Worldcoin Orb AppSec Code Review

Security Assessment

February 20, 2024

Prepared for:
Daniel Girshovich
Tools for Humanity

Prepared by: **Dominik Czarnota, Artur Cygan, and Jack Leightcap**

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 100+ 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.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

This report is considered by Trail of Bits to be public information; it is licensed to Tools for Humanity under the terms of the project statement of work and has been made public at Tools for Humanity's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic.

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 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

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Anders Helsing, Engineering Director, Application Security
anders.helsing@trailofbits.com

The following engineers were associated with this project:

Dominik Czarnota, Consultant
dominik.czarnota@trailofbits.com

Artur Cygan, Consultant
artur.cygan@trailofbits.com

Jack Leightcap, Consultant
jack.leightcap@trailofbits.com

Project Timeline

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

Date	Event
August 14, 2023	Pre-project kickoff call
August 14, 2023	Status update meeting #1
August 21, 2023	Status update meeting #2
August 28, 2023	Delivery of report draft; report readout meeting
September 15, 2023	Delivery of updated report
October 10, 2023	Delivery of updated report incorporating client feedback
December 15, 2023	Delivery of draft fix review addendum
February 20, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

On behalf of the Worldcoin Foundation (the "Foundation"), Tools for Humanity (TFH) engaged Trail of Bits to review the software running on the Worldcoin Orb device and its privacy and security claims. The Orb, a Foundation asset, contains a set of cameras that are used to ensure the uniqueness and realness of participants who sign up in the Worldcoin network to obtain a World ID. The Orb's function is to verify unique humanness. To do this, it scans a QR code provided by the user on a phone, analyzes the user's iris, generates a unique iris code, and sends "iris data" along with the user's identity commitment to the Orb's back end.

A team of three consultants conducted the review from August 7 to August 26, 2023, for a total of six engineer-weeks of effort. Our testing efforts focused on validating a list of security and privacy claims and assessing the security of the Orb software and its operating system. With full access to source code and documentation and shell access to two devices (development and production), we performed static and dynamic testing of the provided codebase, using automated and manual processes. The review concerned only the software running on the Orb device and developed by TFH and did not include any of the back-end code.

Observations and Impact

Although findings TOB-ORB-4 and TOB-ORB-5, TOB-ORB-10, and TOB-ORB-11 describe potential attack surfaces and are of high or medium severity, our analysis did not uncover vulnerabilities in the Orb's code that can be directly exploited in relation to the Project Goals as described. We provide recommendations for hardening the kernel configuration in appendix C.

Recommendations

Based on the findings identified during the security review, Trail of Bits recommends that TFH take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Fuzz the native libraries that take untrusted input.**
- **Use taint tracking to ensure that user eye images or iris codes are never used in unexpected ways.** While we did not find issues with the incorrect use of personally identifiable information (PII) or iris codes, a future code change could introduce such issues. To prevent this, we recommend researching and using taint

tracking solutions to ensure that this data is never used incorrectly (for example, that it is never passed to logging functions). While we have not found a robust solution for implementing taint tracking in Rust, projects such as `LiHRaM/taint`, `facebookexperimental/MIRAI`, and `willcrichton/flowistry` (or their example specifically) may be useful.

- **Further harden the systemd services and Linux kernel used by the Orb.** For example, many of the TFH services running on the device (those listed in appendix D) could be further hardened if the "no new privileges" flag were enabled. The kernel could be further hardened by implementing the recommendations provided in TOB-ORB-8.

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	1
Medium	3
Low	1
Informational	6
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Audit and Logging	1
Configuration	5
Cryptography	1
Data Exposure	2
Data Validation	2
Patching	1

Project Goals

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

- Can we break any of the Worldcoin Orb privacy and security claims?
- Is PII handled securely? Is it prevented from persisting on the device?
- Are all inputs validated properly?
- Can a malicious user-provided QR code or other image exploit the device?
- Are the iris images encrypted such that the device cannot decrypt them when they are sent to the back end during an opt-in signup flow?
- Is the communication with the back end encrypted end-to-end with mutual authentication?
- Is the data sent to the back end signed to ensure that the back end is communicating with a legitimate device?
- Is the over-the-air update mechanism secure?
- Is any sensitive information leaked to logs or to the Datadog metrics service?
- Is the device configuration secure and hardened?

Privacy and Security Claims

As part of the engagement, we sought to validate a list of technical privacy and security claims from TFH. This section lists those claims and includes our analysis of them.

Non-Goals

The following items were out of scope for this analysis, as agreed with TFH:

- “Proving” any of the claims definitively
- Attestation that specific Orb devices in the field adhere to any such claims
- Claims about the privacy and security of past or future versions of the Orb software
- Quantifying potential harms associated with malicious Foundation/TFH employees
- Hardware-related safety claims (e.g., eye safety)

Claim 1: For the default opt-out signup flow, no PII except the iris code is sent by the Orb

We sought to validate the following:

- No PII is written to persistent storage on the Orb.
- No PII except the iris code leaves the Orb (e.g., is sent over the network).

Analysis

The Orb scans the QR code provided by the user and analyzes their iris for uniqueness and humanness. The QR code encodes information such as the user's ID and data policy (whether they opt in to or opt out of encrypting and sending their eye image data to the back end). The eye iris images are used to compute an iris code, which is then used to know whether a given set of eyes is already registered. The signup flow sends the user ID and the eyes pipeline data (which includes iris processing software versions, iris codes, and, for opt-in users, eye images) along with the operator's ID, the Orb device ID, the software version, and a signature. The signature is computed based on a SHA-256 hash of the Orb device ID, the user ID, and the eyes pipeline data using the `orb-sign-iris-code` binary. This binary (which was not within the scope of this audit) uses the secure element to compute the signature. If the user uses the default opt-out signup flow, the Orb does not send the encrypted eye images to the back end or persist them to disk.

We did not find that any PII data is written to persistent storage or that other user data leaves the Orb in any way in the audited version of the software. However, there is a risk that PII data could persist to disk in the future if swap space is ever enabled on the Orb device (it is currently disabled) (TOB-ORB-1) or if a core dump of the Orb processes is ever generated (TOB-ORB-6). While those are not currently security issues, PII data persistence due to future configuration changes can be prevented by following the recommendations provided for those findings.

Claim 2: For the non-default opt-in signup flow, PII is handled securely by the Orb

We sought to validate the following:

- The only PII persisted on the device is on the Orb's SSD and is encrypted.
- Encrypted PII stored on the Orb's SSD cannot be decrypted by the Orb.

Analysis

The encryption functionality is well defined in the `image_saver` module, in which the frames are encrypted and stored on the Orb's SSD. We did not identify any places where PII is persisted outside of the Orb's SSD and/or unencrypted.

The encryption is performed with the sealed box cryptographic construction from libsodium, a widely used and well-regarded library in the community. The public key of the recipient is hard-coded in the Orb's code under the `WORLDCOIN_ENCRYPTION_PUBKEY` constant, and the corresponding private key is not present in the production build. The libsodium library internally generates an ephemeral key pair that is destroyed right after the encryption process. To summarize, once PII is encrypted, the described mechanism does not permit the Orb to decrypt it.

Claim 3: The Orb does not extract any sensitive data from a user's device

We sought to validate the following:

- The only information the Orb receives from a user's phone is in the QR code.

