Unlockd.Finance - LTV-Bot

GitHub Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 13th, 2022 - October 17th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/17/2022	Connor Taylor
0.2	Document Updated	10/19/2022	Connor Taylor
0.3	Draft Review	10/21/2022	Carlos Polop
0.4	Draft Review	10/21/2022	Gabi Urrutia
0.5	Draft Update	11/17/2022	Connor Taylor
0.6	Draft Update Review	11/17/2022	Carlos Polop
0.7	Draft Update Review	11/18/2022	Gabi Urrutia
1.0	Remediation Plan	01/09/2023	Connor Taylor
1.1	Remediation Plan Review	01/11/2023	Carlos Polop

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Connor Taylor	Halborn	Connor.Taylor@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Unlockd.Finance engaged Halborn to conduct a security audit on their smart contracts beginning on October 13th, 2022 and ending on October 17th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided one week for the engagement and assigned a full-time security engineer to audit the security of the application and container. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that application functions operate as intended
- Identify potential security issues within the application code and the containerisation deployment

In summary, Halborn identified some security risks relating to the docker deployment that were successfully addressed by the Unlockd.Finance team.

- Implement strong security policy enforcement relating to the application deployment
- Ensure containers run as a none-root user
- Restrict capabilities within containers to prevent unintended use of low-level functionality

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the infrastructure and can quickly identify flaws in it. The following phases and associated tools were used throughout the term of the audit:

- Familiarization with the application code to find more vulnerabilities & misconfigurations and judge them better
- Automated image scanning, enumeration and metadata extraction.
- Manually testing of application code and container implementation for security issues that could cause logical errors or data leakage on the platform.
- Automatic & manual checks of potential privileges misconfigurations

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The security assessment was scoped on all the contracts present under the src folder on commit https://github.com/UnlockdFinance/ltv-bot/tree/main/src

Following the initial audit the repository was refactored and moved to: https://github.com/UnlockdFinance/backend

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	3	1	2	0

LIKELIHOOD

IMPACT

🟨	🟨	🟥 (HAL-01) (HAL-02)	🟥	🟪
🟩	🟨 (HAL-04)	🟨	🟥 (HAL-03)	🟥
🟩	🟩 (HAL-05)	🟨	🟨	🟥
⬜	🟩 (HAL-06)	🟩	🟨	🟨
⬜	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CONTAINER RUNNING AS ROOT	High	SOLVED - 11/17/2022
(HAL-02) NO POD SECURITY CONSTRAINT ENFORCEMENT	High	SOLVED - 12/28/2022
(HAL-03) SINGLE K8S SECRET CONTAINS MANY CREDENTIALS	High	SOLVED - 12/28/2022
(HAL-04) UNRESTRICTED LINUX CAPABILITIES	Medium	SOLVED - 12/28/2022
(HAL-05) WEAK PASSWORDS IN USE	Low	SOLVED - 12/28/2022
(HAL-06) FIREWALL ALLOWS UNRESTRICTED ACCESS TO SENSITIVE PORTS	Low	SOLVED - 12/28/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) CONTAINER RUNNING AS ROOT - HIGH

Description:

A review of the configuration files used to deploy the application revealed that the application would be running as the root user. As the container runs within a shared kernel, should a successful container escape occur, an attacker would gain full root privileges on the underlying node. This in turn would compromise all containers running on a given node or host.

Recommendation:

Ensure the deployment configuration enforces the use of a user and group above, 10000 to ensure no process collisions occur. This may require additional modification to process within containers to ensure the required permissions are retained.

Remediation plan:

SOLVED: The security context was implemented to prevent containers from running as root, instead using uid and guid 1001:
- f726580ef67728a793865cab283d67695a5341dd.

3.2 (HAL-02) NO POD SECURITY CONSTRAINT ENFORCEMENT - HIGH

Description:

While access to the Kubernetes cluster was out of the scope of this assessment, a static review of the deployment revealed a number of missing security controls within the deployment files accompanying the application.

- Below is a none-exhaustive list of the missing controls:
  - runAsUser/Group set to < 10000
  - Host namespaces
  - privileged
  - allowPrivilegeEscalation
  - readOnlyRootFilesystem
  - seLinux
  - seccomp (can be enforced as a security context)
  - linux capabilities (these should be restricted to only those that are required for the service to function)

Code Location:

Listing 1: src/k8s/deployment-rinkeby.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ltv-bot-rinkeby
  namespace: bot
  labels:
    app: ltv-bot-rinkeby
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ltv-bot-rinkeby
  template:
```

