Galoy

Security Assessment

March 31, 2022

Prepared for:
Justin Carter
Galoy

Prepared by: **Maciej Domański, Emilio López, and Hamid Kashfi**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 80+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and mutually agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

Galoy engaged Trail of Bits to review the security of its platform. From February 28 to March 11, 2022, a team of three consultants conducted a security review of the client-provided source code, with four person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with access to repositories and relevant documentation provided by the Galoy team.

Summary of Findings

The audit uncovered several flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Medium	5
Low	12
Informational	5

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	4
Authentication	5
Configuration	6
Cryptography	3
Data Exposure	1
Data Validation	3
Patching	2

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

- **TOB-GALOY-10**
  The Galoy system relies on SMS one-time passwords (OTPs) for user authentication. The implementation suffers from several issues; for example, the OTPs do not expire within a reasonable amount of time and are not invalidated after a user has logged out of a session or requested a new OTP. These issues could enable an attacker with access to a user's OTP to gain persistent access to the user's account.

- **TOB-GALOY-12**
  Galoy deployments include a Grafana instance. The version of Grafana being deployed is outdated and affected by known security issues. Additionally, the instance is directly exposed to the internet. An attacker could exploit one of the known issues in this publicly accessible instance to gain access to the system.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineers were associated with this project:

Maciej Domański, Consultant
maciej.domanski@trailofbits.com

Emilio López, Consultant
emilio.lopez@trailofbits.com

Hamid Kashfi, Consultant
hamid.kashfi@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
February 24, 2022	Pre-project kickoff call
March 8, 2022	Status update meeting #1
March 14, 2022	Delivery of final report draft; report readout meeting
March 31, 2022	Delivery of public report

Project Goals

The engagement was scoped to provide a security assessment of the Galoy platform. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are there appropriate access controls on critical methods?
- Are the API endpoints implemented securely?
- Is it possible to gain access to the cluster and empty a hot wallet?
- Is it possible to tamper with the ledger and slowly drain funds from a cold wallet?
- Could an attacker gain control of Kubernetes clusters?
- Does the project use cryptography appropriately?
- Is the solution susceptible to denial-of-service attacks?
- Do the repositories expose any sensitive information or secrets?

Project Targets

The engagement involved a review and testing of the targets listed below.

Galoy

Repository	https://github.com/GaloyMoney/galoy
Version	`e893f47094442883d6f9125b83e812b8109f96ef`
Type	TypeScript
Platform	Node.js

Galoy Terraform Infrastructure

Repository	https://github.com/GaloyMoney/galoy-infra
Version	`214c9e4e098acd6c19768252cccc0b90d579b758`
Type	HashiCorp Configuration Language (HCL)
Platform	Terraform

Galoy Helm Charts

Repository	https://github.com/GaloyMoney/charts
Version	`6af7bce904616f8194ece9c50d02dd79ec8c7fbc`
Type	Helm charts
Platform	Helm

Web Wallet

Repository	https://github.com/GaloyMoney/web-wallet
Version	`bad8a92305bbfa1ae31b9dde8db9601e7f1d1d34`
Type	TypeScript
Platform	Node.js

Admin Panel

Repository	https://github.com/GaloyMoney/admin-panel
Version	`10c818c2dddb73d4c6358a1dba446931038be48a`
Type	TypeScript
Platform	Node.js

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

- We performed a full manual review of the `galoy`, `web-wallet`, and `admin-panel` repositories and ran static and dynamic analysis tools over the codebase.
- We performed a similar audit of the `galoy-infra` and `charts` repositories. Although we were not able to exhaustively review every line of code, we inspected all major code paths and critical functionalities; therefore, we are confident that this review covered all security-relevant logic.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- **Third-party dependencies.** The platform uses many third-party packages. We used automated dependency-auditing tools to identify outdated packages with known high- or critical-severity vulnerabilities and manually reviewed only those packages.
- **Apollo GraphQL Server.** The Galoy platform uses the Apollo GraphQL server, a popular and widely used solution for hosting GraphQL. There are very few public advisories associated with this server, which is often indicative of a product that lacks proper third-party security reviews. We believe that an in-depth security audit of this product would benefit the Galoy platform.
- **Services exposed over the internet.** We identified hosts, subdomains, and applications exposed over the internet, including more than 80 subdomains, many of which share the same hosts. These also include web applications exposed unnecessarily, some of which have a long track record of vulnerabilities, such as `JupyterHub`. We used publicly available asset databases and passive information gathering to identify these services. Time constraints prevented us from manually reviewing all of the assets, but we recommend that they be covered in a future review of the Galoy code. Appendix D provides a list of these assets and a link to a visual graph of them.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	We did not identify any severe arithmetic issues. The solution would benefit from using the `Number.isSafeInteger` function before the `Number.parseInt` function and ensuring that other type-related edge cases are checked.	**Satisfactory**
Auditing	We did not find any issues related to events or logging.	**Satisfactory**
Authentication / Access Controls	We identified one functionality that appears to rely on client-side access controls and found that the JSON web token implementation does not follow security best practices. We also found two notable issues related to the handling of authentication and the use of SMS to deliver OTPs.	**Weak**
Complexity Management	Most functions have a clear purpose, and the architecture is modular.	**Satisfactory**
Configuration	We identified services that are overly exposed to the internet (e.g., Galoy's admin panel and Grafana). Limiting their exposure would reduce the number of potential attack vectors.	**Moderate**
Cryptography and Key Management	Sensitive data such as secrets and passwords is stored in the filesystem in plaintext.	**Weak**
Data Handling	We did not find any issues related to the validation of data processed by the system.	**Satisfactory**

Documentation	The public documentation and the documentation provided by the Galoy team adequately describe key aspects of the system.	**Satisfactory**
Maintenance	We identified many outdated packages with known security issues, which indicates that the package management policy or the checks performed in the continuous integration / continuous development pipeline are ineffective.	**Moderate**
Memory Safety and Error Handling	The application handles errors consistently and securely.	**Satisfactory**
Testing and Verification	There are many tests in the codebase, though additional tests for many other operations would be beneficial. We recommend introducing additional tests for edge cases and complex behaviors.	**Moderate**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Insecure download process for the yq tool	Data Validation	**Low**
2	Use of unencrypted HTTP scheme	Cryptography	**Low**
3	Lack of expiration and revocation mechanism for JWTs	Authentication	**Medium**
4	Use of insecure function to generate phone codes	Cryptography	**Low**
5	Redundant basic authentication method	Authentication	**Informational**
6	GraphQL queries may facilitate CSRF attacks	Access Controls	**Low**
7	Potential ReDoS risk	Data Validation	**Informational**
8	Use of MD5 to generate unique GeeTest identifiers	Cryptography	**Low**
9	Reliance on SMS-based OTPs for authentication	Authentication	**Medium**
10	Incorrect handling and implementation of SMS OTPs	Authentication	**High**
11	Vulnerable and outdated Node packages	Patching	**Medium**
12	Outdated and internet-exposed Grafana instance	Patching	**High**

13	Incorrect processing of GET path parameter	Data Validation	**Low**
14	Discrepancies in API and GUI access controls	Access Controls	**Low**
15	Cloud SQL does not require TLS connections	Configuration	**Low**
16	Kubernetes node pools are not configured to auto-upgrade	Configuration	**Informational**
17	Overly permissive firewall rules	Configuration	**Medium**
18	Lack of uniform bucket-level access in Terraform state bucket	Access Controls	**Informational**
19	Insecure storage of passwords	Data Exposure	**Medium**
20	Third-party container images are not version pinned	Configuration	**Low**
21	Compute instances do not leverage Shielded VM features	Configuration	**Informational**
22	Excessive container permissions	Access Controls	**Low**
23	Unsigned and unversioned Grafana BigQuery Datasource plugin	Configuration	**Low**
24	Insufficient validation of JWTs used for GraphQL subscriptions	Authentication	**Low**

Detailed Findings

1. Insecure download process for the yq tool

Severity: Low	Difficulty: High
Type: Data Validation	Finding ID: TOB-GALOY-1
Target: `galoy/ci/image/Dockerfile`, `galoy-infra/modules/inception/gcp/bastion-startup.tmpl`	

Description

The Dockerfile uses the Wget utility to download the yq tool but does not verify the file it has downloaded by its `checksum` or signature. Without verification, an archive that has been corrupted or modified by a malicious third party may not be detected. Figures 1.1 and 1.2 show cases in which a tool is downloaded without verification of its checksum.

```
 6    RUN wget
https://github.com/mikefarah/yq/releases/download/v4.17.2/yq_linux_386.tar.gz -O -
|\
 7      tar xz && mv yq_linux_386 /usr/bin/yq
```

Figure 1.1: The Dockerfile downloads and unarchives the yq tool. (ci/image/Dockerfile#6–7)

```
 41    wget
https://github.com/bodymindarts/cepler/releases/download/v${cepler_version}/cepler-x
86_64-unknown-linux-musl-${cepler_version}.tar.gz \
 42      && tar -zxvf cepler-x86_64-unknown-linux-musl-${cepler_version}.tar.gz \
 43      && mv cepler-x86_64-unknown-linux-musl-${cepler_version}/cepler
/usr/local/bin \
 44      && chmod +x /usr/local/bin/cepler \
 45      && rm -rf ./cepler-*
```

Figure 1.2: The `bastion-startup` script downloads and unarchives the `cepler` tool. (modules/inception/gcp/bastion-startup.tmpl#41–45)

Exploit Scenario

An attacker gains access to the GitHub repository from which yq is downloaded. The attacker then modifies the binary to create a reverse shell upon yq's startup. When a user runs the Dockerfile, the attacker gains access to the user's container.