Analysis

We analyzed the services running on the device, its exposed ports, and the Orb software, and we did not identify any process or functionality of the Orb software and operating system that would break this claim; note, however, that we did not perform an exhaustive and adversarial analysis (that is, we assumed good intentions, and we did not look for malware or rootkits on the devices that we had access to).

We did find an issue with the QR scanner in use: the ZBar library's decoder is configured so that it will scan not only QR codes but also other barcode types (TOB-ORB-5).

Claim 4: The user's iris code is handled securely

We sought to validate the following:

- The user's iris code is not written to persistent storage on the Orb.
- The user's iris code is included only in a single request to the Orb's back end.
- The user's iris code cannot be extracted from the Orb's network traffic.

Analysis

During our code review, we found that the iris code is used in the following places:

- It is logged if the `orb-core` codebase is built with the `log-iris-data` feature, but this feature is not enabled on production builds (which are built through the Nix package manager and the `orb-core/flake.nix` description).
- It is used to compute the signature sent to the back end. Specifically, the iris code is hashed along with other data with the SHA-256 algorithm, and the result is Base64-encoded and then passed to the `orb-sign-iris-code` program to compute the final signature sent to the back end.
- It is sent in a single request to the Orb's back end.

From our analysis, we believe the iris code is not written to persistent storage on the Orb and that it is included only in a single request to the Orb's back end.

For its network communication, the `orb-core` codebase enforces HTTPS only and trusts only two root certificates (from Amazon and Google Trust Services). While this configuration can be improved to make it more secure (TOB-ORB-10), it should not be possible for typical attackers to extract the iris code from the Orb's network traffic; the attacker would have to be in control of one of the trusted certificates.

Project Targets

The engagement involved a review and testing of the Imaginative Imp (3.0.10) release of the Orb software, including its custom user-space applications written in Rust and custom Linux distribution based on Debian.

TFH provided us with the `worldcoin-tob-privacy-audit` repository (commit `2ccc3c01f7bc7103831d49c25382b7cca9ce3ee2`), containing the codebases listed below.

We also received SSH access to two Orb test devices with development and production configurations. The production configuration was specifically unfused to allow for SSH access, as SSH is disabled on production devices according to TFH.

orb-core

Type	Rust
Purpose	The main process of the orb; processes camera streams (scanning of QR codes and users' irises); interfaces with neural networks; manages back-end communication for signups

orb-update-agent

Type	Rust
Purpose	Handles over-the-air updates

orb-supervisor

Type	Rust
Purpose	Manages resources and timing between the other processes

ai-models

Type	Python
Purpose	Neural network libraries utilized by `orb-core`

plug-and-trust

Type	C
Purpose	Builds the tool for signing iris codes (`orb-secure-element`); out of scope for the audit; listed here for completeness

orb-mcu-util

Type	Rust
Purpose	Allows basic access and logging of the MCUs; out of scope for the audit; listed here for completeness

Additionally, the following targets or areas were identified as explicitly out of scope for this engagement:

- The bootloader configuration
- Driver modifications
- TrustZone applets
- The secure element interface
- Hardware
- Any possible attacks that could be carried out by malicious employees (as agreed with TFH on the engagement's kickoff call, and as stated under the non-goals of the Privacy and Security Claims section)

Project Coverage

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

- Review of the user signup flow and the way the obtained data (QR codes and iris images) is handled
- Analysis of the device environment, its running services and configuration, and its permissions and kernel configuration
- Review of the `orb-supervisor` and `orb-update-agent` services
- Fuzzing of ZBar, the QR code scanning library in use
- Investigation of taint tracking tools for Rust, which would help with ensuring that input data is not handled in an unexpected way (e.g., is not exposed in logs or persisted to disk)

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:

- The `ai-models` neural network Python libraries
- Native dependencies used by the Orb, such as the Thermal Camera SDK, Aruco, Alsa, gstreamer, and video4linux

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time	Semgrep Pro rules for Rust
libFuzzer	An in-process, coverage-guided, evolutionary fuzzing engine that can automatically generate a set of inputs that exercise as many code paths in the program as possible	Appendix B

Areas of Focus

Our automated testing and verification work focused on the following system properties:

- The program does not access invalid memory addresses.
- The program does not exercise undefined behavior.
- The program does not use unsafe functions or constructs.

Fuzzing Harnesses

The following are the fuzzing harnesses we developed that exercise a subset of the program's code.

Property	Tool	Result
ZBar's decoder: A fuzzer against ZBar's decoder	libFuzzer	TOB-ORB-4, TOB-ORB-5, Appendix B
ZBar's QR decoder: A harness that tests only ZBar's QR code decoder	libFuzzer	No crashes; Appendix B

Summary of Findings

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

ID	Title	Type	Severity	Fix Status
1	User data may persist to disk if the swap space is ever configured	Data Exposure	Informational	Partially Resolved
2	Risk of wrong SSD health check space reported due to integer overflow	Audit and Logging	Low	Resolved
3	An expired token for a nonexistent API checked into source code	Data Exposure	Informational	Resolved
4	Memory safety issues in the ZBar library	Data Validation	High	Resolved
5	The Orb QR code scanner is configured to detect all code types	Configuration	Medium	Resolved
6	Core dumps are not disabled	Configuration	Informational	Resolved
7	World writable and readable sockets	Configuration	Undetermined	Unresolved
8	Opportunities to harden the static kernel configuration and runtime parameters	Configuration	Informational	Partially Resolved
9	The downloaded list of components to update is not verified	Cryptography	Informational	Resolved
10	Security issues in the HTTP client configuration	Configuration	Medium	Partially Resolved
11	External GitHub CI/CD action versions are not pinned	Patching	Medium	Resolved

12	The deserialize_message function can panic	Data Validation	**Informational**	**Unresolved**

Detailed Findings

1. User data may persist to disk if the swap space is ever configured

Fix Status: **Partially Resolved**	
Severity: **Informational**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-ORB-1
Target: `orb-core`	

Description

The Orb software does not lock memory to RAM. This means that if the swap space is ever configured in the future, PII data such as user iris image data may be swapped to disk and persist there indefinitely.

The severity of this finding is rated as informational because the device does not currently have the swap space configured (figure 1.1).

```
root@localhost:/home/worldcoin# free -h
               total        used        free      shared  buff/cache   available
Mem:           6.7Gi       456Mi       5.7Gi        58Mi       604Mi       6.0Gi
Swap:             0B          0B          0B

root@localhost:/home/worldcoin# swapon --summary
root@localhost:/home/worldcoin#
```

Figure 1.1: The test device does not have swap space configured.

Recommendations

Short term, use the `mlock` syscall to lock the memory where PII data is stored to RAM. This will prevent that memory from being paged to the swap area if the swap area is ever configured on the device.

Long term, when the device is updated to Linux kernel version 5.14 or newer, consider using the `memfd_secret` syscall to further secure important memory areas. This syscall allows allocated memory to be unmapped in the kernel space, which may provide more security guarantees.

2. Risk of wrong SSD health check space reported due to integer overflow

Fix Status: **Resolved**	
Severity: **Low**	Difficulty: **Low**
Type: Audit and Logging	Finding ID: TOB-ORB-2
Target: `orb-core/src/brokers/observer.rs`	

Description

The `handle_ssd_health_check` function (figure 2.1)—used to fill in the SSD health check status request, which is sent to the TFH back end—could report incorrect values due integer overflow. The function copies the values from the `Stats` structure (figure 2.2) to the `Ssd` structure fields; however, these fields are defined as `i32` (figure 2.3), so the `Stats` values may overflow.

