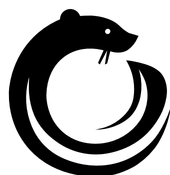
Wormhole Aptos

Audit

Presented by:



OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Harrison Green hgarrereyn@osec.io

Aleksandre Khokhiashvili khokho@osec.io

Contents

01 Executive Summary **2**
Overview . . . 2
Key Findings . . . 2

02 Scope **3**

03 Findings **4**

04 Vulnerabilities **5**
OS-WHA-ADV-00 [low] [resolved] | Ensure Contract Upgrade Integrity . . . 6
OS-WHA-ADV-01 [low] | Wormhole Deployment DOS . . . 8

05 General Findings **9**
OS-WHA-SUG-00 | GuardianSetChanged Event is Unused . . . 10
OS-WHA-SUG-01 | Explicit Global Initialization of Emitter Registry . . . 11
OS-WHA-SUG-02 | Document and Clarify Migration Code . . . 12
OS-WHA-SUG-03 | Prevent Partial Deployment of Wrapped Assets . . . 13

Appendices

A Vulnerability Rating Scale **14**

© 2022 OtterSec LLC. All Rights Reserved.

01 | Executive Summary

Overview

Wormhole Foundation engaged OtterSec to perform an assessment of the Wormhole Bridge for Aptos. This assessment was conducted between September 19th and October 7th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team over to streamline patches and confirm remediation. We delivered final confirmation of the patches October 8th, 2022.

Key Findings

Over the course of this audit engagement, we produced 6 findings total.

In particular, we identified two potential vulnerabilities around the contract upgrade/deployment process. Specifically, we discovered a mechanism for an attacker to prevent contract deployment by preemptively registering a coin account (OS-WHA-ADV-01) and a vulnerability affecting the integrity of the contract upgrade process, enabling an attacker to potentially upgrade the contract to malformed or unintended code (OS-WHA-ADV-00).

We also identified potential weak spots and areas of high-security importance, providing detailed recommendations to mitigate future vulnerabilities. For example, we provided recommendations to clearly outline the security requirements and assumptions about migration functionality during upgrades (OS-WHA-SUG-02) to ensure that protocol developers use safe procedures when writing and deploying updates.

Overall, the Wormhole team was responsive, attentive, and a pleasure to work with.

© 2022 OtterSec LLC. All Rights Reserved.

02 | Scope

The source code was delivered to us in a git repository at github.com/wormhole-foundation/wormhole/. This audit was performed against commit `5837c83`.

A brief description of the programs is as follows.

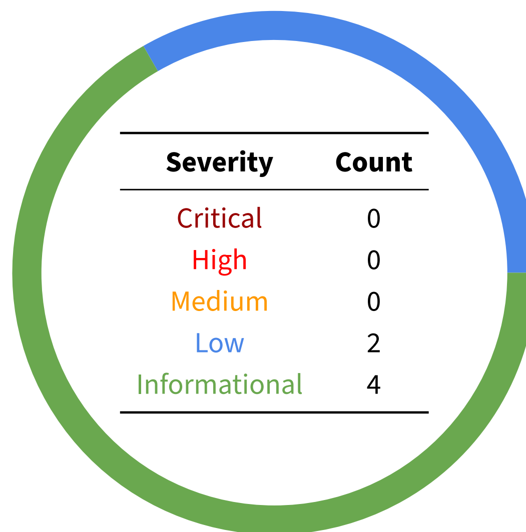
Name	Description
wormhole	Generic cross-chain message endpoint
token_bridge	Token bridge building on top of `wormhole`
coin	Wormhole wrapped coin module
deployer	Deployment utilities for `wormhole` and `token_bridge`

© 2022 OtterSec LLC. All Rights Reserved.

03 | Findings

Overall, we report 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	2
Informational	4

© 2022 OtterSec LLC. All Rights Reserved.

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-WHA-ADV-00	Low	Resolved	Contract upgrade should use stronger hash structures to verify integrity of code and serialized metadata.
OS-WHA-ADV-01	Low	Resolved	Wormhole deployment can be prevented via resource account DOS

© 2022 OtterSec LLC. All Rights Reserved.

