Infinigold Pty Ltd

Offchain Services Security Review

Version: 1.0

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the software and infrastructure supporting the InfiniGold token, a gold-backed stablecoin powered by Ethereum. The system is designed to work within Amazon Web Services (AWS) infrastructure to provide extended functionality to the GoldPass mobile application.

This review focuses on two different system components:

- **Kotlin services**
  Software written using Kotlin, designed to work on AWS Lambda.
- **AWS infrastructure**
  Configuration of various infrastructure components on AWS.

This review focused exclusively on the security aspects of the Kotlin services and AWS infrastructure (see Scope of Review), even though a couple of issues related to additional out-of-scope components (such as the authentication mechanism with the GoldPass API) are also raised in this report.

Overview

InfiniGold allows investors to buy, sell and hold physical gold stored at The Perth Mint - a large refining mint - in a digital form. To support this gold-backed stable coin, InfiniGold has developed a web application that extends the GoldPass mobile application with the following features:

- Prove ownership of an Ethereum address, and associate it with a GoldPass user account;
- Whitelist a user's Ethereum address on the public Ethereum blockchain;
- Allow a user to remove or replace their whitelisted Ethereum address;
- Convert GoldPass certificates into ERC20 tokens on Ethereum;
- Convert ERC20 tokens, which are burned, into GoldPass certificates.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the system. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the InfiniGold system within the scope of the security review. A summary is provided followed by a detailed review of the discovered vulnerabilities which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an open/closed/resolved status, and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as "informational".

Outputs of automated testing that were developed during this assessment are also included for reference, and accompany this report.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the InfiniGold system.

Security Assessment Summary

Scope of Review

Four (4) assets are identified in this report:

- **InfiniGold API Gateway**
  Services which comprise the majority of the business logic behind the InfiniGold system.
- **InfiniGold Front End**
  Static HTML and JavaScript assets that form the front-end interface for the user.
- **AWS Infrastructure**
  Web application infrastructure and configuration using Amazon Web Services.
- **GoldPass API**
  Existing GoldPass infrastructure.
  (out of scope)

This review was conducted against four encrypted ZIP archives[1] provided by the development team, which have the following SHA256 hashes:

```
pmgt-services-release_1.210.0@faad39b3d:   c15055241dcc85c604cdc1d203041852476d609bc8037b9b8cb554294250aeef
aws-deploy-tools-release_2.20.0@4c699fb:   4c64cd1bacab8aaf6fd72aa1fc2e505e0d720894753368e0663225b01f893446
                 dependencies-20190515:   50db1d0cc925e23763c91078e37df40779f80e3835d7fd0337fd8bf3d9b3a462
```

The following is a non-exhaustive list of the AWS services within scope of this review, as was deployed to a staging environment within AWS (labelled `pmgs-sigma`):

- **IAM**
- **Lambda**
  - `address-service`
  - `whitelist-updater`
  - `token-minter`
  - `whitelist-events-consumer`
  - `mint-validator`
  - `mint-creator`
  - `mint-events-consumer`
  - `burn-events-consumer`

[1] this excludes the additional key material, which was also provided in ZIP archives.

  - `event-poller`

- **DynamoDB**
  - `whitelister-nonce`
  - `token-mint`
  - `token-burn-transaction-id`
  - `token-burn`
  - `minter-nonce`
  - `last-seen-block`
  - `investor-ethereum-address`
  - `ethereum-address-verification-challenge`
- **API Gateway**
  - `/api/address/{proxy+}`
  - `/static/{service}/{key+}`
- **SQS Queue**
  - `address-whitelist-queue`
  - `burn-event-queue`
  - `mint-creator-queue`
  - `mint-event-queue`
  - `token-minter-queue`
  - `whitelist-updater-queue`
- **SNS Topic**
  - `contract-events-topic`
  - `operator-notifications-topic`
- **SES**
  - `do-not-reply@dev.infinigoldnotlive.com`

Approach

Requests made with the GoldPass mobile application were intercepted, manipulated, and examined to determine operation of the system. These requests then were imitated using an interactive python notebook which was subsequently utilised for extensive testing [2].

Correct operation was verified by examining logs available via Amazon CloudWatch, and by observing transactions on the live Ropsten Ethereum test network.

The Kotlin source code for the Lambda services was reviewed manually, supported by static source code analysis tools such as detekt.