Recommendations

Short term, have the Dockerfile and other scripts in the solution verify each file they download by its `checksum`.

Long term, implement checks to ensure the integrity of all third-party components used in the solution and periodically check that all components are downloaded from encrypted URLs.

2. Use of unencrypted HTTP scheme	
Severity: **Low**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-GALOY-2
Target: `galoy/src/services/ipfetcher/index.ts`	

Description

The Galoy `ipfetcher` module uses the unencrypted HTTP scheme (figure 2.1). As a result, an attacker in the same network as the host invoking the code in figure 2.1 could intercept and modify both the request and `ipfetcher`'s response to it, potentially accessing sensitive information.

```
 8    const { data } = await axios.get(
 9      `http://proxycheck.io/v2/${ip}?key=${PROXY_CHECK_APIKEY}&vpn=1&asn=1`,
10    )
```

Figure 2.1: `src/services/ipfetcher/index.ts#8–10`

Exploit Scenario

Eve gains access to Alice's network and obtains Alice's `PROXY_CHECK_APIKEY` by observing the unencrypted network traffic.

Recommendations

Short term, change the URL scheme used in the `ipfetcher` service to HTTPS.

Long term, use tools such as WebStorm code inspections to find other uses of unencrypted URLs.

3. Lack of expiration and revocation mechanism for JWTs

Severity: **Medium**	Difficulty: **High**
Type: Authentication	Finding ID: TOB-GALOY-3
Target: `galoy/src/services/jwt.ts`	

Description

The Galoy system uses JSON web tokens (JWTs) for authentication. A user obtains a new JWT by calling the `userLogin` GraphQL mutation. Once a token has been signed, it is valid forever; the platform does not set an expiration time for tokens and cannot revoke them.

```
 7    export const createToken = ({
 8      uid,
 9      network,
10    }: {
11      uid: UserId
12      network: BtcNetwork
13    }): JwtToken => {
14      return jwt.sign({ uid, network }, JWT_SECRET, {
// (...)
25        algorithm: "HS256",
26      }) as JwtToken
27    }
```

Figure 3.1: The creation of a JWT (`src/services/jwt.ts#7–27`)

Exploit Scenario

An attacker obtains a user's JWT and gains persistent access to the system. The attacker then engages in destructive behavior. The victim eventually notices the behavior but does not have a way to stop it.

Recommendations

Short term, consider setting an expiration time for JWTs, and implement a mechanism for revoking tokens. That way, if a JWT is leaked, an attacker will not gain persistent access to the system.

4. Use of insecure function to generate phone codes

Severity: **Low**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-GALOY-4
Target: `galoy/src/services/ipfetcher/index.ts`	

Description

The Galoy application generates a verification code by using the JavaScript function `Math.random()`, which is not a cryptographically secure pseudorandom number generator (CSPRNG).

```
10    const randomIntFromInterval = (min, max) =>
11      Math.floor(Math.random() * (max - min + 1) + min)
12
// (...)
82
83      const code = String(randomIntFromInterval(100000, 999999)) as PhoneCode
84      const galoyInstanceName = getGaloyInstanceName()
85      const body = `${code} is your verification code for ${galoyInstanceName}`
86
87      const result = await PhoneCodesRepository().persistNew({
88        phone: phoneNumberValid,
89        code,
90      })
91      if (result instanceof Error) return result
92
93      const sendTextArguments = { body, to: phoneNumberValid, logger }
94
95      return TwilioClient().sendText(sendTextArguments)
96    }
```

Figure 4.1: `src/app/users/request-phone-code.ts#10–96`

Exploit Scenario

An attacker repeatedly generates verification codes and analyzes the values and the order of their generation. The attacker attempts to deduce the pseudorandom number generator's internal state. If successful, the attacker can then perform an offline calculation to predict future verification codes.

Recommendations

Short term, replace `Math.random()` with a CSPRNG.

Long term, always use a CSPRNG to generate random values for cryptographic operations.

5. Redundant basic authentication method

Severity: **Informational**	Difficulty: **Undetermined**
Type: Authentication	Finding ID: TOB-GALOY-5
Target: `galoy/rc/servers/middlewares/api-key-auth.ts`	

Description

The Galoy application implements a basic authentication method (figure 5.1) that is redundant because the `apiKey` is not being used. Superfluous authentication methods create new attack vectors and should be removed from the codebase.

```
  1    import express from "express"
  2
  3    const formatError = new Error("Format is Authorization: Basic
<base64(key:secret)>")
  4
  5    export default async function (
  6      req: express.Request,
  7      _res: express.Response,
  8      next: express.NextFunction,
  9    ) {
 10      const authorization = req.headers["authorization"]
 11      if (!authorization) return next()
 12
 13      const parts = authorization.split(" ")
 14      if (parts.length !== 2) return next()
 15
 16      const scheme = parts[0]
 17      if (!/Basic/i.test(scheme)) return next()
 18
 19      const credentials = Buffer.from(parts[1], "base64").toString().split(":")
 20      if (credentials.length !== 2) return next(formatError)
 21
 22      const [apiKey, apiSecret] = credentials
 23      if (!apiKey || !apiSecret) return next(formatError)
 24
 25      req["apiKey"] = apiKey
 26      req["apiSecret"] = apiSecret
 27      next()
 28    }
```

Figure 5.1: The basic authentication method implementation (`src/servers/middlewares/api-key-auth.ts#1–28`)

Recommendations

Short term, remove the `apiKey`-related code.

Long term, review and clearly document the Galoy authentication methods.

6. GraphQL queries may facilitate CSRF attacks

Severity: **Low**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-GALOY-6
Target: `galoy/src/graphql`	

Description

The Galoy application's `/graphql` endpoint handles queries sent via GET requests. It is impossible to pass state-changing mutations or subscriptions in GET requests, and authorized queries need the `Authorization: Bearer` header. However, if a state-changing GraphQL operation were mislabeled as a query (typically a non-state-changing request), the endpoint would be vulnerable to cross-site request forgery (CSRF) attacks.

Exploit Scenario

An attacker creates a malicious website with JavaScript code that sends requests to the `/graphql` endpoint (figure 6.1). When a user visits the website, the JavaScript code is executed in the user's browser, changing the server's state.

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="http://192.168.236.135:4002/graphql">
      <input type="hidden" name="query"
value="query&#32;&#123;&#10;&#9;btcPriceList&#40;range&#58;ONE&#95;MONTH&#41;&#32;&#
123;&#10;&#9;&#9;price&#32;&#123;&#10;&#9;&#9;&#9;offset&#10;&#9;&#9;&#125;&#10;&#9;
&#9;timestamp&#10;&#9;&#125;&#10;&#125;" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Figure 6.1: In this proof-of-concept CSRF attack, the malicious website sends a request (the `btcPriceList` query) when the victim clicks "Submit request."

Recommendations

Short term, disallow the use of the GET method to send queries, or enhance the CSRF protections for GET requests.

Long term, identify all state-changing endpoints and ensure that they are protected by an authentication or anti-CSRF mechanism. Then implement tests for those endpoints.