OS-WHA-ADV-00 [low] [resolved] | Ensure Contract Upgrade Integrity

Description

In both wormhole and token-bridge, contract upgrade via governance is a two step process:

1. A user invokes contract_upgrade::submit_vaa with a governance VAA that includes a hash of the intended upgrade code.
2. A user invokes contract_upgrade::upgrade with the actual module code which subsequently invokes code::publish_package_txn to upgrade the contract.

Both of these calls are permissionless (can be invoked by any user). Integrity is protected in the first call by verifying the guardian signatures on the VAA. Integrity is protected in the second call by ensuring the provided module code matches the hash.

However, in the original implementation, we identified two issues that would allow an attacker to provide alternative module code that still matches the stored hash:

*/sources/contract_upgrade.move RUST

```
public entry fun upgrade(
    metadata_serialized: vector<u8>,
    code: vector<vector<u8>>
) acquires UpgradeAuthorized {
    ...
    let c = copy code;
    vector::reverse(&mut c);
    let a = vector::empty<u8>();
    while (!vector::is_empty(&c)) vector::append(&mut a,
     ↪  vector::pop_back(&mut c));
    assert!(keccak256(a) == hash, E_UNEXPECTED_HASH);
    ...
}
```

Serialized Metadata

A module upgrade package contains a list of code modules (vector<vector<u8>>) and serialized metadata (vector<u8>). In the original implementation, metadata_serialized was not included in the hash. Therefore, an attacker could provide any arbitrary metadata that would not cause the deployment to abort.

© 2022 OtterSec LLC. All Rights Reserved.

Module Boundaries

The original hash was computed by first concatenating the code modules and then taking the hash of the concatenated structure. However, this implementation does not properly validate module code boundaries. Specifically, moving N bytes of code from the end of one module to the start of another would not affect the hash:

```text
H([a,b,c,d], [e,f]) == H([a,b], [c,d,e,f])
```

It is unclear whether an attacker could exploit this collision to deploy a malformed version of the module. However, since the fix is simple, it is preferable to reduce the attack surface as much as possible.

Remediation

Include the serialized metatdata in the hash structure and compute a "hash of hashes" to enforce boundaries between module code:

(|| represents concatenation.)

```text
H(H(metadata) || H(code1) || H(code2) || ... || H(codeN))
```

Patch

This issue was fixed in dee93c4.

© 2022 OtterSec LLC. All Rights Reserved.

OS-WHA-ADV-01 [low] | Wormhole Deployment DOS

Description

During initialization of the wormhole module, it attempts to register an `AptosCoin` account in order to be able to receive fees:

wormhole/sources/wormhole.move RUST

```
module wormhole::wormhole {
    ...
    fun init_internal(...) {
        ...
        coin::register<AptosCoin>(&wormhole);
    }
    ...
}
```

However, `coin::register` is a one-time operation. If `coin::register` has previously been called on this address, this initialization code will abort and the wormhole program will be unable to initialize.

While it is usually not possible to register coins for users you can not sign for, the Aptos framework provides a special mechanism to register `AptosCoin` for *any* user via `aptos_account::create_account`:

aptos$_a$ccount.move RUST

```
public entry fun create_account(auth_key: address) {
    let signer = account::create_account(auth_key);
    coin::register<AptosCoin>(&signer);
}
```

Therefore, with this mechanism an attacker could register `AptosCoin` for the wormhole program before deployment in order to prevent it from properly initializing.

Remediation

Check if `AptosCoin` has been registered and conditionally invoke `coin::register` only if it hasn't been registered.

© 2022 OtterSec LLC. All Rights Reserved.

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-WHA-SUG-00	The GuardianSetChanged event is defined but not emitted.
OS-WHA-SUG-01	Explicitly enforce that emitter registry creation is a one-time operation.
OS-WHA-SUG-02	Explicitly document the security properties of migration code.
OS-WHA-SUG-03	Prevent partial deployment of wrapped assets in token-bridge.

© 2022 OtterSec LLC. All Rights Reserved.