As a result, the reported SSD disk space values will always be in the `i32` range, which is from around -2 GB to 2 GB, or exactly in the range of -2,147,483,648 to 2,147,483,648 bytes, even if the read disk space values exceed this range.

```
#[allow(clippy::cast_possible_truncation)]
fn handle_ssd_health_check(&mut self, report: &ssd::Stats) {
    self.status_request.ssd.space_left = report.available_space as _;
    self.status_request.ssd.file_left = report.documents as _;
    self.status_request.ssd.signup_left_to_upload = report.signups as _;
}
```

Figure 2.1: `orb-core/src/brokers/observer.rs`

```
/// SSD statistics.
pub struct Stats {
    /// Available space on the SSD.
    pub available_space: u64,
    /// Number of signups left to upload.
    pub signups: isize,
    /// Number of files left to upload.
    pub documents: isize,
    /// Number of files left to upload.
    pub documents_size: u64,
}
```

Figure 2.2: `orb-core/src/ssd.rs`

```
pub struct Ssd {
    pub file_left: i32,
    pub space_left: i32,
    pub signup_left_to_upload: i32,
}
```

Figure 2.3: `orb-core/src/backend/status.rs`

Exploit Scenario

The Orb device has 21,474,836,482 bytes of disk space left, which is around 21 GB. However, the values copied from the `Stats` structure overflow, causing `handle_ssd_health_check` to report just 1 byte of space left to the back end.

Recommendations

Short term, change the types used in the `Ssd` structure to `u64` so that its values can hold the full range of possible disk space values.

Long term, add unit tests for this case. Consider banning the `#[allow(clippy::cast_possible_truncation)]` annotation from the code to prevent similar issues from happening in the future.

3. An expired token for a nonexistent API checked into source code

Fix Status: **Resolved**

Severity: **Informational**	Difficulty: **Undetermined**
Type: Data Exposure	Finding ID: TOB-ORB-3
Target: `orb-core/src/consts.rs`	

Description

One of the defined `orb-core` constants is an unused and expired `DISTRIBUTOR_API_TOKEN` token (figures 3.1–3.3). The token was used to communicate with an API that no longer exists. Nonetheless, such secret values should never be kept in plaintext in source code repositories, as they can become valuable tools to attackers if the repository is compromised or if an employee who should not have access to production tokens becomes malicious.

```
/// Distributor API URL.
pub const DISTRIBUTOR_API_URL: &str = "https://api.getworldcoin.com/v1/graphql";

/// Distributor API Token.
pub const DISTRIBUTOR_API_TOKEN: &str =
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IlpVSUZ3Y0FXMGxBVEozSmxtQjRzWSJ9.eyJodH
RwczovL2hhc3VyYS5pby9qd3QvY2xhaW1zIjp7IngtaGFzdXJhLWFsbG93ZWQtcm9sZXMiOlsib3JiIl0sIn
gtaGFzdXJhLWF1dGgwLWlkIjoiYXV0aDB8NjA4MDYxZmUwZjAxN2EwMDY5YTZiZjIwIiwieC1oYXN1cmEtZG
VmYXVsdC1yb2xlIjoib3JiIn0sIm5pY2tuYW1lIjoib3JiIiwibmFtZSI6Im9yYkB3b3JsZGNvaW4ub3JnIi
wicGljdHVyZSI6Imh0dHBzOi8vcy5ncmF2YXRhci5jb20vYXZhdGFyLzliZTQwYzVlMzNkZWQyOGNjOGE1YT
I5NTgwYWQ1M2FjP3M9NDgwJnI9cGcmZD1odHRwcyUzQSUyRiUyRmNkbi5hdXRoMC5jb20lMkZhdmF0YXJzJT
JGb3IucG5nIiwidXBkYXRlZF9hdCI6IjIwMjEtMDQtMjFUMTc6NDA6NDAuNTAzWiIsImVtYWlsIjoib3JiQH
dvcmxkY29pbi5vcmciLCJlbWFpbF92ZXJpZmllZCI6ZmFsc2UsImlzcyI6Imh0dHBzOi8vYXV0aC53b3JsZG
NvaW4tZGlzdHJpYnV0b3JzLmNvbS8iLCJzdWIiOiJhdXRoMHw2MDgwNjFmZTBmMDE3YTAwNjlhNmJmMjAiLC
JhdWQiOiJPVHpiZU5jVlFERjJZb3UyNkR6U2JQbEVQOXNUcjBMMyIsImlhdCI6MTYxOTAyNjg0MSwiZXhwIj
oxNjUwMTMwODQxLCJhdF9oYXNoIjoic21TVzUtRVNUS0pPbXRwTEV3WWVuZyIsIm5vbmNlIjoiS0JERmx5X2
1QSlllQ3FybE1wREN0c0kyczBBajBReXcifQ.gfScnofoSxFo7cHcmYu6dINfDGZRUEAPbCkARtR8gEnb4bN
9kTFCKYhuGvUxXn4ffHjQUuu-qg4s5ABuqp9XddTYMZDYsEYw7lguiT68RSj9U18067ac6CP8Ltmg96g742K
o27Dt-_7isKDT1CUI55pbO4tW1QY1B23hTGR6CFMN2cdgEOrSFh7kHHeRzLT20p7dqQ-k0WWyDraGMK9jzXo
chzs5w10ziWYMvEWfg9gAXkUVR9ZxEvlHT2g1BKrcDH1Wyp-WdgdLg3d9AOYzNsaFi9kPg5u8QDUVsW5inaR
KpF0FD0v4Q1B6oNlTEpU49a7v-VNVL0Uu6346r7_5Vg";
```

Figure 3.1: `orb-core/src/consts.rs`

```
{"alg":"RS256","typ":"JWT","kid":"ZUIFwcAW0lATJ3JlmB4sY"}

{"https://hasura.io/jwt/claims":{"x-hasura-allowed-roles":["orb"],"x-hasura-auth0-id
":"auth0|608061fe0f017a0069a6bf20","x-hasura-default-role":"orb"},"nickname":"orb","
name":"orb@worldcoin.org","picture":"https://s.gravatar.com/avatar/9be40c5e33ded28cc
8a5a29580ad53ac?s=480&r=pg&d=https%3A%2F%2Fcdn.auth0.com%2Favatars%2For.png","update
d_at":"2021-04-21T17:40:40.503Z","email":"orb@worldcoin.org","email_verified":false,
"iss":"https://auth.worldcoin-distributors.com/","sub":"auth0|608061fe0f017a0069a6bf
20","aud":"OTzbeNcVQDF2You26DzSbPlEP9sTr0L3","iat":1619026841,"exp":1650130841,"at_h
ash":"smSW5-ESTKJOmtpLEwYeng","nonce":"KBDFly_mPJYeCqrlMpDCtsI2s0Aj0Qyw"}
```

Figure 3.2: The header and payload fields decoded from the JWT shown in figure 3.1

```
In [1]: from datetime import datetime

In [2]: print(datetime.utcfromtimestamp(1650130841).strftime('%Y-%m-%d %H:%M:%S'))
2022-04-16 17:40:41
```

Figure 3.3: The expiration time (the `"exp"` *claim highlighted in figure 3.2) converted to a datetime string, showing that the token has already expired*

Recommendations

Short term, remove the hard-coded `DISTRIBUTOR_API_TOKEN` token and the `DISTRIBUTOR_API_URL` constants from the `orb-core` codebase, as those values are not used at all and the related API does not exist anymore.