References

- Cross-Origin Resource Sharing, Mozilla documentation
- Cross-Site Request Forgery Prevention, OWASP Cheat Sheet Series

7. Potential ReDoS risk

Severity: **Informational**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-GALOY-7
Target: `galoy/rc/servers/middlewares/api-key-auth.ts`	

Description

The `caseInsensitiveRegex` function takes an `input: string` parameter and uses it to create a new RegExp object (figure 7.1). Users cannot currently control the input parameter (the regular expression) or the string; however, if users gain that ability as the code is developed, it may enable them to cause a regular expression denial of service (ReDoS).

```
13    export const caseInsensitiveRegex = (input: string) => {
14      return new RegExp(`^${input}$`, "i")
15    }
```

Figure 7.1: `src/services/mongoose/users.ts#13–15`

```
37    const findByUsername = async (
38      username: Username,
39    ): Promise<Account | RepositoryError> => {
40      try {
41        const result = await User.findOne(
42          { username: caseInsensitiveRegex(username) },
```

Figure 7.2: `src/services/mongoose/accounts.ts#37–42`

Exploit Scenario

An attacker registers an account with a specially crafted username (line 2, figure 7.3), which forms part of a regex. The attacker then finds a way to pass the malicious regex (line 1, figure 7.3) to the `findByUsername` function, causing a denial of service on a victim's machine.

```
1    let test = caseInsensitiveRegex("(.*){1,32000}[bc]")
2    let s = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!"
3    s.match(test)
```

Figure 7.3: A proof of concept for the ReDoS vulnerability

Recommendations

Short term, ensure that input passed to the `caseInsensitiveRegex` function is properly validated and sanitized.

8. Use of MD5 to generate unique GeeTest identifiers

Severity: **Low**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-GALOY-8
Target: `galoy/src/services/geetest.ts`	

Description

The Galoy application uses MD5 hashing to generate a unique identifier during GeeTest service registration. MD5 is an insecure hash function and should never be used in a security-relevant context.

```
 33    const register = async (): Promise<UnknownCaptchaError | GeetestRegister> =>
{
 34      try {
 35        const gtLib = new GeetestLib(config.id, config.key)
 36        const digestmod = "md5"
 37        const params = {
 38          digestmod,
 39          client_type: "native",
 40        }
 41        const bypasscache = await getBypassStatus() // not a cache
 42        let result
 43        if (bypasscache === "success") {
 44          result = await gtLib.register(digestmod, params)
```

Figure 8.1: src/services/geetest.ts#33–44

Recommendations

Short term, change the hash function used in the `register` function to a stronger algorithm that will not cause collisions, such as SHA-256.

Long term, document all cryptographic algorithms used in the system, implement a policy governing their use, and create a plan for when and how to deprecate them.

9. Reliance on SMS-based OTPs for authentication	
Severity: **Medium**	Difficulty: **Medium**
Type: Authentication	Finding ID: TOB-GALOY-9
Target: `galoy/src/services/mongoose/phone-code.ts`	

Description

Galoy's authentication process is heavily reliant on the delivery of one-time passwords (OTPs) over SMS. This authentication method contravenes best practices and should not be used in applications that handle financial transactions or other sensitive operations.

SMS-based OTP leaves users vulnerable to multiple attacks and is considered an unsafe authentication method. Several of the most common and effective attack scenarios are described below.

- Text messages received by a mobile device can be intercepted by rogue applications on the device. Many users blindly authorize untrusted third-party applications to access their mobile phones' SMS databases; this means that a vulnerability in a third-party application could lead to the compromise and disclosure of the text messages on the device, including Galoy's SMS OTP messages.
- Another common technique used to target mobile finance applications is the interception of notifications on a device. Android operating systems, for instance, broadcast notifications across applications by design; a rogue application could subscribe to those notifications to access incoming text message notifications.
- Attackers also target SMS-based two-factor authentication and OTP implementations through SIM swapping. In short, an attacker uses social engineering to gather information about the owner of a SIM card and then, impersonating its owner, requests a new SIM card from the telecom company. All calls and text messages will then be sent to the attacker, leaving the original owner of the number out of the loop. This approach has been used in many recent attacks against crypto wallet owners, leading to millions of dollars in losses.

Recommendations

Short term, avoid using SMS authentication as anything other than an optional way to validate an account holder's identity and profile information. Instead of SMS-based OTP, provide support for hardware-based two-factor authentication methods such as Yubikey tokens, or software-based time-based one-time password (TOTP) implementations such as Google Authenticator and Authy.

References

- What is a Sim Swap? Definition and Related FAQs, Yubico

10. Incorrect handling and implementation of SMS OTPs

Severity: **High**	Difficulty: **Low**
Type: Authentication	Finding ID: TOB-GALOY-10
Target: `galoy/src/services/mongoose/phone-code.ts#L19`	

Description

Users authenticate to the web panel by providing OTPs sent to them over SMS. We identified two issues in the OTP authentication implementation:

1. The generated OTPs are persistent because OTP expiration dates are calculated incorrectly. The `Date.now()` method returns the epoch time in milliseconds, whereas it is meant to return the time in seconds.

```
294   const PhoneCodeSchema = new Schema({
295     created_at: {
296       type: Date,
297       default: Date.now,
298       required: true,
```

Figure 10.1: The default date value is expressed in milliseconds. (src/services/mongoose/schema.ts#294–298)

```
11    export const VALIDITY_TIME_CODE = (20 * 60) as Seconds
```

Figure 10.2: The default validity period is expressed in seconds. (src/config/index.ts#11)

```
49    const age = VALIDITY_TIME_CODE
50    const validCode = await isCodeValid({ phone: phoneNumberValid, code, age })
51    if (validCode instanceof Error) return validCode
```

Figure 10.3: Validation of an OTP's age (src/app/users/login.ts#49–51)

```
18    }): Promise<true | RepositoryError> => {
19      const timestamp = Date.now() / 1000 - age
20      try {
21        const phoneCode = await PhoneCode.findOne({
22          phone,
23          code,
24          created_at: {
25            $gte: timestamp,
26          },
```

Figure 10.4: The codebase validates the timestamp in seconds, while the default date is in milliseconds, as shown in figure 10.1. (src/services/mongoose/phone-code.ts#18–26)

2. The SMS OTPs are never discarded. When a new OTP is sent to a user, the old one remains valid regardless of its expiration time. A user's existing OTP tokens also remain valid if the user manually logs out of a session, which should not be the case. Tests of the `admin-panel` and `web-wallet` code confirmed that all SMS OTPs generated for a given phone number remain valid in these cases.

Exploit Scenario

After executing a successful phishing attack against a user, an attacker is able to intercept an OTP sent to that user, gaining persistent access to the victim's account. The attacker will be able to use the code even when the victim logs out of the session or requests a new OTP.

Recommendations

Short term, limit the lifetime of OTPs to two minutes. Additionally, immediately invalidate an OTP, even an unexpired one, when any of the following events occur:

- The user logs out of a session
- The user requests a new OTP
- The OTP is used successfully
- The OTP reaches its expiration time
- The user's account is locked for any reason (e.g., too many login attempts)

References

- NIST best practices for implementing authentication tokens

11. Vulnerable and outdated Node packages

Severity: **Medium**	Difficulty: **Low**
Type: Patching	Finding ID: TOB-GALOY-11
Target: `galoy/yarn.lock`, `web-panel/yarn.lock`, `admin-panel/yarn.lock`	

Description
We used the `yarn audit` and snyk tools to audit the project dependencies and components for known vulnerabilities and outdated versions, respectively. The project uses many outdated packages with known security vulnerabilities ranging from critical to low severity. A list of vulnerable and outdated packages is included in appendix C.

Vulnerabilities in packages imported by an application are not necessarily exploitable. In most cases, an affected method in a vulnerable package needs to be used in the right context to be exploitable. We manually reviewed the packages with high- or critical-severity vulnerabilities and did not find any vulnerabilities that could be exploited in the Galoy application. However, that could change as the code is further developed.

Exploit Scenario
An attacker fingerprints one of Galoy's components, identifies an out-of-date package with a known vulnerability, and uses it in an exploit against the component.

Recommendations
Short term, update the outdated and vulnerable dependencies.

Long term, integrate static analysis tools that can detect outdated and vulnerable libraries (such as the `yarn audit` and snyk tools) into the build and / or test pipeline. This will improve the system's security posture and help prevent the exploitation of project dependencies.

12. Outdated and internet-exposed Grafana instance

Severity: **High**	Difficulty: **Low**
Type: Patching	Finding ID: TOB-GALOY-12
Target: `grafana.freecorn.galoy.io`	

Description

The Grafana admin panel is exposed over the internet. A management interface should not be exposed over the internet unless it is protected by a secondary authentication or access control mechanism; these mechanisms (e.g., IP address restrictions and VPN solutions) can mitigate the immediate risk to an application if it experiences a vulnerability.

Moreover, the Grafana version deployed at grafana.freecorn.galoy.io is outdated and vulnerable to known security issues.

