Hack Analysis: Binance Bridge, October 2022

Immunefi · Follow
Published in Immunefi
7 min read · Mar 17

Listen Share

Introduction

On October 7th, 2022, the Binance Bridge was hacked due to a flaw in the IAVL Merkle proof verification system, which allowed a malicious hacker to steal 2m BNB, or about $600m USD equivalent at the time.

In response to the exploit, Binance paused the system for hours and introduced a blacklist mechanism.

In this article, we will be going through the process of how the malicious payload and proof were generated, which allowed the hacker to steal 2m BNB. We will also take it to the next level by crafting a payload that could have been used to drain the token Hub contract.

This article was written by Omik, an Immunefi smart contract triager.

Background

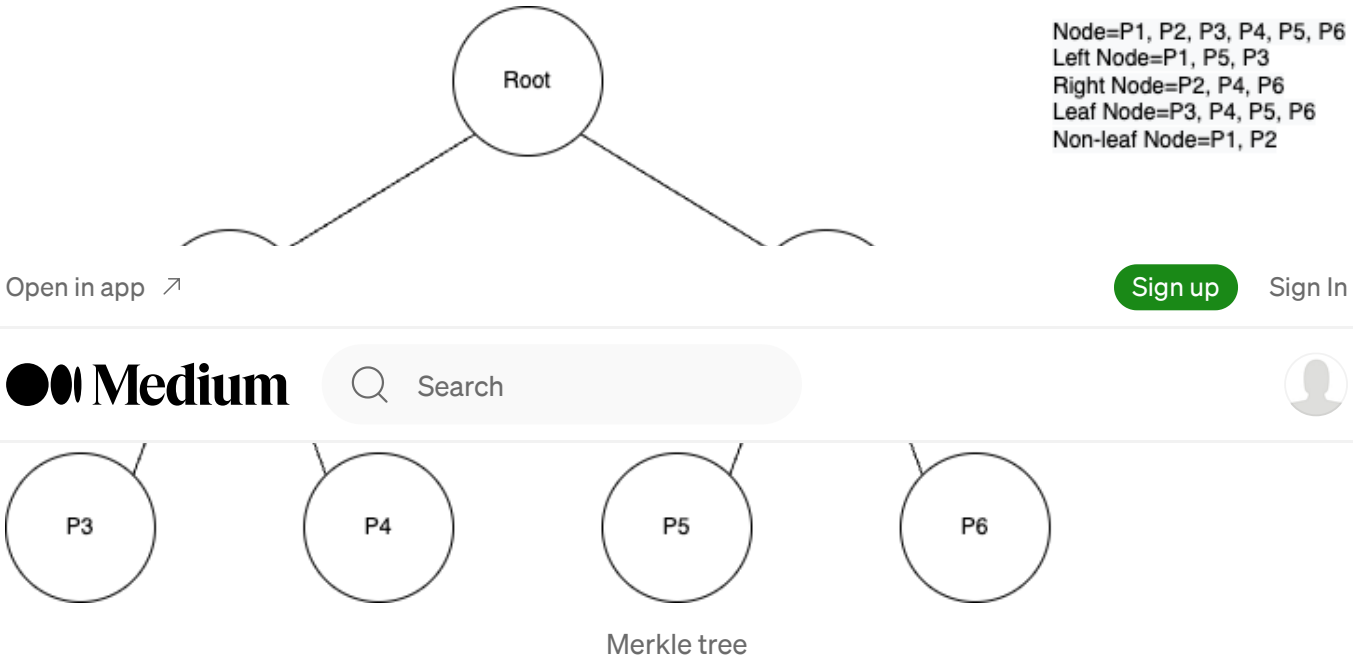
Binance is a centralized exchange (CEX) that offers a wide range of services, including cryptocurrency trading, staking, and portfolio management. It has two chains under its name: BNB Beacon Chain (BEP), which is used for governance, and BNB Chain (BNB), an L1 blockchain that is compatible with EVM and allows anyone to deploy and execute smart contracts on its chain.