Long term, improve the CI/CD tooling used for the Orb to detect dead code such as this unused token.

References

- GitHub Docs: Removing sensitive data from a repository

4. Memory safety issues in the ZBar library

Fix Status: **Resolved**	
Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ORB-4
Target: `orb-core/qr-code/src/scanner.rs`, ZBar library code	

Description

The Orb uses the ZBar library, which has multiple memory safety issues. The library is used to scan user-provided QR codes with the identity commitment. We found two issues by fuzzing the `zbar_scan_image` function, which is directly used in the Orb's Rust code. (Refer to appendix B for more information on our fuzzing of the ZBar library.)

We identified the following issues related to memory:

1. The `match_segment_exp` function may read a stack buffer out of bounds. The function is used when the Orb reads GS1 DataBar codes.
2. The `_zbar_sq_decode` function has a memory leak issue. The function is used when the Orb reads Digital Seal SQCode codes, which are a variation of QR codes. The leak likely happens when the function encounters one of the error conditions and the allocated memory is not freed.
3. ZBar's QR code reader also seems to have a memory leak issue, as reported in `mchehab/zbar#258`.

We did not investigate the full impact of these issues or try to exploit them; nonetheless, the out-of-bounds stack buffer read could help attackers in exploiting the device and gaining the ability to execute code on it, and the two memory leaks can result in memory exhaustion, which would cause a denial of service.

Also, while the first two issues are not related to QR codes, they still affect the Orb since the Orb will try to read most code types if not configured otherwise, as highlighted in finding TOB-ORB-5. We confirmed this by constructing a fuzzing harness that uses the same configuration as the Orb. This means that the first two bugs we found would not be triggerable if the Orb were configured to scan only QR codes. However, the QR code scanner's memory leak issue could still affect the Orb device.

Exploit Scenario

An attacker shows the Orb a malicious barcode that uses a memory corruption vulnerability to give the attacker the ability to execute arbitrary code on the device. After taking control of the program, the attacker is able to exploit the Orb.

Recommendations

Short term, take the following actions:

- Disable scanning of barcode types other than QR codes, as recommended for fixing finding TOB-ORB-5.
- Work with the ZBar library maintainers to fix the issues described in this finding and to release a fixed version of the ZBar library.

Long term, extract out the functionality of QR code scanning to an external process and sandbox it so that it will have limited access if a vulnerability within it were exploited. This could be achieved with the help of the `sandboxed-api` tool.

5. The Orb QR code scanner is configured to detect all code types

Fix Status: **Resolved**

Severity: **Medium**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-ORB-5

Target: `orb-core/qr-code/src/scanner.rs`

Description

The Orb uses the ZBar library to scan user-provided QR codes. However, the ZBar library can scan different types of barcodes other than QR codes, and the Orb does not reconfigure the library to scan only QR codes. As a result, it may scan a barcode format that was not intended to be used with the Worldcoin Orb system. This increases the attack surface of the system and may lead to other issues, especially since the ZBar library has memory safety issues, as detailed in finding TOB-ORB-4.

Exploit Scenario

An attacker finds a way to exploit a memory safety issue in the ZBar library by scanning a non–QR code image. They use this vulnerability to exploit the Orb device.

Recommendations

Short term, disallow detection of all code types except QR codes in the ZBar library. Use the `zbar_image_scanner_set_config` function to explicitly enable QR codes and disable other barcode types. Example code to do so is shown in figure 5.1 (though there may be a better method). Also, note that while some code types are not enabled by default in the ZBar library, it is still worth explicitly disabling them so that an update of the library cannot enable them.

```
// Allow QR codes
zbar_image_scanner_set_config(scanner, ZBAR_QRCODE, ZBAR_CFG_ENABLE, 1);
// Disable all other types
zbar_image_scanner_set_config(scanner, ZBAR_EAN2, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_EAN5, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_EAN8, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_UPCE, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_ISBN10, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_UPCA, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_EAN13, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_ISBN13, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_COMPOSITE, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_I25, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_DATABAR, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_DATABAR_EXP, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_CODABAR, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_CODE39, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_PDF417, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_DATABAR, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_DATABAR_EXP, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_SQCODE, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_CODE93, ZBAR_CFG_ENABLE, 0);
zbar_image_scanner_set_config(scanner, ZBAR_CODE128, ZBAR_CFG_ENABLE, 0);
```

Figure 5.1: Example ZBar code to enable scanning of QR codes and to disable other barcode types

Long term, add a functional, integration, or device test to ensure that the Orb device does not scan barcode types other than QR codes, such as with the example images from the ZBar repository and malicious barcodes from MalQR. Also, the ZBar library should be inspected every time it is updated to ensure that the available barcode types have not changed; this will prevent new and unexpected barcode types from being accepted by the Orb. Document this process internally to ensure it is done.

6. Core dumps are not disabled

Fix Status: **Resolved**	
Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-ORB-6
Target: Kernel runtime parameters (`sysctl`)	

Description

The Orb device does not have core dumps disabled in its kernel runtime parameters (figure 6.1). If the generation of a core dump of one of the Worldcoin Orb processes is triggered, PII data could be persisted to the device disk.

```
root@localhost:~# sysctl -a | grep core_
kernel.core_pattern = core
kernel.core_pipe_limit = 0
kernel.core_uses_pid = 0
```

Figure 6.1: The production device core dump configuration

The severity of this finding is rated as informational because it is unlikely that a core dump can be generated with the current device configuration.

For a core dump to be generated, the target process would have to run in a writable path, but the overlay path (`/`) is mounted as read-only on the production device (though some paths like the `/tmp` path are still writable). The target process would also need to have a nonzero core file size resource limit set. Nonetheless, since the current working directory and a resource limit of a process could change, it is still worth it to disable the core dumps completely.

Recommendations

Short term, disable core dumps by setting the `kernel.core_pattern=/dev/null` `sysctl` parameter. This can be done by creating a `sysctl` configuration file in the `/etc/sysctl.d/99-worldcoin-disable-core-dumps.conf` path when provisioning the device.

References

- `core` Linux manual page
- Arch Linux: Core dumps

7. World writable and readable sockets

Fix Status: **Unresolved**	
Severity: **Undetermined**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-ORB-7
Target: systemd socket configuration	

Description

The socket files in the `/tmp` directory of the production device have permissions that are too broad. The `camsock`, `nvscsock`, and `worldcoin_bus_socket` sockets can be read and written to by any user on the system; additionally, the `wpa_ctrl_1216-1` socket can be read by any user (figure 7.1).

