinished Website
Description
Host: 10.0.17.12 The `scrumdiddlyumptious.warehouse.lebonboncroissant.com` store site lacks several critical functionality necessary for customer usage. Customers were unable to add items to their cart or make payments. Additionally, refreshing the site resulted in a 404 Not Found error and the site could not be accessed with commonly used browsers such as Internet Explorer.
Tactical Remediation
To remediate this, code should be written to add the missing functionality and improve the stability of the site.
Strategic Remediation
As a long-term remediation, a code review policy should be implemented to ensure that services are thoroughly tested before being deployed.

Reproduction:

Refreshing the page results in the following error:



Site JavaScript files such as `payment.js` were sparse, lacking the functionality to be actually used by customers:

```
import React from 'react';
import Login from './login';

class Payment extends React.Component {

  constructor(props) {
    super(props);
    this.state = {};
  }

  render() {
    if(this.props.token === '') {
    return <Login setToken={this.props.setToken} />
    } else {
      return(
        <div className="App-container">
          <div className="App-content">
            <h1 className="pageName">Payment Page Content!</h1>
          </div>
        </div>
      );
    }
  };
};

export default Payment;
```

Improvement: Ghostcat Vulnerability
Description
Host: 10.0.17.50 The Apache Tomcat server was previously vulnerable to CVE-2020-1938, which potentially could have led to remote code execution. JServ was no longer listening, so the server is no longer vulnerable.

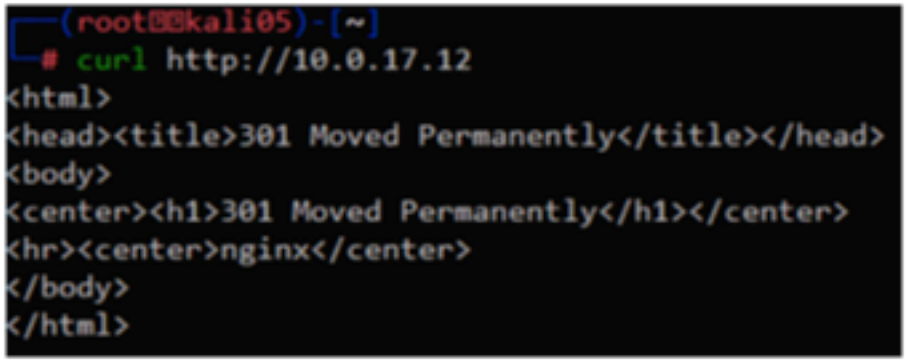
Reproduction:

Running the command `nmap 10.0.17.50 -p8009` no longer results in an open result.

Improvement: Cleartext Communications
Description
Hosts: 10.0.17.10-13 Web servers now redirect HTTP to HTTPS instead of communicating over cleartext. This improvement helps protect sensitive information such as customer and LBC data as well as helps defend against man-in-the-middle attacks.

Reproduction:

Attempting to run `curl http://10.0.17.12` will result in a Moved Permanently response and accessing the site in a web browser will result in an automatic redirection to the HTTPS site:



Improvement: SSH Shared Host Keys
Description
Hosts: 10.0.17.10-16, 10.0.17.50, 10.0.17.87 Each SSH server now has a unique host key. This prevents attackers that compromise one host from impersonating multiple hosts.

Reproduction:

Running the command `nmap 10.0.17.0/24 --script ssh-hostkey -p22` no longer results in the following output:

```
… ssh-hostkey: Possible duplicate hosts …
```

Improvement: Werkzeug Debugger Console
Description
Host: 10.0.17.13 The host previously had an instance of Werkzeug running with the debugger console exposed with only a pin securing it. The console has now been removed, no longer exposing the vulnerability.

Reproduction:

Running the command `nmap 10.0.17.13 -p 2000` no longer results in an open result.

Appendix

Risk Classification

	1- Negligible *Insignificant disruption to operations. Or disruption of less than 2 hours.*	**2 - Minor** *Minimal disruption to operations. Or between 2 and 5 hours of disruption.*	**3 - Moderate** *Noticeable disruption to operations. Or between 5 and 10 hours of disruption.*	**4 - Major** *Operations are severely impacted. Or between and 24 hours of disruption.*	**5 - Catastrophic** *Operations are completed halted or multiple days of disruption*
5 - Very Likely *Above 80% chance of exploitation. Or would require no-skill to exploit.*	5	10	15	20	25
4 - Likely *Between 79% and 60% chance of exploitation. Or would require minimal skill to exploit.*	4	8	12	16	20
3 - Moderate *Between 59% and 30% chance of exploitation. Or would require some skill to exploit.*	3	6	9	12	15
2- Unlikely *Between 29% and 10% chance of exploitation. Or would require expert skills to exploit.*	2	4	6	8	10
1 - Incidental	1	2	3	4	5

Less than a 9% change of exploitation. Would only be exploited through accidental means					

The team evaluated findings during the engagement using a custom risk/impact matrix-based loosely on the Penetration Execution standard and the CVSS scoring matrix. The team sought to give a rating that reflected the potential damage to the business if the vulnerability was exploited. This possibility was multiplied with the skill level required and the likelihood of an incident occurring not overly to alarm the IT team of Le BonBon Croissant. The result is a composite impact/risk score used to rank all finds from the engagement, with the most severe score being 25 and the lowest being 1. The impact score for the first half of the composite score is from 1 to 5, with 1 having little to no impact on business operations and 5 being catastrophic failures. The likelihood score is also on a scale from 1 to 5, with 1 being a very little or accidental chance of exploitation, where 5 would be a high likelihood or requiring almost no skill to exploit. These sub-scores result in the composite risk/impact score used throughout the assessment.

CVSS scores were used to determine the difficulty and, therefore, likelihood score for the matrix. The CVSS scores are still included within the technical findings section for the engineering team to reference. The main components of a CVSS score are the impact on the CIA triad, the level of access required, the attack vector, and the complexity of the vulnerability.

Components are informative to the remediation team but give little insight to business operation and therefore are left within the technical findings section. The risk matrix above determines the team's business impact score to rate the vulnerability's risk.