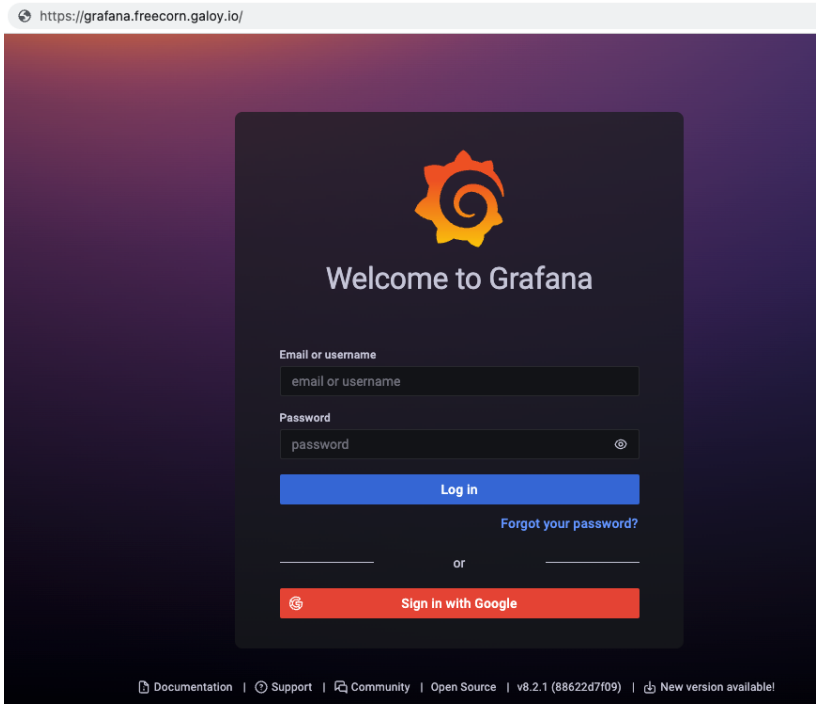

Figure 12.1: The outdated Grafana version (8.2.1) with known security issues

The version banner on the login page (figure 12.1) identifies the version as v8.2.1 (`88622d7f09`). This version has multiple moderate- and high-risk vulnerabilities. One of them, a path traversal vulnerability (CVE-2021-43798), could enable an unauthenticated

attacker to read the contents of arbitrary files on the server. However, we could not exploit this issue, and the Galoy team suggested that the code might have been patched through an upstream software deployment.

Time constraints prevented us from reviewing all Grafana instances for potential vulnerabilities. We reviewed only the grafana.freecorn.galoy.io instance, but the recommendations in this finding apply to all deployed instances.

Exploit Scenario

An attacker identifies the name of a valid plugin installed and active on the instance. By using a specially crafted URL, the attacker can read the contents of any file on the server (as long as the Grafana process has permission to access the file). This enables the attacker to read sensitive configuration files and to engage in remote command execution on the server.

Recommendations

Short term, avoid exposing any Grafana instance over the internet, and restrict access to each instance's management interface. This will make the remote exploitation of any issues much more challenging.

Long term, to avoid known security issues, review all deployed instances and ensure that they have been updated to the latest version. Additionally, review the Grafana log files for any indication of the attack described in CVE-2021-43798, which has been exploited in the wild.

References

- List of publicly known vulnerabilities affecting recent versions of Grafana

13. Incorrect processing of GET path parameter

Severity: **Low**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-GALOY-13
Target: `web-wallet/src/server/ssr-router.ts`	

Description

If the value of the hidden `path` parameter in the GET request in figure 13.1 does not match the value in the `appRoutesDef` array, the request will cause an unhandled error (figure 13.2). The error occurs when the result of the `serverRenderer` function is `undefined` (line 21, figure 13.3), because the "Invalid route path" error is thrown in the call to the `renderToStringWithData` function (figure 13.4).

```
GET /?path=aaaa HTTP/1.1
Host: localhost:3000
```

Figure 13.1: The HTTP request that triggers the error

```
HTTP/1.1 500 Internal Server Error
// (...)
ReferenceError: /Users/md/work/web-wallet/views/index.ejs:8
    6|     <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    7|
 >> 8|     <title><%- pageData.title %></title>
    9|
    10|     <link rel="stylesheet" href="/themes/<%- GwwConfig.walletTheme
-%>/colors.css" />
    11|     <link rel="stylesheet" href="/bundles/<%- gVars['main'][0] -%>" />
pageData is not defined at eval ("web-wallet/views/index.ejs":12:17)
    at index (web-wallet/node_modules/ejs/lib/ejs.js:692:17)
    at tryHandleCache (web-wallet/node_modules/ejs/lib/ejs.js:272:36)
    at View.exports.renderFile [as engine]
(web-wallet/node_modules/ejs/lib/ejs.js:489:10)
    at View.render (web-wallet/node_modules/express/lib/view.js:135:8)
    at tryRender (web-wallet/node_modules/express/lib/application.js:640:10)
    at Function.render (web-wallet/node_modules/express/lib/application.js:592:3)
    at ServerResponse.render
(web-wallet/node_modules/express/lib/response.js:1017:7)
    at web-wallet/src/server/ssr-router.ts:24:18</pre></body></html>
```

Figure 13.2: The HTTP response that shows the unhandled error

```
21    const vars = await serverRenderer(req)({  // undefined
22      path: checkedRoutePath,
23    })
24    return res.render("index", vars) // call when the vars is undefined
```

Figure 13.3: src/server/ssr-router.ts#21–24

```
10    export const serverRenderer =
11      (req: Request) =>
12      async ({
13        path,
14        flowData,
15      }: {
16        path: RoutePath | AuthRoutePath
17        flowData?: KratosFlowData
18      }) => {
19        try {
// (...)
43          const initialMarkup = await renderToStringWithData(App)
// (...)
79          })
80        } catch (err) {
81          console.error(err)
82        }
```

Figure 13.4: src/renderers/server.tsx#10–82

Exploit Scenario

An attacker finds a way to inject malicious code into the hidden `path` parameter. This results in an open redirect vulnerability, enabling the attacker to redirect a victim to a malicious website.

Recommendations

Short term, ensure that errors caused by an invalid `path` parameter value (one not included in the `appRoutesDef` whitelist) are handled correctly. A `path` parameter should not be processed if it is unused.

Long term, use Burp Suite Professional with the Param Miner extension to scan the application for hidden parameters.

14. Discrepancies in API and GUI access controls

Severity: **Low**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-GALOY-14
Target: `galoy/src/graphql/root/mutation/user-update-username.ts`	

Description

Although the Web Wallet's graphical user interface (GUI) does not allow changes to a username (figure 14.1), they can be made through the GraphQL `userUpdateUsername` mutation (figure 14.2).