BSC Token Hub is the bridge between BNB Beacon Chain (BEP2) and BNB Chain (BNB). It interacts with the Cross Chain contract, which allows a relayer to forward the user's token to BSC from BEP2. It verifies the token transfer through an IAVL verification, which can be validated from payload, proofs, package sequence, height, and channel id. And if the verification is deemed successful, the token would be transferred to the user in BSC.

Root Cause

Before discussing the root cause of the hack, let's briefly go through the Merkle tree structure and how the proofs can be generated to verify the data. We'll also go through the proofs of IAVL verification and how they differ from the usual proofs that we see in most smart contracts.

Merkle tree



Merkle tree

With a Merkle tree structure, we can verify if data is included in a database without disclosing all of the data in that database by hashing its value and comparing it against the root hash of the Merkle tree.

The data in each node in the Merkle tree is a hash of each child concatenated value. If it's a leaf node, the data is the hash value of the underlying data. So, the data in P1 is the hash of P3, and P4 (i.e P1=h(P3, P4)), and the data in P3 is the hash value of the data (i.e P3=h(data)).

To be able to verify the data against the Merkle tree, we must first know what's the path node of the data that we want to verify. The path node is nodes that lie on the path from the leaf node to the root node of the Merkle tree. The path node essentially is a guidance node that will bring our data to reach its root node.

For example, suppose we have some data in P3, and we want to verify whether this data is included in the Merkle tree or not. First, we must hash our data to generate the P3 value. Then, we must determine the path node of P3. In this case, the path node for P3 is P4 and P2. So by providing only the P4 and P2 values, we can reach the root node. The value for P4 and P2 can be considered as the Merkle proofs of the data we have in P3. And these proofs can be used to verify the data in P3.

You might ask: why is P1 not included in the path node? This is because we can generate the P1 value ourselves by concatenating the value of P3 and P4, which is equal to the value of P1.

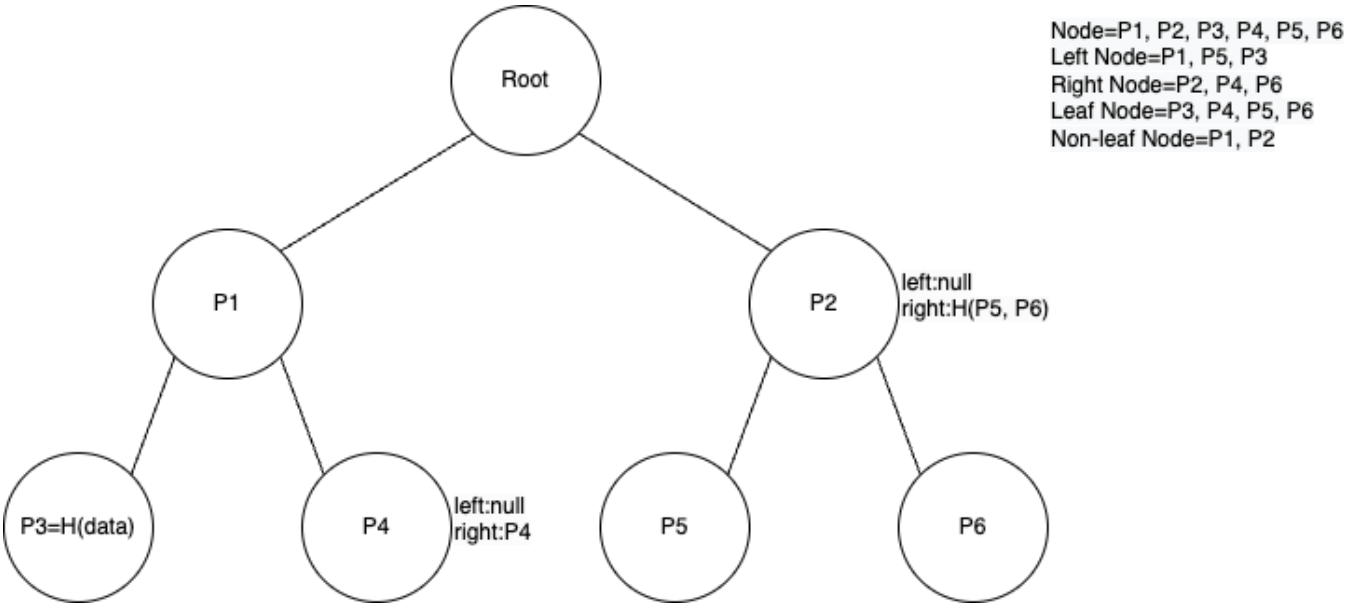
Naturally, one character difference in hashing will produce a completely different output. Therefore, the order of the concatenation of the node to generate the P1 or any non-leaf node is crucial. And there are multiple ways to do this ordering. In this case, we will cover the common ordering in the Merkle tree and the ordering that was used by Binance, which unfortunately led to the exploit.