AWS configuration was inspected manually by means of the AWS interactive web console. Automated scans which utilise the AWS API, such as CloudSploit Scans, were also used.

[2] The output of this testing is provided alongside this report, and an interactive version is available upon request.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the course of this review. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

The testing team identified two medium severity, three low severity, and a range informational issues. Informational issues are grouped into a single vulnerability listing per asset, as are AWS hardening issues.

Each vulnerability is assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected system(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
IOS-01	Replay of Request Triggers Repeat E-mail	Medium	Open
IOS-02	Authentication Token Exposure	Low	Open
IOS-03	Insecure Cookie Handling	Low	Open
IOS-04	API Does not Enforce Strict Transport Security (HSTS)	Low	Open
IOS-05	AWS Configuration Hardening	Low	Open
IOS-06	API Gateway - Informational Issues	Informational	Open
IOS-07	GoldPass API - Informational Issues	Informational	Open

IOS-01	Replay of Request Triggers Repeat E-mail		
Asset	GoldPass API		
Status	**Open**		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Please note: This issue relates to the GoldPass API, which is not in scope for this review.

Description

When transferring GoldPass certificates, i.e. transferring gold, an API call is made to the mint with a `transaction_id` parameter, set by the client. This is typically set to the current UNIX time and is intended to differentiate between repeated, otherwise identical transactions.

If this value is overridden so that exactly the same request is repeated, the e-mail sent to the recipient informing them that they have received funds is sent a second time. This remains the case even if the repeated API call is made several days after the other.

This could be misleading to GoldPass users and could be used in phishing attacks whereby users believe they are sent the same amount of gold more than once.

To illustrate, assume a victim is selling Widgets. An attacker could make payment for a single Widget to a victim, which the victim would mail to the attacker. The attacker then requests that a second Widget be sent to them at the same cost, but instead of transferring a second payment, the attacker just repeats the previous API call. This triggers a repeat e-mail to be sent, potentially misleading the victim into thinking they have been paid again when they in fact have not.

Note that submitting a second transfer request with the same `transaction_id` but a different value in another field (e.g. amount), will actually cause a second transfer to occur. In this case both of these transfers will have the same supplied `transaction_id`, which is misleading since `transaction_id` is not a unique identifier as the name implies.

Recommendations

Only send an e-mail upon a new transfer, and reject repeat requests (i.e. prevent replay attacks).

We suggest one of two solutions to remediate confusion regarding the purpose of `transaction_id`:

- Resubmitting any request with a repeated `transaction_id` should not cause any action, even if other fields have changed. In this case an error should be thrown.
- Rename the `transaction_id` parameter to `nonce`, which does not imply uniqueness.

IOS-02	Authentication Token Exposure		
Asset	InfiniGold Front End		
Status	Open		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The front-end "PGMT Address" interface is loaded from the path `/static/address/index.html`, hosted by the API Gateway, using a HTTP GET request. The JavaScript which executes within this page relies on the presence of a HTTP GET parameter named `authToken`, which is an authentication token provided by the GoldPass app.

In loading the interface, the GoldPass app submits a HTTP GET request is to the API Gateway with the `authToken` parameter included. As such, the `authToken` is unnecessarily exposed to the `/static` endpoint in the gateway.

Please note that this request could not be intercepted during testing due to the way certificates are handled within the mobile application. For testing, it would be necessary to modify the mobile App itself to bypass these checks and intercept the request. Since the mobile App is not within scope of this review, this issue has not been fully reproduced and is based on static analysis of the "PGMT Address" feature.

This issue relates to, but is distinct from, issue IOS-03 in which the `authToken` is also exposed via the cookie.

Recommendations

Consider Passing the `authToken` parameter to the interface's JavaScript directly (for example by using a web-view's `localStorage`), without including it as a HTTP GET parameter.

IOS-03	Insecure Cookie Handling		
Asset	InfiniGold Front End		
Status	Open		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The front-end interface is loaded using a HTTP GET parameter named `authToken`, which is a token provided by the GoldPass API subsequently used for authentication.

The server does not authenticate requests using the cookie, but instead uses a HTTP header named `Authorization` which includes the authentication token.

On line [6] of `address.js`, the `authToken` is saved to the browser's cookies, which is an unnecessary exposure of the authentication token. Further, cookies can be saved with additional flags that instruct the browser on how it should be used.

The cookie is saved without the secure flag set and so in the case where a browser is manipulated into making an insecure HTTP request, the authentication cookie will be included in the request headers.