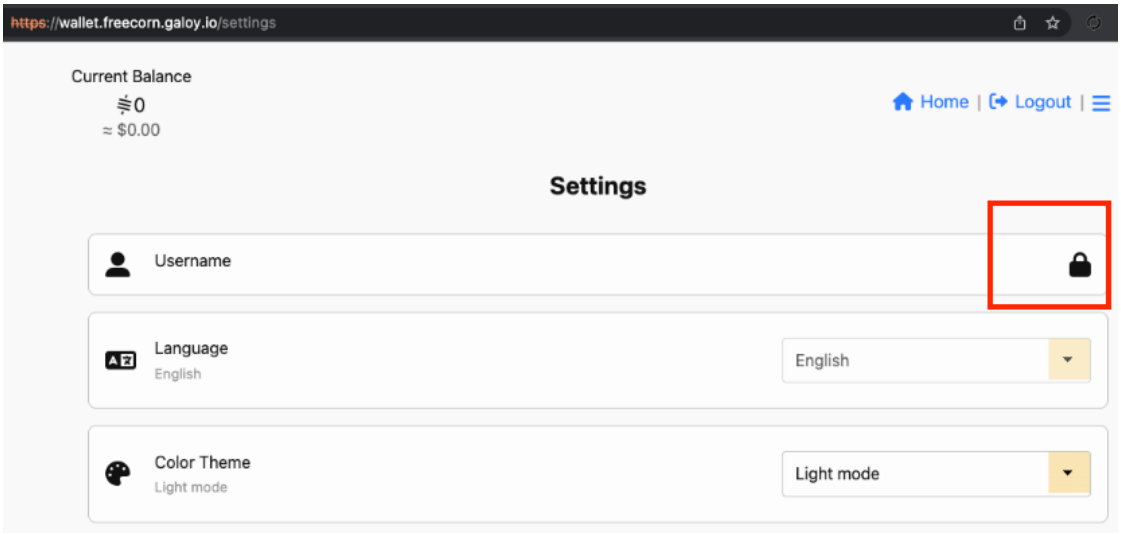

Figure 14.1: The lock icon on the "Settings" page indicates that it is not possible to change a username.

```
POST /graphql HTTP/2
Host: api.freecorn.galoy.io
Content-Length: 345
Content-Type: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiI2MjI3ODYwMWJlOGViYWYxZWRmNDBhNDYiLCJ
uZXR3b3JrIjoibWFpbm5ldCIsImlhdCI6MTY0Njc1NzU4NX0.ed2dk9gMQh5DJXCPpitj5wq78n0gFnmulRp
2KIXTVX0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Origin: https://wallet.freecorn.galoy.io

{"operationName":"userUpdateUsername","variables":{"input":{"username":"aaaaaaaaaaaa
aaa"}},"query":"mutation userUpdateUsername($input: UserUpdateUsernameInput!) {\n
userUpdateUsername(input: $input) {\n    errors {\n      message\n      __typename\n
}\n    user {\n      id\n      username\n      __typename\n    }\n    __typename\n
}\n}"}

HTTP/2 200 OK
// (...)

{"data":{"userUpdateUsername":{"errors":[],"user":{"id":"04f01fb4-6328-5982-a39a-eeb
027a2ceef","username":"aaaaaaaaaaaaaaa","__typename":"User"},"__typename":"UserUpdat
eUsernamePayload"}}}
```

Figure 14.2: The HTTP request-response cycle that enables username changes

Exploit Scenario

An attacker finds a discrepancy in the access controls of the GUI and API and is then able to use a sensitive method that the attacker should not be able to access.

Recommendations

Short term, avoid relying on client-side access controls. If the business logic of a functionality needs to be blocked, the block should be enforced in the server-side code.

Long term, create an access control matrix for specific roles in the application and implement unit tests to ensure that appropriate access controls are enforced server-side.

15. Cloud SQL does not require TLS connections

Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-GALOY-15
Target: `galoy-infra/modules/platform/gcp/cloud-sql/main.tf`	

Description
Terraform's declarative configuration file for the Cloud SQL instance does not indicate that PostgreSQL should enforce the use of Transport Layer Security (TLS) connections. Similarly, the Galoy solution does not use the Cloud SQL Auth proxy, which provides strong encryption and authentication using identity and access management. Because the database is exposed only in a virtual private cloud (VPC) network, this finding is of low severity.

Exploit Scenario
An attacker manages to eavesdrop on traffic in the VPC network. If one of the database clients is misconfigured, the attacker will be able to observe the database traffic in plaintext.

Recommendations
Short term, configure Cloud SQL to require the use of TLS, or use the Cloud SQL Auth proxy.

Long term, integrate Terrascan or another automated analysis tool into the workflow to detect areas of improvement in the solution.

References
- Configure SSL/TLS certificates, Cloud SQL documentation
- Connect from Google Kubernetes Engine, Cloud SQL documentation

16. Kubernetes node pools are not configured to auto-upgrade

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-GALOY-16
Target: `galoy-infra/modules/platform/gcp/kube.tf`	

Description

The Galoy application uses Google Kubernetes Engine (GKE) node pools in which the auto-upgrade functionality is disabled. The auto-upgrade functionality helps keep the nodes in a Kubernetes cluster up to date with the Kubernetes version running on the cluster control plane, which Google updates on the user's behalf. Auto-upgrades also ensure that security updates are timely applied. Disabling this setting is not recommended by Google and could create a security risk if patching is not performed manually.

```
124    management {
125     auto_repair  = true
126     auto_upgrade = false
127    }
```

Figure 16.1: The auto-upgrade property is set to `false`. (`modules/platform/gcp/kube.tf#124–127`)

Recommendations

Short term, enable the auto-upgrade functionality to ensure that the nodes are kept up to date and that security patches are timely applied.

Long term, remain up to date on the security features offered by Google Cloud. Integrate Terrascan or another automated tool into the development workflow to detect areas of improvement in the solution.

References

- Auto-upgrading nodes, GKE documentation

17. Overly permissive firewall rules

Severity: **Medium**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-GALOY-17
Target: `galoy-infra/modules/inception/gcp/bastion.tf`, `galoy-infra/modules/platform/gcp/firewall.tf`	

Description

The VPC firewall configuration is overly permissive. This configuration, in conjunction with Google Cloud's default VPC rules, allows most communication between pods (figure 17.2), the bastion host (figure 17.3), and the public internet (figure 17.1).

```
 92    resource "google_compute_firewall" "bastion_allow_all_inbound" {
 93     project = local.project
 94     name    = "${local.name_prefix}-bastion-allow-ingress"
 95
 96     network = google_compute_network.vpc.self_link
 97
 98     target_tags   = [local.tag]
 99     direction     = "INGRESS"
100     source_ranges = ["0.0.0.0/0"]
101
102     priority = "1000"
103
104     allow {
105       protocol = "all"
106     }
107    }
```

Figure 17.1: The bastion ingress rules allow incoming traffic on all protocols and ports from all addresses. (`modules/inception/gcp/bastion.tf#92–107`)

```
resource "google_compute_firewall" "intra_egress" {
 project     = local.project
 name        = "${local.name_prefix}-intra-cluster-egress"
 description = "Allow pods to communicate with each other and the master"
 network     = data.google_compute_network.vpc.self_link
 priority    = 1000
 direction   = "EGRESS"

 target_tags = [local.cluster_name]
 destination_ranges = [
   local.master_ipv4_cidr_block,
   google_compute_subnetwork.cluster.ip_cidr_range,
   google_compute_subnetwork.cluster.secondary_ip_range[0].ip_cidr_range,
 ]

 # Allow all possible protocols
 allow { protocol = "tcp" }
 allow { protocol = "udp" }
 allow { protocol = "icmp" }
 allow { protocol = "sctp" }
 allow { protocol = "esp" }
 allow { protocol = "ah" }
}
```

Figure 17.2: Pods can initiate connections to other pods on all protocols and ports. (modules/platform/gcp/firewall.tf#1–23)

```
resource "google_compute_firewall" "dmz_nodes_ingress" {
 name        = "${var.name_prefix}-bastion-nodes-ingress"
 description = "Allow ${var.name_prefix}-bastion to reach nodes"
 project     = local.project
 network     = data.google_compute_network.vpc.self_link
 priority    = 1000
 direction   = "INGRESS"

 target_tags = [local.cluster_name]
 source_ranges = [
   data.google_compute_subnetwork.dmz.ip_cidr_range,
 ]

 # Allow all possible protocols
 allow { protocol = "tcp" }
 allow { protocol = "udp" }
 allow { protocol = "icmp" }
 allow { protocol = "sctp" }
 allow { protocol = "esp" }
```

```
63     allow { protocol = "ah" }
64   }
```

Figure 17.3: The bastion host can initiate connections to pods on all protocols and ports. (modules/platform/gcp/firewall.tf#44–64)

Exploit Scenario 1

An attacker gains access to a pod through a vulnerability in an application. He takes advantage of the unrestricted egress traffic and misconfigured pods to launch attacks against other services and pods in the network.

Exploit Scenario 2

An attacker discovers a vulnerability on the Secure Shell server running on the bastion host. She exploits the vulnerability to gain network access to the Kubernetes cluster, which she can then use in additional attacks.

Recommendations

Short term, restrict both egress and ingress traffic to necessary protocols and ports. Document the expected network interactions across the components and check them against the implemented firewall rules.

Long term, use services such as the Identity-Aware Proxy to avoid exposing hosts directly to the internet, and enable VPC Flow Logs for network monitoring. Additionally, integrate automated analysis tools such as `tfsec` into the development workflow to detect firewall issues early on.

References

- Using IAP for TCP forwarding, Identity-Aware Proxy documentation

18. Lack of uniform bucket-level access in Terraform state bucket

Severity: **Informational**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-GALOY-18
Target: `galoy-infra/modules/bootstrap/gcp/tf-state-bucket.tf`	

Description
Uniform bucket-level access is not enabled in the `bootstrap` module bucket used to store the Terraform state. When enabled, this feature implements a uniform permission system, providing access at the bucket level rather than on a per-object basis. It also simplifies the access controls / permissions of a bucket, making them easier to manage and reason about.

```
1   resource "google_storage_bucket" "tf_state" {
2    name     = "${local.name_prefix}-tf-state"
3    project  = local.project
4    location = local.tf_state_bucket_location
5    versioning {
6      enabled = true
7    }
8    force_destroy = local.tf_state_bucket_force_destroy
9   }
```

Figure 18.1: The bucket definition lacks a `uniform_bucket_level_access` field set to `true`. (`modules/bootstrap/gcp/tf-state-bucket.tf#1–9`)

Exploit Scenario
The permissions of some objects in a bucket are misconfigured. An attacker takes advantage of that fact to access the Terraform state.