Common ordering

The common concatenation ordering for Merkle verification in the Merkle tree is done by comparing the values of its child node. If the value of the child node is bigger than the value of the other child, then it's being concatenated to the child node that has a bigger value.

For example, node P1 has a P3 and P4 node as its child node. To get the value of P1, we must concat the P3 and P4 values and then hash the concatenated value. If the P3 value is bigger than the P4, then the ordering is (P3, P4), and if the P4 value is bigger than the P3, then the ordering is (P4, P3).

Ordering in the IAVL verification



IAVL Merkle tree

IAVL verification does its concatenation ordering by adding additional attributes to its path nodes. These attributes are right and left attributes. This was done to determine whether it should be hashed to the left or right side of the node.

For example, we can look at node P4 which has left attributes set to null and right attributes set to P4. This means that we can get the value P1 by hashing its child value with the P4 value as the right side of the data, or P1=H(P3, P4).

The root cause of this exploit is that the right attribute of the node was not used to calculate the root hash of the tree. Since the proof was user-controllable, the hacker was able to pass the malicious right attributes, which is the hash of the malicious payload, and the proof would be considered valid.

```
    if !bytes.Equal(derivedRoot, lpath.Right) {
        return nil, treeEnd, false, cmn.ErrorWrap(ErrInvalidRoot, "intermediate
    }
```

Proof of Concept

Once we understand the root cause of this issue, how can we set the right attributes in the proofs, and how did the attacker come up with that proof?

Emiliano Bonassi explained in one of his tweets that the attacker was using the proof from the very first cross-chain transaction. The proof was taken from this transaction.

Let's try to deconstruct the proof to make it more readable (this was inspired by Emiliano's script)

1. We take the legit proof from the above transaction.

2. It’s hard for us to read and modify the proof in its original form. That is why we need to unmarshal the proof. This will allow us to read and modify the data while maintaining its object type.

3. Decode the proof.

4. Print out the `rangeProof` of the legit proof.

5. Hash the malicious payload.

6. Craft the package sequence.

7. Add a new leave to the proof.

8. Add empty inner node to the proof.

9. Hash the new malicious leave to the right attribute of the proof.

10. Print out the malicious proof.

11. Marshal the malicious proof.

12. Verify the malicious proof to confirm that the malicious proof was valid.

The Attack

The relevant function `handlePackage()` can only be called by a relayer. To become a relayer, you must deposit 100 BNB to the relayer contract, which is exactly what the hacker did before they exploited the contract.

Since the precompile contract has already been updated, we can't test the PoC by forking locally. So, we will try to test the PoC locally without forking the mainnet. To verify if our attack works, we can use the verify function in the IAVL library.

We can begin the attack by crafting the payload and then passing the payload to the Go code, in order to generate the proof for the malicious payload.

The payload can be crafted by calling `craftPayload()` and passing the token address, amount, recipient, and the expiration time of the transaction. To take the native token (BNB), pass the token address as address(0).

Once we have the payload, we can set the payload to the `hackerPayloadOri` var in the `main.go` file, and run the `main.go`.