These permissions allow users or processes that should not have access to these sockets to access them; depending on the data that these sockets read/write, any user or process may be able to gain additional privileges or exploit the device.

```
root@localhost:/home/worldcoin# ls -la --color=auto /tmp
total 0
drwxrwxrwt 12 root      root      320 Aug 16 14:07 .
drwxrwxrwx  1 root      root      211 Jul  9 09:53 ..
drwxrwxrwt  2 root      root       40 Mar 27 17:54 .ICE-unix
drwxrwxrwt  2 root      root       40 Mar 27 17:54 .Test-unix
drwxrwxrwt  2 root      root       40 Mar 27 17:54 .X11-unix
drwxrwxrwt  2 root      root       40 Mar 27 17:54 .XIM-unix
drwxrwxrwt  2 root      root       40 Mar 27 17:54 .font-unix
srwxrwxrwx  1 root      root        0 Mar 27 17:54 camsock
srwxrwxrwx  1 root      root        0 Mar 27 17:54 nvscsock
drwx------  2 worldcoin worldcoin  40 Aug 11 14:22 pulse-PKdhtXMmr18n
drwx------  3 root      root       60 Mar 27 17:54
systemd-private-a86ceebec50c43a3b1b3c79e36389670-haveged.service-2XXwnh
drwx------  3 root      root       60 Mar 27 17:54
systemd-private-a86ceebec50c43a3b1b3c79e36389670-systemd-logind.service-etQOVh
drwx------  3 root      root       60 Aug 14 20:57
systemd-private-a86ceebec50c43a3b1b3c79e36389670-systemd-resolved.service-EYsKKg
drwx------  3 root      root       60 Aug 16 13:13
systemd-private-a86ceebec50c43a3b1b3c79e36389670-systemd-timesyncd.service-veio4h
srw-rw-rw-  1 worldcoin worldcoin   0 Mar 27 17:54 worldcoin_bus_socket
srwxr-xr-x  1 root      root        0 Mar 27 17:54 wpa_ctrl_1216-1
```

Figure 7.1: Permissions of sockets in the `/tmp` path that are too broad

Recommendations

Short term, change the permissions with which the sockets are created so they have the least required permissions in order for the Worldcoin Orb software to function properly. Do not set any of the socket files to be readable or writable by any user of the system.

Long term, add tests to ensure the production device never ends up with a world readable or writable socket file or any other important file.

8. Opportunities to harden the static kernel configuration and runtime parameters

Fix Status: **Partially Resolved**	
Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-ORB-8
Target: Kernel configuration	

Description

The Orb's kernel configuration and runtime parameters can be improved to increase the overall security of the device and its applications and to decrease the potential attack surface.

The following table shows the runtime parameters that can be hardened. These parameters can be read from the device by reading files under the `/proc/sys/` path or through the `sysctl` utility tool. (Some parameters require root privileges to be read.)

Parameter	Current Value	Recommendation
`kernel.kptr_restrict`	1	Set the parameter to 2 to prevent kernel pointers from being leaked, even to privileged processes.
`kernel.unprivileged_bpf_disabled`	`0`	Set the parameter to 1 to prevent unprivileged processes from using the `bpf` syscall. Refer also to the notes regarding this parameter below the table.
`net.core.bpf_jit_enable`	1	Set the parameter to `0` to disable the BPF JIT mechanism. The BPF JIT mechanism increases the attack surface of the kernel.
`net.core.bpf_jit_harden`	`0`	If the BPF JIT mechanism is still enabled, set the parameter to 2 to enable hardening for the BPF JIT compiler.
`bpf_jit_kallsyms`	1	If the BPF JIT mechanism is still enabled, set the parameter to `0` to disable the exporting of kernel symbols for the BPF JIT mechanism.

Regarding BPF, the `bpf` syscall can be used by unprivileged processes as long as the `kernel.unprivileged_bpf_disabled` `sysctl` option is disabled, which is its default value. While this syscall should not allow a user or process to escalate privileges or crash the system, it had bugs in the past that did allow such actions, such as CVE-2023-0160, CVE-2022-2785, CVE-2022-0500, CVE-2021-4204, CVE-2020-8835, and CVE-2017-16995. (Refer also to this blog post on the CVE-2017-16995 bug.)

The kernel configuration can also be improved. We scanned the configuration from the provided development device with the `kconfig-hardened-check` project (as the production device is already hardened and does not expose its kernel configuration). The results from this scan are provided in appendix C.

Recommendations

Short term, adjust the `sysctl` options as recommended in this finding. This will harden the device and lower its attack surface.

Long term, take the following actions:

- Use the kernel lockdown feature to prevent certain modifications to the kernel image and certain configuration options. This feature was added in version 5.4 of the Linux kernel.
- Rebuild the Linux kernel and harden its configuration options according to the `kconfig-hardened-check` project results provided in appendix C.

9. The downloaded list of components to update is not verified

Fix Status: **Resolved**	
Severity: **Informational**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-ORB-9
Target: `orb-update-agent/update-agent/src/main.rs`	

Description

When the `orb-update-agent` service downloads an update claim that contains the list of components to be updated, it does not verify the claim's `signature` field (figure 9.1). As a result, if a malicious Orb server attempts to hijack an Orb device by serving it a malicious update claim, the `orb-update-agent` service would not detect that the claim is malicious.

The severity of this finding is rated as informational because the code contains a commented-out signature verification check with a "TODO" comment indicating plans to enable it (figure 9.2).

Additionally, the Orb uses the dm-verity Linux kernel feature to ensure the runtime integrity of the filesystems. We have not checked the efficacy of this feature, but the use of it should prevent a malicious update from being used.

```
#[derive(Serialize, Debug)]
pub struct Claim {
    version: String,
    manifest: crate::Manifest,
    #[serde(rename = "manifest-sig")]
    signature: String,
    sources: HashMap<String, Source>,
    system_components: crate::Components,
}
```

Figure 9.1: The `signature` field in the `Claim` structure
(`orb-update-agent/update-agent-core/src/claim.rs`)

```
fn run() -> eyre::Result<()> {
    ...
    // TODO: Enable signature verification
    //    let mut file = std::fs::File::open(settings.pubkey).expect("failed to open
pubkey");
    //    let mut contents: Vec<u8> = Vec::new();
    //    file.read_to_end(&mut contents)?;
    //
    //    debug!("len: {:?}", contents.len());

    // let public_key = ring::signature::UnparsedPublicKey::new(
    //     &ring::signature::RSA_PKCS1_1024_8192_SHA256_FOR_LEGACY_USE_ONLY,
    //     contents,
    // );

    // claim.verify(public_key)?;
```

Figure 9.2: The commented-out signature verification check
(`orb-update-agent/update-agent/src/main.rs`)

Recommendation

Short term, add the update claim signature verification check.

10. Security issues in the HTTP client configuration

Fix Status: **Partially Resolved**	
Severity: **Medium**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-ORB-10
Target: `orb-update-agent/update-agent/src/main.rs`	

Description

Both the `orb-core` and `orb-update-agent` codebases use a similar HTTP client configuration, but each has its own code to initialize an HTTP client (figures 10.1 and 10.2). We identified the following issues with that code:

- The clients do not have a minimum TLS version set, which should be 1.3.
- The `orb-update-agent` service does not restrict the client to use only HTTPS requests.
- The clients do not have redirects disabled. As a result, a malicious server could perform a server-side request forgery (SSRF) attack on the client, making it hit some of its internal resources.