Recommendations
Short term, enable uniform bucket-level access in this bucket.

Long term, integrate automated analysis tools such as `tfsec` into the development workflow to identify any similar issues and areas of improvement.

19. Insecure storage of passwords

Severity: **Medium**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-GALOY-19
Target: Galoy	

Description

Galoy passwords are stored in configuration files and environment variables or passed in as command-line arguments.

There are two issues with this method of storage: (1) the default keys are low entropy (figure 19.1) and (2) the fact that there are default keys in the first place suggests that users deploying components may not realize that they need to set passwords.

```
 53    export BITCOINDRPCPASS=rpcpassword
// (...)
 68    export MONGODB_PASSWORD=password
// (...)
 79    export JWT_SECRET="jwt_secret"
```

Figure 19.1: An example configuration file with passwords (`.envrc#53–79`)

Passing in sensitive values through environment variables (figure 19.2) increases the risk of a leak for several reasons:

- Environment variables are often dumped to external services through crash-logging mechanisms.
- All processes started by a user can read environment variables from the `/proc/$pid/environ` file. Attackers often use this ability to dump sensitive values passed in through environment variables (though this requires finding an arbitrary file read vulnerability in the application).
- An application can also overwrite the contents of a special `/proc/$pid/environ` file. However, overwriting the file is not as simple as calling `setenv(SECRET, "******")`, because runtimes copy environment variables upon initialization and then operate on the copy. To clear environment variables from that special environ file, one must either overwrite the stack data in which they are located or make a low-level `prctl` system call with the `PR_SET_MM_ENV_START` and

PR_SET_MM_ENV_END flags enabled to change the memory address of the content the file is rendered from.

```
12    const jwtSecret = process.env.JWT_SECRET
```

Figure 19.2: src/config/process.ts#12

Certain initialization commands take a password as a command-line argument (figures 19.3 and 19.4). If an attacker gained access to a user account on a system running the script, the attacker would also gain access to any password passed as a command-line argument.

```
65    command: ['/bin/sh']
66    args:
67    - '-c'
68    - |
69      if [ ! -f /root/.lnd/data/chain/bitcoin/${NETWORK}/admin.macaroon ]; then
70        while ! test -f /root/.lnd/tls.cert; do sleep 1; done
71        apk update; apk add expect
72        /home/alpine/walletInit.exp ${NETWORK} $LND_PASS
73      fi
```

Figure 19.3: charts/lnd/templates/statefulset.yaml#65–73

```
55    set PASSWORD [lindex $argv 1];
```

Figure 19.4: charts/lnd/templates/wallet-init-configmap.yaml#55

In Linux, all users can inspect other users' commands and their arguments. A user can enable the proc filesystem's `hidepid=2 gid=0` mount options to hide metadata about spawned processes from users who are not members of the specified group. However, in many Linux distributions, those options are not enabled by default.

Recommendations

Short term, take the following actions:

- Remove the default encryption keys and avoid using any one default key across installs. The user should be prompted to provide a key when deploying the Galoy application, or the application should generate a key using known-good cryptographically secure methods and provide it to the user for safekeeping.
- Avoid storing encryption keys in configuration files. Configuration files are often broadly readable or rendered as such accidentally.

Long term, ensure that keys, passwords, and other sensitive data are never stored in plaintext in the filesystem, and avoid providing default values for that data. Also take the following steps:

- Document the risks of providing sensitive values through environment variables.
- Encourage developers to pass sensitive values through standard input or to use a launcher that can fetch them from a service like HashiCorp Vault.
- Allow developers to pass in those values from a configuration file, but document the fact that the configuration file should not be saved in backups, and provide a warning if the file has overly broad permissions when the program is started.

20. Third-party container images are not version pinned

Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-GALOY-20
Target: `charts/ci/pipeline.yml`	

Description

The continuous integration (CI) pipeline and Helm charts reference third-party components such as Docker registry images by named tags (or by no tag at all). Registry tags are not immutable; if an attacker compromised an image publisher's account, the pipeline or Kubernetes cluster could be provided a malicious container image.

```
87    - name: build-chain-dl-image
88      serial: true
89      plan:
90      - {get: chain-dl-image-def, trigger: true}
91      - task: build
92        privileged: true
93        config:
94          platform: linux
95          image_resource:
96            type: registry-image
97            source:
98              repository: vito/oci-build-task
```

Figure 20.1: A third-party image referenced without an explicit tag (ci/pipeline.yml#87–98)

```
270   resource_types:
271   - name: terraform
272     type: docker-image
273     source:
274       repository: ljfranklin/terraform-resource
275       tag: latest
```

Figure 20.2: An image referenced by the "latest" tag (ci/pipeline.yml#270–275)

Exploit Scenario

An attacker gains access to a Docker Hub account hosting an image used in the CI pipeline. The attacker then tags a malicious container image and pushes it to Docker Hub. The CI pipeline retrieves the tagged malicious image and uses it to execute tasks.

Recommendations

Short term, refer to Docker images by SHA-256 digests to prevent the use of an incorrect or modified image.

Long term, integrate automated tools such as `Checkov` into the development workflow to detect similar issues in the codebase.

21. Compute instances do not leverage Shielded VM features

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-GALOY-21
Target: `galoy-infra/modules/inception/gcp/bastion.tf`	

Description
The bastion host definition does not enable all of Google Cloud's Shielded VM (virtual machine) features for Compute Engine VM instances. These features provide verifiable integrity of VM instances and assurance that VM instances have not been compromised by boot- or kernel-level malware or rootkits.

Three features provide this verifiable integrity: Secure Boot, virtual trusted platform module (vTPM)-enabled Measured Boot, and integrity monitoring.

Google also offers Shielded GKE nodes, which are built on top of Shielded VMs and provide strong verifiable node identity and integrity to increase the security of GKE nodes. The node pool definition does enable this feature but disables Secure Boot checks on the node instances.

```
168    shielded_instance_config {
169     enable_secure_boot          = false
170     enable_integrity_monitoring = true
171    }
```

Figure 21.1: Secure Boot is disabled. (`modules/platform/gcp/kube.tf#168–171`)

Exploit Scenario
The bastion host is compromised, and persistent kernel-level malware is installed. Because the bastion host is still operational, the malware remains undetected for an extended period.

Recommendations
Short term, enable these security features to increase the security and trustworthiness of the infrastructure.

Long term, integrate automated analysis tools such as `tfsec` into the development workflow to detect other areas of improvement in the solution.

References

- What is Shielded VM?, Compute Engine documentation
- Using GKE Shielded Nodes, GKE documentation

22. Excessive container permissions

Severity: **Low**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-GALOY-22
Target: `charts`	

Description

Kubernetes containers launch processes under user and group IDs corresponding to users and groups on the host system. Container processes that are running as root usually have more permissions than their workload requires. If such a process were compromised, the permissions would enable the attacker to perform further attacks against the container or host.

Kubernetes provides several ways to further limit these permissions, such as disabling the `allowPrivilegeEscalation` flag to ensure that a child process of a container cannot gain more privileges than its parent, dropping all Linux capabilities, and enforcing Seccomp and AppArmor profiles.

We found several instances of containers run as root, with `allowPrivilegeEscalation` enabled by omission (figure 22.1) or with low user IDs that overlap with host user IDs (figure 22.2). In some of the containers, Linux capabilities were not dropped (figure 22.2), and neither Seccomp nor AppArmor profiles were enabled.

```
24    containers:
25    - name: auth-backend
26      image: "{{ .Values.image.repository }}@{{ .Values.image.digest }}"
27      ports:
28      - containerPort: 3000
29      env:
30      - (...)
```

Figure 22.1: Without a `securityContext` field, commands will run as root and a container will allow privilege escalation by default.
(charts/galoy-auth/charts/auth-backend/templates/deployment.yaml#24–30)

```
38    securityContext:
39     # capabilities:
40     #   drop:
41     #   - ALL
42     readOnlyRootFilesystem: true
43     runAsNonRoot: true
44     runAsUser: 1000
45     runAsGroup: 3000
```

Figure 22.2: User ID 1000 is typically used by the first non-system user account. (charts/bitcoind/values.yaml#38–45)

Exploit Scenario

An attacker is able to trigger remote code execution in the Web Wallet application. The attacker then leverages the lax permissions to exploit CVE-2022-0185, a buffer overflow vulnerability in the Linux kernel that allows her to obtain root privileges and escape the Kubernetes pod. The attacker then gains the ability to execute code on the host system.