```
  metadata:
    labels:
      app: ltv-bot-rinkeby
  spec:
    containers:
      - name: ltv-bot-rinkeby
        image: gcr.io/unlockd-357515/ltv-bot:v0.0.1
        imagePullPolicy: Always
        env:
          - name: UPSHOT_KEY
            value: UPSHOT_KEY
          - name: NFTBANK_KEY
            value: NFTBANK_KEY
    restartPolicy: Always
```

Recommendation:

A policy enforcement mechanism should be implemented that allows the following controls to be enforced against all pods as a baseline. This can be further supplemented through the use of a third-party active monitoring solution within the clusters:

- runAsUser/Group set to > 10000
- Host namespaces
- privileged
- allowPrivilegeEscalation
- readOnlyRootFilesystem
- seLinux
- seccomp (can be enforced as a security context)
- linux capabilities (these should be restricted to only those that are required for the service to function)
- (more information on these policies can be found here, here and here)

  Note: PSP's have been depreciated in the latest versions of Kubernetes in favor of custom admission controllers (OPA/Gatekeeper, K-Rail, Kyverno).

Remediation plan:

SOLVED: Policy enforcement through Kyverno was implemented via baseline and restricted policies. These policies were found to implement strong defense-in-depth controls across all resources.
- 51ffd5a6637bbb6cd6ef55bfedb30f8e101e7173

3.3 (HAL-03) SINGLE K8S SECRET CONTAINS MANY CREDENTIALS - HIGH

Description:

A review of the recent changes to the Kubernetes deployment file revealed the use of a single Secret object used to house secrets for multiple deployments. Kubernetes controls the distribution of secrets by restricting Secret objects to only nodes with actively deployed pods associated with the Secret object. Due to the use of a single Secret object, this meant that sensitive passwords and private keys were exposed to all deployed pods across all nodes, potentially exposing database credentials and private keys.

Additionally, it should be noted that Kubernetes secrets are stored unencrypted within the YAML configuration files as such sensitive data such as cryptographic keys should not be stored within Secret objects.

Affected Resources:

Listing 2: k8s/goerli/secrets.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: ltv-bot-secret
type: Opaque
data:
  postgres-user: [REDACTED]
  postgres-password: [REDACTED]
  infura-key: [REDACTED]
  upshot-key: [REDACTED]
  nftbank-key: [REDACTED]
  owner-private-key: [REDACTED]
  ltv-manager-private-key: [REDACTED]
  liquidator-private-key: [REDACTED]
```

Recommendation:

It is recommended that secrets are split into multiple Secret objects to ensure the isolation and adherence to the principle of least privilege is applied.

Additionally, the use of a Key Management Solution is recommended to ensure secrets are appropriately encrypted. More information about encrypting secrets within Kubernetes can be found here:
- https://kubernetes.io/docs/tasks/administer-cluster/kms-provider/

Remediation plan:

SOLVED: Secrets were removed from the deployments and replaced with strongly encrypted stored secrets.
- 51ffd5a6637bbb6cd6ef55bfedb30f8e101e7173

3.4 (HAL-04) UNRESTRICTED LINUX CAPABILITIES - MEDIUM

Description:

The application container was found to have all available Linux capabilities active. This meant that it was possible to perform privileged actions such as writing raw TCP packets to the underlying host's TCP/IP stack using the NET_CAP_RAW capability. In a properly segregated network, this misconfiguration could allow low-level enumeration of resources on the network.

Recommendation:

Capabilities used by containers should be restricted following the concept of least privilege. This should ensure all potentially dangerous capabilities are disabled by default.

More information regarding modifying Linux capabilities within container can be found here: https://linuxera.org/capabilities-seccomp-kubernetes/

Remediation plan:

SOLVED: Policy enforcement through Kyverno has been implemented via baseline and restricted policies. These policies were found to implement strong defense-in-depth controls across all resources.- 51ffd5a6637bbb6cd6ef55bfedb30f8e101e7173

3.5 (HAL-05) WEAK PASSWORDS IN USE - LOW

Description:

A review of the secrets within the Kubernetes cluster revealed the use of weak credentials used to connect to the postgreSQL database. While this deployment is intended to be deployed in a testnet environment, credentials should conform to a strict password policy enforcing robust length and complexity requirements.

This ensures that in the event credentials are mistakenly unmodified between the test and production deployment phases, strong security controls are in place to prevent malicious access to services.

Recommendation:

Passwords should be updated to conform to a strong password policy within the constraints applied by the software in use. Ideally, a password length of at least 16 characters with upper and lower case characters alongside numeric and special characters to improve the complexity.

Remediation plan:

SOLVED: Weak secrets were removed from the deployment and replaced with strongly encrypted stored secrets.
- 51ffd5a6637bbb6cd6ef55bfedb30f8e101e7173

3.6 (HAL-06) FIREWALL ALLOWS UNRESTRICTED ACCESS TO SENSITIVE PORTS - LOW

Description:

A review of the current GCP firewall rules revealed a number of default rules allowing access to sensitive ports from any source address were active. The accessible ports included: ICMP, SSH and RDP. While this configuration does not represent a direct risk, these ports are commonly scanned and attacked by threat actors in the wild.

Affected Resources:

- default-allow-icmp
- default-allow-ssh
- default-allow-rdp

Recommendation:

These firewall rules should be removed in favor of source restricted rules for any ports that are required.

Remediation plan:

SOLVED: The permissive firewall rules have been disabled. Complete removal of these rules is recommended to ensure accidental or malicious reactivation of these rules does not occur.

THANK YOU FOR CHOOSING

HALBORN