Additionally, the clients are currently set to trust the Amazon and Google Trust Services root certificates. While this is better than trusting all default root certificates, it still creates the risk of system compromise by a malicious third-party. The TFH team knows about this issue and has a long-term goal to use a private CA in the future.

```rust
fn initialize() -> Result<Client, Error> {
    let amazon_cert = AMAZON_ROOT_CA_1_CERT
        .get_or_try_init(|| Certificate::from_pem(AMAZON_ROOT_CA_1_PEM))
        .map_err(Error::CreateAmazonRootCa1Cert)?
        .clone();
    let google_cert = GTS_ROOT_R1_CERT
        .get_or_try_init(|| Certificate::from_pem(GTS_ROOT_R1_PEM))
        .map_err(Error::CreateGtsRootR1Cert)?
        .clone();
    Client::builder()
        .add_root_certificate(amazon_cert)
        .add_root_certificate(google_cert)
        .tls_built_in_root_certs(false)
        .user_agent(APP_USER_AGENT)
        .timeout(Duration::from_secs(120))
        .build()
        .map_err(Error::BuildClient)
}
```

Figure 10.1: `orb-update-agent/update-agent/src/client.rs`

```
/// Creates a new HTTP client.
///
/// # Panics
///
/// If [`init_cert`] hasn't been called yet.
pub fn client() -> reqwest::Result<reqwest::Client> {
    reqwest::Client::builder()
        .user_agent(APP_USER_AGENT)
        .timeout(REQUEST_TIMEOUT)
        .connect_timeout(CONNECT_TIMEOUT)
        .tls_built_in_root_certs(false)
        .add_root_certificate(
            AWS_CA_CERT.get().expect("the AWS root certificate is not
initialized").clone(),
        )
        .add_root_certificate(
            GTS_CA_CERT.get().expect("the GTS root certificate is not
initialized").clone(),
        )
        .https_only(true)
        .build()
}
```

Figure 10.2: `orb-core/src/backend/mod.rs`

Exploit Scenario 1

An attacker finds a way to make the `orb-update-agent` service connect to the back end via HTTP. Since the HTTP client does not enforce HTTPS, the attacker is able to perform a person-in-the-middle attack against the updater and further attack the device.

Exploit Scenario 2

A sophisticated malicious actor who has access to the AWS/GTS root certificate and is in a person-in-the-middle position against the Orb device creates an HTTPS certificate for the TFH back-end domain and attacks the device by spoofing the TFH back-end server with the created certificate.

Recommendations

Short term, take the following actions:

- Set a minimum TLS version. This can be done by calling the `min_tls_version` method in the request client builder object. Ideally, the minimum version should be set to 1.3. However, the currently used `native-tls` back end does not allow 1.3 to be set as the minimum TLS version. Because of that, we also recommend switching to a different back end that allows 1.3 to be set as the minimum version.

- Enforce the use of HTTPS-only connections in the `orb-update-agent` HTTP client. This can be done by calling the `https_only(true)` method in the request client builder object.
- Move the implementation of the two HTTP request clients from `orb-core` and `orb-update-agent` to a utility library. This will allow the same HTTP client configuration to be used by both codebases.
- Disable redirects in the clients by calling the `redirect(Policy::none())` method on the request client builder object. This will prevent SSRF attacks from malicious servers against the Orb device.

Long term, take the following actions:

- Create and pin a TFH root certificate.
- Authenticate the device on the server with client certificates.

These actions will ensure that the device connects to a legitimate TFH server, reduce risks stemming from trust in certificates from third parties, and prevent fake devices from connecting to the server.

References

- OWASP: Pinning Cheat Sheet
- `sfackler/rust-native-tls#140`: An issue related to the lack of TLS 1.3 support in the `native-tls` back end

11. External GitHub CI/CD action versions are not pinned

Fix Status: **Resolved**	
Severity: **Medium**	Difficulty: **Medium**
Type: Patching	Finding ID: TOB-ORB-11
Target: GitHub Actions workflows and actions	

Description

The GitHub Actions pipelines use several third-party actions. These actions are part of the supply chain for TFH's CI/CD pipelines and can execute arbitrary code in the pipelines. A security incident in any of the third-party GitHub accounts or organizations can lead to a compromise of the CI/CD pipelines and any artifacts they produce.

This issue does not impact the release builds of the Orb software since they are not built through the GitHub Actions CI/CD workflows.

The following actions are owned by GitHub organizations that might not be affiliated directly with the API/software they are managing:

- `EndBug/add-and-commit@v9`
- `actions/cache@v3`
- `actions/checkout@v1`
- `actions/checkout@v2`
- `actions/checkout@v3`
- `actions/download-artifact@v2`
- `actions/download-artifact@v3`
- `actions/setup-python@v2`
- `actions/setup-python@v3`
- `actions/setup-python@v4`
- `actions/upload-artifact@v2`
- `actions/upload-artifact@v3`
- `arduino/setup-protoc@v1`
- `aws-actions/configure-aws-credentials@v2`
- `cachix/cachix-action@v12`
- `cachix/install-nix-action@v22`
- `dawidd6/action-get-tag@v1` (archived)
- `docker/build-push-action@v3`
- `docker/setup-buildx-action@v2`
- `docker/setup-qemu-action@v2`

- `dtolnay/rust-toolchain@1.62.0`
- `dtolnay/rust-toolchain@1.64.0`
- `dtolnay/rust-toolchain@1.67.1`
- `dtolnay/rust-toolchain@nightly`
- `goto-bus-stop/setup-zig@v1`
- `goto-bus-stop/setup-zig@v2`
- `marocchino/sticky-pull-request-comment@v2`
- `softprops/action-gh-release@v1`
- `taiki-e/install-action@parse-changelog`

Note that we included GitHub actions from organizations like GitHub Actions and Docker even though TFH already implicitly trusts these organizations by virtue of using their software. However, if any of their repositories is hacked, that may impact TFH's CI builds.

Exploit Scenario

A private GitHub account with write permissions from one of the untrusted GitHub actions is taken over by social engineering (e.g., a user is using an already leaked password and is convinced to send a 2FA code to an attacker). The attacker updates the GitHub action to include code to exfiltrate the code and secrets in CI/CD pipelines that use that action, including the TFH CI/CD pipelines.

Recommendations

Short term, pin all external and third-party actions to a Git commit hash. Avoid pinning them to a Git tag, as tags can be changed after creation. We also recommend using the `pin-github-action` tool to manage pinned actions. GitHub Dependabot is capable of updating GitHub actions that use commit hashes.

Long term, audit all pinned actions or replace them with a custom implementation.

12. The deserialize_message function can panic

Fix Status: **Unresolved**	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ORB-12
Target: `orb-core/src/port.rs`	

Description

The `deserialize_message` function does not have a limit on the `size` value that is decoded from the received buffer, so the process can panic if the decoded size exceeds the size of the provided buffer (figure 12.1). This may lead to a program crash if the data comes from an untrusted source.

The severity of this finding is rated as informational because this function is used to exchange data between Orb processes and we did not find a way to trigger a panic from untrusted input.

```
unsafe fn deserialize_message<T>(buf: &[u8]) -> &T::Archived
where
    T: Archive + for<'a> Serialize<SharedSerializer<'a>>,
{
    let size =
usize::from_ne_bytes(buf[..mem::size_of::<usize>()].try_into().unwrap());
    let bytes = &buf[mem::size_of::<usize>()..mem::size_of::<usize>() + size];
    unsafe { rkyv::archived_root::<T>(bytes) }
}
```

Figure 12.1: `orb-core/src/port.rs`

Recommendations

Short term, change the `deserialize_message` function to return an error if it fails to deserialize a buffer.

Long term, add tests or fuzzing for the `deserialize_message` function to make sure it works as expected when receiving invalid inputs and does not crash the program.