Recommendations

Short term, review and adjust the `securityContext` configuration of all charts used by the Galoy system. Run pods as non-root users with high user IDs that will not overlap with host user IDs. Drop all unnecessary capabilities, and enable security policy enforcement when possible.

Long term, integrate automated tools such as Checkov into the CI pipeline to detect areas of improvement in the solution. Additionally, review the Docker recommendations outlined in appendix E.

References

- Kubernetes container escape using Linux Kernel exploit, CrowdStrike
- 10 Kubernetes Security Context settings you should understand, snyk

23. Unsigned and unversioned Grafana BigQuery Datasource plugin

Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-GALOY-23
Target: `charts/charts/monitoring/values.yaml`	

Description

The `BigQuery Datasource` plugin is installed as part of the Grafana configuration found in the Helm charts. The plugin, which is unsigned, is pulled directly from the master branch of the `doitintl/bigquery-grafana` GitHub repository, and signature checks for the plugin are disabled. Grafana advises against running unsigned plugins.

```
10    plugins:
11     -
https://github.com/doitintl/bigquery-grafana/archive/master.zip;doit-bigquery-dataso
urce
12
13    grafana.ini:
14     plugins:
15       allow_loading_unsigned_plugins: "doitintl-bigquery-datasource"
```

Figure 23.1: The plugin is downloaded directly from the GitHub repository, and signature checks are disabled. (`charts/monitoring/values.yaml#10–15`)

Exploit Scenario

An attacker compromises the `doitintl/bigquery-grafana` repository and pushes malicious code to the master branch. When Grafana is set up, it downloads the plugin code from the master branch. Because unsigned plugins are allowed, Grafana directly loads the malicious plugin.

Recommendations

Short term, install the `BigQuery Datasource` plugin from a signed source such as the Grafana catalog, and disallow the loading of any unsigned plugins.

Long term, review the vendor recommendations when configuring new software and avoid disabling security features such as signature checks. When referencing external code and software releases, do so by immutable hash digests instead of named tags or branches to prevent unintended modifications.

References

- Plugin Signatures, Grafana Labs

24. Insufficient validation of JWTs used for GraphQL subscriptions

Severity: **Low**	Difficulty: **Low**
Type: Authentication	Finding ID: TOB-GALOY-24
Target: `galoy/src/servers/graphql-server.ts`	

Description

The GraphQL API uses JWTs to implement access controls on subscriptions, which allow clients to receive pushed updates about certain changes from the server. In the Galoy application, clients use subscriptions mainly to receive updates on Lightning invoice statuses and asset prices.

Upon receiving a connection request, the application decodes the token included in the request but does not verify whether a trusted party has signed it. Without this verification, the application could accept a forged token. Moreover, because there is no verification of the authentication scheme, the application may accept requests with strings other than `Bearer`.

```
309    async onConnect(connectionParams, webSocket, connectionContext) {
310      const { request } = connectionContext
311
312      let token: string | jwt.JwtPayload | null = null
313      const authz =
314        connectionParams.authorization || connectionParams.Authorization
315      if (authz) {
316        const rawToken = authz.slice(7)
317        token = jwt.decode(rawToken)
318      }
319
320      return sessionContext({
321        token,
322        ip: request?.socket?.remoteAddress,
323
324        // TODO: Resolve what's needed here
325        apiKey: null,
326        apiSecret: null,
327        body: null,
328      })
329    },
```

Figure 24.1: The token is decoded but not verified.
(`src/servers/graphql-server.ts#309–329`)

Exploit Scenario

An attacker creates a properly formatted HS256 JWT that references Bob's user ID (`uid`) and signs it with her key. She can then use the token to impersonate Bob and to subscribe to status changes in the GraphQL API.

Recommendations

Short term, update the GraphQL implementation to verify JWT signatures before decoding JWTs.

Long term, expand the test suite to cover authorization checks.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Outdated Dependencies

This appendix lists certain of the vulnerable dependencies and components referenced in TOB-GALOY-11. For the sake of concision, it includes only those dependencies and components with vulnerabilities of at least moderate severity. The Galoy developers should not use this filtering (`--level moderate`) when using the `yarn audit` tool. Developers should use updated versions of these dependencies wherever possible.

```
$ yarn audit --groups dependencies --json --level moderate | npx compact-yarn-audit
```

Figure C.1: The command used to generate these results

github.com/GaloyMoney/admin-panel				
Severity	**Title**	**Module**	**Via**	**Fixed In**
Critical	Prototype Pollution	`immer`	`react-dev-utils`	`immer: ">=9.0.6"`
Critical	Authorization Bypass	`url-parse`	`webpack-dev-server`	`url-parse: ">=1.5.8"`
High	Uncontrolled Resource consumption	`ansi-html`	`@pmmmwh/react-refresh-webpack-plugin`	`ansi-html: ">=0.0.8"`
High	Potential memory exposure	`dns-packet`	`webpack-dev-server`	`dns-packet: ">=1.3.2"`
High	Exposure of sensitive info	`follow-redirects`	`webpack-dev-server`	`follow-redirects: ">=1.14.7"`
High	Regular expression DoS	`glob-parent`	`webpack`	`glob-parent: ">=5.1.2"`
High	Regular expression DoS	`glob-parent`	`webpack-dev-server`	`glob-parent: ">=5.1.2"`
High	Prototype Pollution	`immer`	`react-dev-utils`	`immer: ">=9.0.6"`
High	Regular Expression DoS	`is-svg`	`optimize-css-assets-webpack-plugin`	`is-svg: ">=4.2.2"`
High	ReDOS in IS-SVG	`is-svg`	`optimize-css-assets-webpack-plugin`	`is-svg: ">=4.3.0"`
High	Regular Expression DoS	`ssri`	`webpack`	`ssri: ">=6.0.2"`
High	Arbitrary File Creation	`tar`	`terser-webpack-plugin`	`tar: ">=6.1.2"`
High	Arbitrary File Creation	`tar`	`terser-webpack-plugin`	`tar: ">=6.1.1"`
High	Arbitrary File Creation	`tar`	`terser-webpack-plug`	`tar: ">=6.1.7"`

			in	
High	Arbitrary File Creation	tar	terser-webpack-plugin	tar: ">=6.1.9"
High	Arbitrary File Creation	tar	terser-webpack-plugin	tar: ">=6.1.9"

github.com/GaloyMoney/galoy

Severity	Title	Module	Via	Fixed In
Critical	Type confusion in mpath	mpath	medici	mpath: ">=0.8.4"
Critical	Type confusion in mpath	mpath	mongoose	mpath: ">=0.8.4"
Critical	Authorization Bypass	url-parse	twilio	url-parse: ">=1.5.8"
High	Server-Side Request Forgery	axios	gt3-server-node-express-sdk	axios: ">=0.21.1"
High	Incorrect Comparison	axios	gt3-server-node-express-sdk	axios: ">=0.21.2"
High	Exposure of sensitive info	follow-redirects	gt3-server-node-express-sdk	follow-redirects: ">=1.14.7"

github.com/GaloyMoney/web-wallet

Severity	Title	Module	Via	Fixed In
Moderate	Improper CSP	next	@ory/integrations	next: ">=12.1.0"
Moderate	Regular Expression DoS	postcss	@ory/themes	postcss: ">=7.0.36"

D. Identified Hosts

This appendix lists the hosts and subdomains directly linked to the galoy.io domain. We did not review them closely but are providing them as preliminary information. We used the DNS dumpster tool, which uses a variety of open-source information-gathering methods to generate subdomain lists.

We also created a high-resolution graph of the hosts and subdomains identified for galoy.io.