If the output of the `main.go` gives the same root hash for both the legit proof and malicious proof and returns no error when verifying the proof, it implies that our PoC is valid.

```
LEGIT ROOT HASH = e09159530585455058cf1785f411ea44230f39334e6e0f6a3c54dbf069df2
EVIL ROOT HASH = e09159530585455058cf1785f411ea44230f39334e6e0f6a3c54dbf069df2b
```

```
    error computing root hash? <nil>
    error verifying proof? <nil>
```

Conclusion

The Binance Bridge hack was one of the biggest hacks in 2022 and highlights the importance of security when integrating with other codebases. In this case, we learned that both sides of the attributes must be taken into account when calculating the root hash of the proof.

Binance fixed this issue by completely reverting the transaction if both sides of the attribute were set. That's why we couldn't run the POC by forking the mainnet locally.

Full source code for all payload components is available below.

`main.go`

`craftPayload.sol`

Binance Bridge | Binance | Bnb | Bug Bounty | Hacks

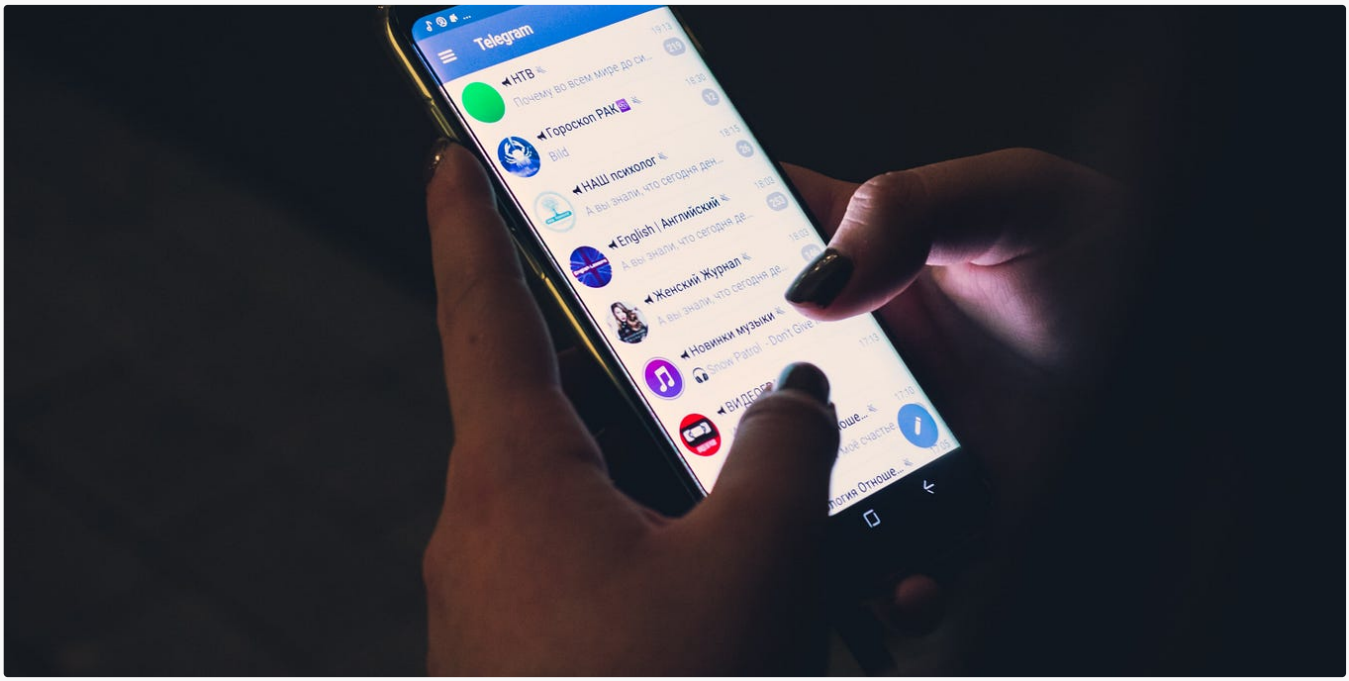
Follow

Written by Immunefi

3.7K Followers · Editor for Immunefi

Immunefi is the premier bug bounty platform for smart contracts, where hackers review code, disclose vulnerabilities, get paid, and make crypto safer.