OS-WHA-SUG-00 | GuardianSetChanged Event is Unused

Description

The `GuardianSetChanged` event is defined in the wormhole state module:

wormhole/sources/state.move RUST

```
struct GuardianSetChanged has store, drop {
    oldGuardianIndex: U32,
    newGuardianIndex: U32,
}
```

It would make sense to emit this event during `state::update_guardian_set_index` however this is currently unimplemented.

Remediation

Emit the event during `state::update_guardian_set_index`.

© 2022 OtterSec LLC. All Rights Reserved.

OS-WHA-SUG-01 | Explicit Global Initialization of Emitter Registry

Description

Messages in wormhole are sent via *emitters* which are tracked by a global *emitter registry*. In order to ensure message integrity, emitter IDs should not be reused and emitter sequence numbers should not be reset.

Currently, the emitter registry is stored in `wormhole::state` and created by a call to `emitter::init_emitter_registry()` during wormhole initialization. The current implementation currently performs this global initialization exactly once which is correct.

In order to ensure this initialization does not occur more than once, consider explicitly enforcing this property. For example, one approach is to construct a `HasCreatedEmitterRegistry` struct which is moved to the `wormhole` account during the initial creation and can not be dropped or removed.

© 2022 OtterSec LLC. All Rights Reserved.

OS-WHA-SUG-02 | Document and Clarify Migration Code

Description

On module upgrade, it will occasionally be required to perform some addition *migration* functionality which initializes new structures, copies data, etc...

In the current implementation, migration logic is included in `contract_upgrade::migrate` which can be optionally called after a contract upgrade. Specifically, on upgrade, the `Migrating` struct is stored on the account which indicates that migration can be performed. At this point, anyone can invoke `contract_upgrade::migrate` to perform the migration and remove the `Migrating` struct marker.

While this implementation properly guards against migration happening more than once, it does not enforce the following properties (that might be expected):

1. Migration code is not required to be run (it is optional, but permissionless).
2. Migration code does not run atomically with the module upgrade.

These properties are potentially unexpected and may lead to situations where developers assume migration code will execute atomically. In order to safely implement migration code, protocol developers should understand that users may interact with the protocol in between module upgrade and module migration.

Patch

This has been fixed in fa01fc1.

Specifically, the documentation has been expanded to clarify these points above. Additionally a new `is_migrating` function has been added in order to implement features that should be gated until after the migration has been run.

© 2022 OtterSec LLC. All Rights Reserved.

OS-WHA-SUG-03 | Prevent Partial Deployment of Wrapped Assets

Description

In order to handle non-native tokens (i.e. tokens that belong to other chains), wormhole's token-bridge deploys a "wrapped coin" which acts as a proxy for the remote coin.

Wrapped asset deployment happens in several steps:

1. A remote chain performs an *attestation* in order to emit an `AssetMeta` packet containing metadata about the asset.
2. A user relays the signed VAA in Aptos by invoking `token_bridge::wrapped::create_wrapped_coin_type` with the VAA. This function call creates a new resource account which will own the coin and deploys a template coin module on this account.
3. A user relays the signed VAA and calls `token_bridge::wrapped::create_wrapped_coin<T>` to invoke `coin::initialize` on the newly deployed module and store the associated information in the token bridge state.

The reason that steps 2 and 3 happen separately is due to the module deployment in step 2. Specifically, it is not possible to invoke a function on a module in the same transaction where you deploy it.

In the current implementation there is an inconsistency with the way these two functions handle native tokens. Since wrapped assets represent *foreign* assets, it does not make sense (and should not be allowed) to construct wrapped assets for native tokens (e.g. `AptosCoin`).

This assertion is only enforced by proxy in the second deployment step (`token_bridge::wrapped::create_wrapped_coin<T>`) which means that when trying to deploy a wrapped coin for a native token, the resource account and module will successfully be deployed.

There are no security implications due to this inconsistency but protocol uniformity could be improved by preventing deployment of wrapped native tokens at an earlier point.

© 2022 OtterSec LLC. All Rights Reserved.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation

© 2022 OtterSec LLC. All Rights Reserved.