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. ZBar Library Fuzzing

We discovered that the ZBar library used by the Orb to scan QR codes is not fuzz tested continuously. Given that the library serves as an entry point to the system, we decided to run our own fuzzing campaign against it to test its security.

We developed a few fuzzing harnesses using the libFuzzer fuzz testing framework and AddressSanitizer against the ZBar library version that is used in the Orb device (version 0.23.90). This helped us find the memory safety issues described in finding TOB-ORB-4.

The developed fuzzing harnesses were shared with the Worldcoin Orb team and with the ZBar library maintainers. We agreed to redact the rest of this appendix because the found bugs were not fixed during the assessment, and doing so prevents the easy discovery of other bugs. We also shared the fuzzing harnesses with the ZBar library maintainers and suggested that they integrate them into the OSS-Fuzz project to find issues continuously.

C. Kernel Configuration Hardening Recommendations

This appendix is an addition to the TOB-ORB-8 finding. We retrieved the kernel configuration from the provided development device by reading its `/proc/config.gz` file. We scanned the resulting configuration with the `kconfig-hardened-check` project. The results that failed the checks are provided in figure C.1.

Note that the production device runs a slightly more hardened configuration, so it does not expose its kernel configuration in the same way as the development device does. Nonetheless, we recommend inspecting the production device configuration and hardening it further.

```
dc@ns3203937:~$ kconfig-hardened-check -c kernel.config | grep FAIL
CONFIG_IOMMU_DEFAULT_DMA_STRICT         |kconfig|     y      |defconfig | self_protection  | FAIL: is not found
CONFIG_ARM64_EPAN                       |kconfig|     y      |defconfig | self_protection  | FAIL: is not found
CONFIG_RODATA_FULL_DEFAULT_ENABLED      |kconfig|     y      |defconfig | self_protection  | FAIL: "is not set"
CONFIG_ARM64_PTR_AUTH_KERNEL            |kconfig|     y      |defconfig | self_protection  | FAIL: is not found
CONFIG_ARM64_BTI_KERNEL                 |kconfig|     y      |defconfig | self_protection  | FAIL: is not found
CONFIG_MITIGATE_SPECTRE_BRANCH_HISTORY  |kconfig|     y      |defconfig | self_protection  | FAIL: is not found
CONFIG_KFENCE                           |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_ZERO_CALL_USED_REGS              |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_HW_RANDOM_TPM                    |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_STATIC_USERMODEHELPER            |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_RANDSTRUCT_FULL                  |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_RANDSTRUCT_PERFORMANCE           |kconfig| is not set |   kspp   | self_protection  | FAIL: CONFIG_RANDSTRUCT_FULL is 
not "y"
CONFIG_GCC_PLUGIN_LATENT_ENTROPY        |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_MODULE_SIG_FORCE                 |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_INIT_STACK_ALL_ZERO              |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_GCC_PLUGIN_STACKLEAK             |kconfig|     y      |   kspp   | self_protection  | FAIL: "is not set"
CONFIG_STACKLEAK_METRICS                |kconfig| is not set |   kspp   | self_protection  | FAIL: CONFIG_GCC_PLUGIN_STACKLEAK
is not "y"
CONFIG_STACKLEAK_RUNTIME_DISABLE        |kconfig| is not set |   kspp   | self_protection  | FAIL: CONFIG_GCC_PLUGIN_STACKLEAK
is not "y"
CONFIG_RANDOMIZE_KSTACK_OFFSET_DEFAULT  |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_CFI_CLANG                        |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_CFI_PERMISSIVE                   |kconfig| is not set |   kspp   | self_protection  | FAIL: CONFIG_CFI_CLANG is not "y"
CONFIG_WERROR                           |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_SHADOW_CALL_STACK                |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_KASAN_HW_TAGS                    |kconfig|     y      |   kspp   | self_protection  | FAIL: is not found
CONFIG_SECURITY_YAMA                    |kconfig|     y      |   kspp   | security_policy  | FAIL: "is not set"
CONFIG_SECURITY_LANDLOCK                |kconfig|     y      |   kspp   | security_policy  | FAIL: is not found
CONFIG_BPF_UNPRIV_DEFAULT_OFF           |kconfig|     y      |defconfig |cut_attack_surface| FAIL: "is not set"
CONFIG_COMPAT                           |kconfig| is not set |   kspp   |cut_attack_surface| FAIL: "y"
CONFIG_MODULES                          |kconfig| is not set |   kspp   |cut_attack_surface| FAIL: "y"
CONFIG_DEVMEM                           |kconfig| is not set |   kspp   |cut_attack_surface| FAIL: "y"
CONFIG_IO_STRICT_DEVMEM                 |kconfig|     y      |   kspp   |cut_attack_surface| FAIL: "is not set"
CONFIG_GENERIC_TRACER                   |kconfig| is not set |  grsec   |cut_attack_surface| FAIL: "y"
CONFIG_FUNCTION_TRACER                  |kconfig| is not set |  grsec   |cut_attack_surface| FAIL: "y"
CONFIG_STACK_TRACER                     |kconfig| is not set |  grsec   |cut_attack_surface| FAIL: "y"
CONFIG_PROC_VMCORE                      |kconfig| is not set |  grsec   |cut_attack_surface| FAIL: "y"
CONFIG_DEBUG_FS                         |kconfig| is not set |  grsec   |cut_attack_surface| FAIL: "y"
CONFIG_FAIL_FUTEX                       |kconfig| is not set |  grsec   |cut_attack_surface| OK: is not found
CONFIG_KCMP                             |kconfig| is not set |  grsec   |cut_attack_surface| FAIL: "y"
CONFIG_RSEQ                             |kconfig| is not set |  grsec   |cut_attack_surface| FAIL: "y"
CONFIG_FB                               |kconfig| is not set |maintainer|cut_attack_surface| FAIL: "y"
CONFIG_VT                               |kconfig| is not set |maintainer|cut_attack_surface| FAIL: "y"
CONFIG_STAGING                          |kconfig| is not set |  clipos  |cut_attack_surface| FAIL: "y"
CONFIG_KALLSYMS                         |kconfig| is not set |  clipos  |cut_attack_surface| FAIL: "y"
CONFIG_EFI_CUSTOM_SSDT_OVERLAYS         |kconfig| is not set |  clipos  |cut_attack_surface| FAIL: "y"
CONFIG_COREDUMP                         |kconfig| is not set |  clipos  |cut_attack_surface| FAIL: "y"
CONFIG_BPF_SYSCALL                      |kconfig| is not set | lockdown |cut_attack_surface| FAIL: "y"
CONFIG_FTRACE                           |kconfig| is not set |    my    |cut_attack_surface| FAIL: "y"
CONFIG_TRIM_UNUSED_KSYMS                |kconfig|     y      |    my    |cut_attack_surface| FAIL: "is not set"
CONFIG_ARCH_MMAP_RND_BITS               |kconfig|     33     |    my    | harden_userspace | FAIL: "18"
[+] Config check is finished: 'OK' - 138 / 'FAIL' - 48
```

Figure C.1: Kernel hardening suggestions from the `kconfig-hardened-check` project

D. systemd Security Analysis

We executed `systemd-analyze security $SERVICE` to analyze the following production services:

- `worldcoin-coarse-location-cacher.service`
- `worldcoin-security-mcu-logger.service`
- `worldcoin-supervisor.service`
- `worldcoin-core.service`
- `worldcoin-short-lived-token-daemon.service`
- `worldcoin-update-agent.service`
- `worldcoin-dbus.service`
- `Worldcoin-short-lived-token-dumb-client.service`

The following lints failed for all services. We recommend conducting a manual review of each of them:

- Service has access to the host's network
- Service has administrator privileges
- Service may allocate Internet sockets
- Service may override UNIX file/IPC permission checks
- Files created by service are world-readable by default
- Service has full access to home directories
- Service has access to other software's temporary files
- Service may modify the control group file system
- Service has full access to the OS file hierarchy

The following lints failed for some services but are considered of secondary importance or are intended to fail:

- Service runs under a static non-root user identity
- Service may change UID/GID identities/capabilities
- Service has `ptrace()` debugging abilities
- Service may create user namespaces
- Service may allocate exotic sockets
- Service may change file ownership/access mode/capabilities unrestricted
- Service has network configuration privileges
- Service has raw I/O access
- Service may load kernel modules
- Service processes may change the system clock
- Service has no device ACL
- Service does not define an IP address whitelist
- Service processes may acquire new privileges

- Service potentially has access to hardware devices
- Service may install system mounts
- Service has access to other users
- Service may write to the hardware clock or system clock
- Service may read from or write to the kernel log ring buffer
- Service may load or read kernel modules
- Service may alter kernel tunables
- Service may allocate packet sockets
- Service may create SUID/SGID files
- Service may execute system calls with all ABIs
- Service does not filter system calls
- Service process receives ambient capabilities
- Service has audit subsystem access
- Service may send UNIX signals to arbitrary processes
- Service may create device nodes
- Service has elevated networking privileges
- Service has access to kernel logging
- Service has privileges to change resource use parameters
- Service may create cgroup namespaces
- Service may create IPC namespaces
- Service may create network namespaces
- Service may create file system namespaces
- Service may create process namespaces
- Service may acquire realtime scheduling
- Service may allocate netlink sockets
- Service runs within the host's root directory
- Service may adjust SMACK MAC
- Service may issue `reboot()`
- Service may change ABI personality
- Service may create writable executable memory mappings
- Service user may leave SysV IPC objects around
- Service may create hostname namespaces
- Service may mark files immutable
- Service may issue `chroot()`
- Service may change system host/domain name
- Service may establish wake locks
- Service may create file leases
- Service may use `acct()`
- Service may issue `vhangup()`
- Service may program timers that wake up the system
- Service may allocate local sockets

The following lints passed for all services:

- Service may lock memory into RAM
- Service does not maintain its own delegated control group subtree
- Service doesn’t share key material with other services
- Service has no supplementary groups
- Service child processes cannot alter service state

E. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From December 4 to December 6, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the TFH team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 12 issues described in this report, TFH has resolved five issues, has partially resolved five issues, and has not resolved the remaining two issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Difficulty	Status
1	User data may persist to disk if the swap space is ever configured	Informational	High	Partially Resolved
2	Risk of wrong SSD health check space reported due to integer overflow	Low	Low	Resolved
3	An expired token for a nonexistent API checked into source code	Informational	Undetermined	Resolved
4	Memory safety issues in the ZBar library	High	High	Resolved
5	The Orb QR code scanner is configured to detect all code types	Medium	High	Resolved
6	Core dumps are not disabled	Informational	High	Resolved
7	World writable and readable sockets	Undetermined	High	Unresolved

8	Opportunities to harden the static kernel configuration and runtime parameters	Informational	High	Partially Resolved
9	The downloaded list of components to update is not verified	Informational	High	Resolved
10	Security issues in the HTTP client configuration	Medium	High	Partially Resolved
11	External GitHub CI/CD action versions are not pinned	Medium	Medium	Resolved
12	The deserialize_message function can panic	Informational	High	Unresolved

Detailed Fix Review Results

TOB-ORB-1: User data may persist to disk if the swap space is ever configured
Partially resolved. The memory pages holding PII are still at risk of being persisted if the swap space is ever used. The device does not use the swap space, so the risk does not exist in the current configuration. We still recommend using the `mlock` system call on memory pages containing PII as a hardening/second-line-of-defense measure.

TOB-ORB-2: Risk of wrong SSD health check space reported due to integer overflow
Resolved in `orb-core/#791`. The types used in the `Ssd` structure were changed to `i64` to account for the possibility of an error. Even though we recommended using the `u64` type, the `i64` type can represent all of the real-life scenarios.

TOB-ORB-3: An expired token for a nonexistent API checked into source code
Resolved in `orb-core/#843`. The API token was removed from the code.

TOB-ORB-4: Memory safety issues in the ZBar library
Resolved in `orb-core/#942`. A new barcode scanning library, `rxing`, was added. It is written in Rust, which significantly reduces the chance of encountering memory safety issues. At the time of the fix review, the original ZBar scanning code was still in place behind a configuration flag.

The client provided the following context for this finding's fix status:

> *The configuration option to re-enable zbar was only used for testing, and the production software will ship with rxing as the only QR scanning library. zbar will not even be installed on production devices.*

TOB-ORB-5: The Orb QR code scanner is configured to detect all code types
Resolved in `orb-core/#942`. A new barcode scanning library, `rxing`, was added. It is configured to detect QR codes only. At the time of the fix review, the original ZBar scanning code was still in place behind a configuration flag.

The client provided the following context for this finding's fix status:

> *The configuration option to re-enable zbar was only used for testing, and the production software will ship with rxing as the only QR scanning library. zbar will not even be installed on production devices.*

TOB-ORB-6: Core dumps are not disabled
Resolved in `orb-os/#2`. The Linux kernel build configuration was updated with the `CONFIG_COREDUMP=n` option, which disables core dumps.

TOB-ORB-7: World writable and readable sockets
Unresolved. The client provided the following context for this finding's fix status:

> *Marked as no-fix.*

TOB-ORB-8: Opportunities to harden the static kernel configuration and runtime parameters
Partially resolved in `orb-os/#2`. The `bpf` syscall was disabled, eliminating BPF-related attack vectors. However, the `kernel.kptr_restrict` option was not configured per our recommendation. The PR also enabled KASLR; however, this change was rolled back due to a UEFI crash. We recommend enabling KASLR once doing so is possible, as it is a strong kernel exploit mitigation protection.

TOB-ORB-9: The downloaded list of components to update is not verified
Resolved in `orb-update-agent/#232` and `orb-os/#151`. The code that verifies the signature was implemented.

TOB-ORB-10: Security issues in the HTTP client configuration
Partially resolved in `orb-update-agent/fade320`, `orb-update-agent/ac33088`, `orb-update-agent/a49705d`, and `orb-core/#973`. The TFH team implemented all of our recommendations for the HTTP client configuration in `orb-update-agent`. However, the HTTP client configuration in `orb-core` is still missing the option that enforces the minimum TLS version.

TOB-ORB-11: External GitHub CI/CD action versions are not pinned
Resolved in `orb-core/#918`, `orb-supervisor/#33`, `TrustZone/#30`, `orb-update-agent/#239`, `plug-and-trust/#69`, `orb-mcu-firmware/#457`, and `orb-control-api-client/#135`. The external actions were pinned to their respective hashes.

TOB-ORB-12: The deserialize_message function can panic
Unresolved. The client provided the following context for this finding's fix status:

> *Marked as no-fix.*

F. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	Description
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.