The severity of this issue is exacerbated by the lack of a configured HSTS header, as detailed in IOS-04.

This issue relates to IOS-02, which also exposes `authToken` via the HTTP GET parameters.

Recommendations

When saving the cookie, set the secure flag to true.

Alternatively, since the cookie is never read by the server, the authentication token could instead be saved to local storage. This would prevent it from being sent with the request automatically, while still allowing the JavaScript to access it when making API calls.

IOS-04	API Does not Enforce Strict Transport Security (HSTS)		
Asset	InfiniGold API Gateway		
Status	**Open**		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The InfiniGold API fails to include the *HSTS* response header.

HTTP Strict Transport Security, or *HSTS*, is a protection mechanism that instructs browsers to solely communicate over HTTPS connections and prevent users from accepting invalid certificates.

An attacker who is able to modify a legitimate user's network traffic could bypass the application's use of SSL/TLS encryption and use it as a platform for attacks against its users. This attack is performed by rewriting HTTPS links as HTTP, so that if a targeted user's browser does not attempt to use an encrypted connection.

The lack of HSTS can also be exploited by presenting invalid certificates to users, who may accept it, enabling man-in-the-middle attacks using tools and techniques such as `SSLStrip`. We note that this vulnerability is mitigated by the way in which the GoldPass mobile app. handles certificates.

This vulnerability exacerbates the severity of the authentication cookie in issue IOS-03.

Recommendations

Enable HTTP Strict Transport Security (HSTS) by adding a HTTP response header labelled `Strict-Transport-Security`, with a value of `'max-age=expireTime'`, where `expireTime` is the time in seconds that browsers should remember that the site should only be accessed using HTTPS. We recommend this value be set to `31536000` seconds (one year).

IOS-05	AWS Configuration Hardening		
Asset	AWS Infrastructure		
Status	Open		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

1. **MFA not enforced for all users.**
   Multi-factor authentication (MFA) is not enabled for the user account `bgardiner`, this should not be possible as MFA should always be enforced.
2. **Weak password policies.**
   The password policies for accounts in Amazon's IAM are not strict enough, as would be appropriate for an account with administrator access to the system:
   - Minimum password length is too short, should be at least 14 characters;
   - Passwords do not expire and so regular rotation is not enforced;
   - The password policy does not require a varied selection of character groups (e.g. symbols, numbers, upper/lowercase);
   - The password policy does not prevent reuse of previous passwords.
3. **AWS Config service not enabled.**
   The AWS Config service records the configuration of AWS resources and maintains a history of changes. This is useful for compliance and is invaluable for performing forensics after a security incident.
4. **CloudTrail is not integrated with CloudWatch**
   Amazon CloudTrail records information about various actions performed on an AWS account. While these actions are currently being logged to an S3 bucket, the logs should also be integrated with Amazon CloudWatch so that suspicious activity will trigger alerts and system notifications. For this to be useful, appropriate metrics must be configured in CloudWatch which trigger notifications on events such as network ACL changes, policy changes, and sign-in failures.
5. **S3 bucket logging and versioning not enabled**
   Should a security incident occur, the forensic data collected from S3 bucket logging and the version history of files stored in the bucket could be invaluable.
6. **SQS does not use encryption**
   The Amazon Simple Queue Service can be configured to use keys stored in Amazon's KMS, which enables data to be encrypted while they exist in queues. This prevents an attacker who is able to read the contents of queues from deciphering the messages.
7. **SNS does not use encryption**
   The Amazon Simple Notification Service can be configured to use keys stored in the Amazon's KMS, which enables data to be encrypted at rest. This prevents an attacker who is able to read the SNS data stores from learning the detailed operation of the system.
8. **AWS KMS key rotation not enabled**
   Amazon's Key Management Service (KMS) has an option to perform automatic yearly key rotation, which is currently not enabled.
9. **Nonrestrictive SNS default policy**
   The access policy on both topics in Amazon's Simple Notification Service (SNS) grants permissions to all objects under the same account as the topic, including all users, roles, services. This is mitigated by the

fact that IAM permissions of users must also explicitly provide access (both the topic policy and IAM policy must allow access), however applying a stricter access policy to the SNS directly would improve security.