Identified Subdomains of the galoy.io Domain		
*.mobilewallet.staging.galoy.io	hub.freecorn.galoy.io	www.admin.staging.galoy.io
admin-api.galoy-testnet.galoy.io	hub.hackathon.galoy.io	www.galoy.io
admin-api.mainnet.galoy.io	hub.mainnet.galoy.io	www.grafana.freecorn.galoy.io
admin-api.staging.galoy.io	hub.staging.galoy.io	www.grafana.hackathon.galoy.io
admin-api.testnet.galoy.io	ln.bitcoinbeach.galoy.io	www.grafana.staging.galoy.io
admin.bitcoinbeach.galoy.io	ln.bitcoinbeach.testnet.galoy.io	www.graphql.mainnet.galoy.io
admin.bitcoinbeach.testnet.galoy.io	lnpage.galoy-bitcoinbeach.galoy.io	www.graphql.testnet.galoy.io
admin.freecorn.galoy.io	lnpage.galoy-testnet.galoy.io	www.hub.hackathon.galoy.io
admin.galoy-bitcoinbeach.galoy.io	lnpage.mainnet.galoy.io	www.lnpage.galoy-bitcoinbeach.galoy.io
admin.galoy-testnet.galoy.io	lnpage.staging.galoy.io	www.mobilewallet.staging.galoy.io
admin.mainnet.galoy.io	lnpage.testnet.galoy.io	www.pay.staging.galoy.io
admin.staging.galoy.io	lnpay.mainnet.galoy.io	www.revised.galoy.io
api.freecorn.galoy.io	lnpay.testnet.galoy.io	www.specter.freecorn.galoy.io
api.mainnet.galoy.io	mobilewallet.staging.galoy.io	www.specter.mainnet.galoy.io
api.staging.galoy.io	pay.freecorn.galoy.io	www.specter.testnet.galoy.io
api.testnet.galoy.io	pay.mainnet.galoy.io	www.tips.testnet.galoy.io
auth.staging.galoy.io	pay.staging.galoy.io	www.www.revised.galoy.io
billpay.mainnet.galoy.io	pay.testnet.galoy.io	
bitcoinbeach.testnet.galoy.io	rev.galoy.io	
ci.galoy.io	revised.galoy.io	
galoy.io	specter.freecorn.galoy.io	
get.galoy.io	specter.galoy-bitcoinbeach.galoy.io	
grafana.freecorn.galoy.io	specter.mainnet.galoy.io	
grafana.galoy-bitcoinbeach.galoy.io	specter.staging.galoy.io	
grafana.galoy.io	specter.testnet.galoy.io	
grafana.hackathon.galoy.io	tips.mainnet.galoy.io	
grafana.mainnet.galoy.io	tips.testnet.galoy.io	
grafana.staging.galoy.io	try.galoy.io	
grafana.testnet.galoy.io	www.admin-api.galoy-testnet.galoy.io	
graphql-admin.mainnet.galoy.io	www.admin-api.mainnet.galoy.io	
graphql-admin.testnet.galoy.io	www.admin-api.staging.galoy.io	
graphql.mainnet.galoy.io	www.admin.mainnet.galoy.io	
graphql.staging.galoy.io		
graphql.testnet.galoy.io		

E. Docker Security Recommendations

This appendix provides general recommendations regarding the use of Docker. We suggest following the guidance included in the "Basic Security" and "Limiting Container Privileges" sections and reviewing the list of options to avoid. This appendix also describes the Linux features that form the basis of Docker container security measures and includes a list of additional references.

Basic Security

- Do not add users to the `docker` group. Inclusion in the `docker` group allows a user to escalate his or her privileges to root without authentication.

- Do not run containers as a root user. If user namespaces are not utilized, the root user within the container will be the real root user on the host. Instead, create another user within the Docker image and set the container user by leveraging the USER instruction in the image's Dockerfile specification. Alternatively, pass in the `--user $UID:$GID` flag to the `docker run` command to set the user and user group.

- Do not use the `--privileged` flag. Using this flag allows the process within the container to access all host resources, hijacking the machine.

- Do not mount the Docker daemon socket (usually `/var/run/docker.sock`) into the container. A user with access to the Docker daemon socket will be able to spawn a privileged container to "escape" the container and access host resources.

- Carefully weigh the risks inherent in mounting volumes from special filesystems such as `/proc` or `/sys` into a container. If a container has write access to the mounted paths, a user may be able to gain information about the host machine or escalate his own privileges.

Limiting Container Privileges

- Using the `--cap-add=...` flag, pass the `--cap-drop=all` flag to the `docker run` command to drop all Linux capabilities and enable only those necessary to the process within a container. Note, though, that adding capabilities could allow the process to escalate its privileges and "escape" the container.

- Pass the `--security-opt=no-new-privileges:true` flag to the `docker run` command to prevent processes from gaining additional privileges.

- Limit the resources provided to a container process to prevent denial-of-service scenarios.

- Do not use root (`uid=0` or `gid=0`) in a container if it is not needed. Use `USER ...` in the Dockerfile (or use `docker run --user $UID:$GID ...`).

The following recommendations are optional:

- Use user namespaces to limit the user and group IDs available in the container to only those that are mapped from the host to the container.
- Adjust the Seccomp and AppArmor profiles to further limit container privileges.
- Consider using SELinux instead of AppArmor to gain additional control over the operations a given container can execute.

Options to Avoid

Flag	Description
`--privileged`	A flag that "removes ALL security"
`--cap-add=all`	Adds all Linux capabilities
`--security-opt apparmor=unconfined`	Disables AppArmor
`--security-opt seccomp=unconfined`	Disables Seccomp
`--device-cgroup-rule='a *:* rwm'`	Enables access to all devices (according to this documentation)
`--pid=host`	Uses host pid namespace
`--uts=host`	Uses host uts namespace
`--network=host`	Uses host network namespace, which grants access to all network interfaces available on a host

Linux Features Foundational to Docker Container Security

Feature	Description
Namespaces	This feature is used to isolate or limit the view (and therefore the use) of a global system resource. There are various namespaces, such as PID, network, mount, UTS, IPC, user, and cgroup, each of which wraps a different resource. For example, if a process creates a new PID namespace, the process will act as if its PID=1 and will not be able to send signals to processes created in its parent namespace. The namespaces to which a process belongs are listed in the `/proc/$PID/ns/` directory (each with its own ID) and can also be accessed by using the `lsns` tool.
Control groups (cgroups)	This is a mechanism for grouping processes/tasks into hierarchical groups and metering or limiting resources within those groups, such as memory, CPUs, I/Os, or networks. The cgroups to which a process belongs can be read from the `/proc/$PID/cgroup` file. A cgroup's entire hierarchy will be indicated in a `/sys/fs/cgroup/<cgroup controller` or `hierarchy>/` directory if the cgroup controllers are mounted in that directory. (Use the `mount | grep cgroup` command to see whether they are.) There are two versions of cgroups, cgroups v1 and cgroups v2, which can be (and often are) used at the same time.
Linux capabilities	This feature splits root privileges into "capabilities." Although this setting is primarily related to the actions a privileged user can take, there are different process-capability sets, some of which are used to calculate the user's effective capabilities (such as after running a suid binary). As such, dropping all Linux capabilities from all capability sets will help prevent a process from gaining additional privileges (such as through suid binaries). The Linux process-capability sets for a given process can be read from the `/proc/$PID/status` file, specifically its `CapInh`, `CapPrm`, `CapEff`, `CapBnd`, and `CapAmb` values (which correspond to the inherited, permitted, effective, bound, and ambient capability sets, respectively). Those values can be decoded into meaningful capability

	names by using the `capsh --decode=$VALUE` tool.
The "no new privileges" flag	Enabling this flag for a process will prevent the user who launched the process from gaining additional privileges (such as through suid binaries).
Seccomp BPF syscall filtering	Seccomp BPF enables the filtering of arguments passed in to a program and the syscalls executed by it. It does this by writing a "BPF program" that is later run in the kernel. Refer to the Docker default Seccomp policy. One can write a similar profile and apply it with the `--security-opt seccomp=<file>` flag.
AppArmor Linux Security Module (LSM)	AppArmor is a Linux Security Module that limits a container's access to certain resources by enforcing a mandatory access control. AppArmor profiles are loaded into a kernel. A profile can be in either "complain" or "enforce" mode. In "complain" mode, violation attempts are logged only into the syslog; in "enforce" mode, such attempts are blocked. To see which profiles are loaded into a kernel, use the aa-status tool. To see whether a given process will work under the rules of an AppArmor profile, read the `/proc/$PID/attr/current` file. If AppArmor is not enabled for the process, the file will contain an "`unconfined`" value. If it is enabled, the file will return the name of the policy and its mode (e.g., "`docker-default (enforce)`"). Refer to the Docker AppArmor profile template and the generated form of the profile.

Additional References

- Understanding Docker Container Escapes: A Trail of Bits blog post that breaks down a container escape technique and explains the constraints required to use that technique
- Namespaces in Operation, Part 1: Namespaces Overview: A seven-part LWN article that provides an overview of Linux namespace features

- False Boundaries and Arbitrary Code Execution: An old but thorough post about Linux capabilities and the ways that they can be used in privilege escalation attempts
- Technologies for Container Isolation: A Comparison of AppArmor and SELinux: A comparison of AppArmor and SELinux