10. **Dead Letter Queues not configured**
    AWS has a feature called Dead Letter Queue (DLQ) for SNS and Lambda services. Any item which cannot be processed, or causes an error state, will be saved to the DLQ for examination. If an error occurs within the InfiniGold infrastructure, it is essential that the offending data be captured for debugging purposes. This may also serve as a useful audit trail, should a security incident occur.

Recommendations

1. Ensure all users are configured with MFA.
2. Increase the requirements on passwords so that they:
    - are at least 14 characters;
    - passwords expire every 3-12 months;
    - passwords contain characters from all groups;
    - previously used passwords are not permitted.
3. Configure the AWS Config service.
4. Integrate Amazon CloudTrail with CloudWatch, and configure metrics so that notifications are raised upon suspicious activity.
5. Enable logging and versioning of Amazon S3 buckets.
6. Server-side encryption should be enabled for SQS, using keys in KMS.
7. Server-side encryption should be enabled for SNS, using keys in KMS.
8. Enable automatic key rotation in KMS.
9. Apply a more restrictive access policy to both SNS topics as appropriate.
10. Enable error logging using a DLQ resource in AWS Lambda and the redrive policy in SQS.

IOS-06	API Gateway - Informational Issues
Asset	InfiniGold API Gateway
Status	Open
Rating	Informational

Description

The following informational issues were identified

1. **API Gateway not validated by client certificates.**
   The API Gateway does not utilise client certificates, and so the requests made to the pmgts-address-service are not authenticated and are more likely to suffer a MitM attack between the API gateway and address service.
2. **DMARC policy set to 'none'.**
   The DMARC policy for both `dev.infinigoldnotlive.com` and `infinigoldnotlive.com` (the latter would be used) do not specify that e-mails which fail the DMARC checks should either be quarantined or rejected; instead no action is taken.[3] As such, an attacker may be able to trick a user into signing an alternative challenge, should they send a well-timed challenge e-mail which spoofs the InfiniGold from address.
3. **No DKIM signing of challenge email.**
   DKIM signing is not enabled for the e-mail address which sends the challenge e-mail addresses. While DKIM is present automatically for Amazon SES, it is not present for the `dev.infinigoldnotlive.com` domain. This increases the likelihood that InfiniGold e-mail addresses can be spoofed.
4. **Old version of web3j.**
   The version of web3j that is specified in the application is `v4.1.1` (released 20th January), however the latest version is `4.3.0` (released 9th May). While there does not appear to be any security relevant improvements between these versions, it is best practice to use the latest version.

Description

We recommend implementing the following changes:

1. Install client certificates into the API gateway configuration and validate that incoming requests are from the expected certificate in the address service's Kotlin code.
2. Configure the DMARC policy to `p=quarantine` or `p=reject`.
3. Configure DKIM signing in Amazon SES and update appropriate DNS records
4. Upgrade the dependency on web3j to use the latest version.

[3] For more information on the DMARC records, see: https://dmarcian.com/dmarc-inspector/?domain=infinigoldnotlive.com.

IOS-07	GoldPass API - Informational Issues
Asset	GoldPass API
Status	Open
Rating	Informational

Please note: These issues relate to the GoldPass API, which is not in scope for this review.

Description

The following informational issues were identified

1. **Inconsistent response to authentication request with invalid agent.**
   When requesting an authentication token from the GoldPass API, using an invalid "agent" field, the HTTP response status code is `400`, however the error message returned is `InternalError`. This status code and error message are inconsistent, since HTTP `400` codes indicate an invalid request, whereas `InternalError` would suggest a HTTP `5xx` status code.
2. **Requesting very long token expiry causes HTTP `500` internal error.**
   Consumers of the authentication API can request that their token expire after a specified delay. The maximum permitted delay is 60 minutes, so if an expiry delay of greater than 60 minutes is requested then the returned expiry is only 60 minutes. However, if an expiry of more than 24 days is requested, then a HTTP `500` status code, indicating a server error, is returned.
3. **No 'logout' method**
   After an authentication token is issued, there's no way (identified in this review) to revoke the token. If a token was known to be compromised for some reason, it would be best to provide a way in which it could be invalidated. This issue is mitigated by the fact that the maximum expiry of the token is only 60 minutes.

Recommendations

Consider implementing the following changes:

1. Change the `InternalError` text to `RequestError` or `InvalidAgent`.
2. Modify the returned expiry date to be consistent, regardless of what input value is provided.
3. Provide a method by which an authentication token can be invalidated.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact			
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium
	Low	Medium	High
		Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.