More from Immunefi and Immunefi



Immunefi in Immunefi

How Not to Get Hacked on Telegram

The lightweight chat client Telegram is one of the most common methods of communication in crypto, and there's a good reason for that. SIM...

5 min read · Jul 28, 2021

919 5

Immunefi in Immunefi

Balancer Rounding Error Bugfix Review

Summary

8 min read · Oct 13

35

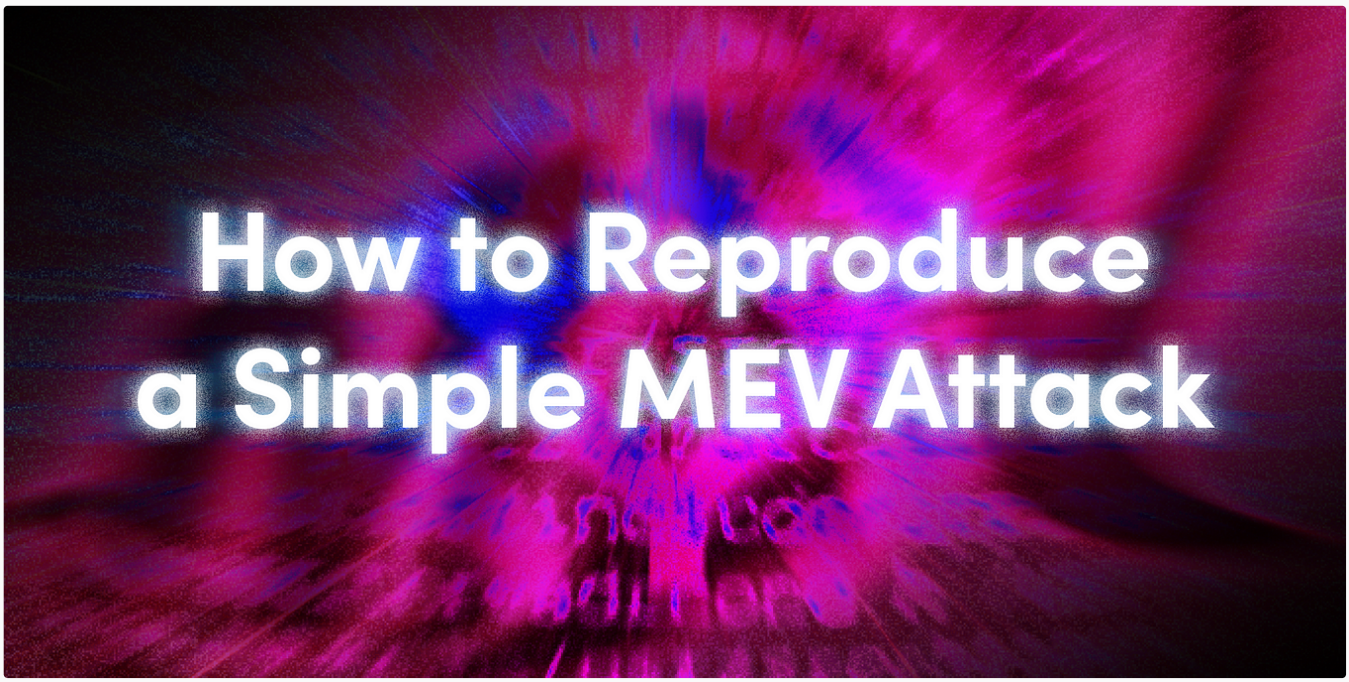
Immunefi in Immunefi

Sui Temporary Total Network Shutdown Bugfix Review

Summary

5 min read · Oct 14

9



Immunefi in Immunefi

How To Reproduce A Simple MEV Attack

Introduction

6 min read · Jul 21

63 2

See all from Immunefi

See all from Immunefi

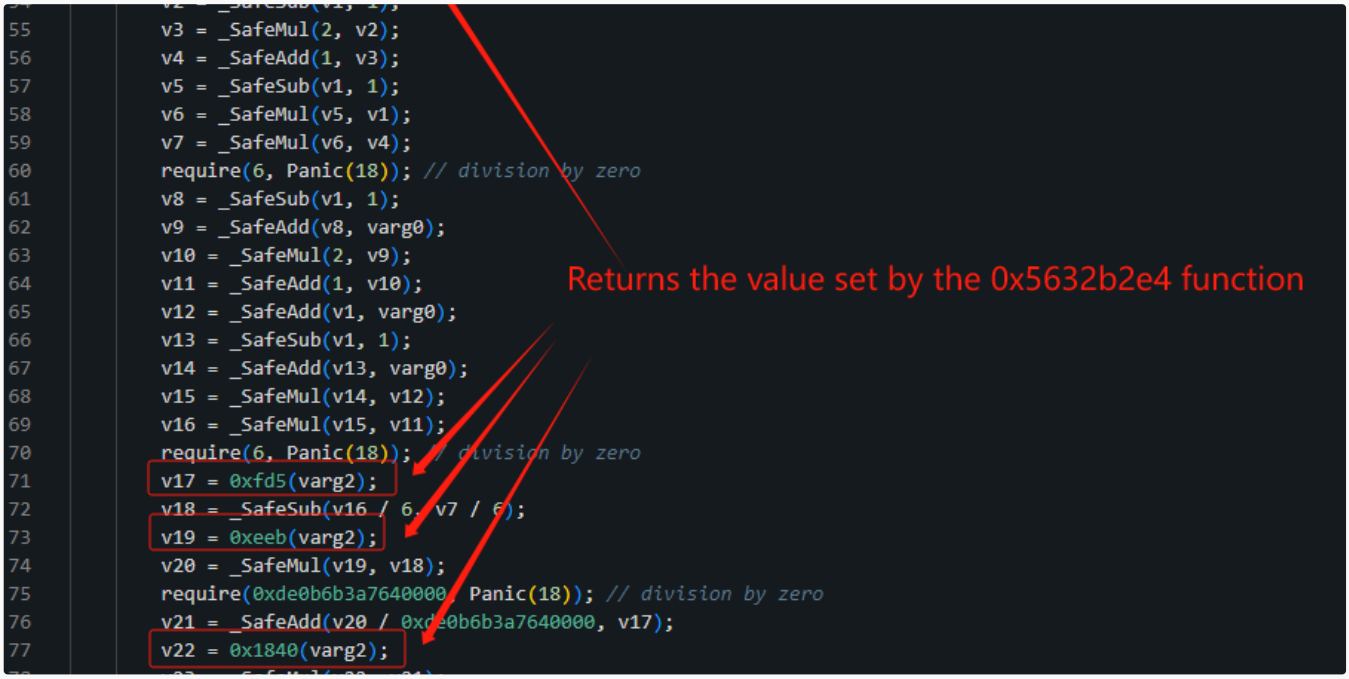
Recommended from Medium

Immunefi in Immunefi

Silo Finance Logic Error Bugfix Review

Summary

6 min read · Jun 16



Shashank in SolidityScan

StarsArena Hack Analysis

Overview:

2 min read · Oct 13

18

Lists

Medium Publications Accepting Story Submissions

154 stories · 884 saves

```
on executeFlashLoan(uint256 amount) external onlyOwne
dress asset = address(pool.asset());
ol.flashLoan(
  this,
  asset,
  amount,
  bytes("")
```

Kundan Kumar Kushwaha in CoinsBench

#1. Unstoppable—Damn Vulnerable DeFi

To pass the challenge make the Vault stop offering flash loans.

6 min read · Jun 18

4 1

Midas Author in Midas Capital

Midas Exploit Post-Mortem

At 5:35 pm UTC on June 17h, 2023, the isolated pool on Midas Capital aimed at supporting the partners from Ankr and Helio Finance on BNB...

5 min read · Jun 20

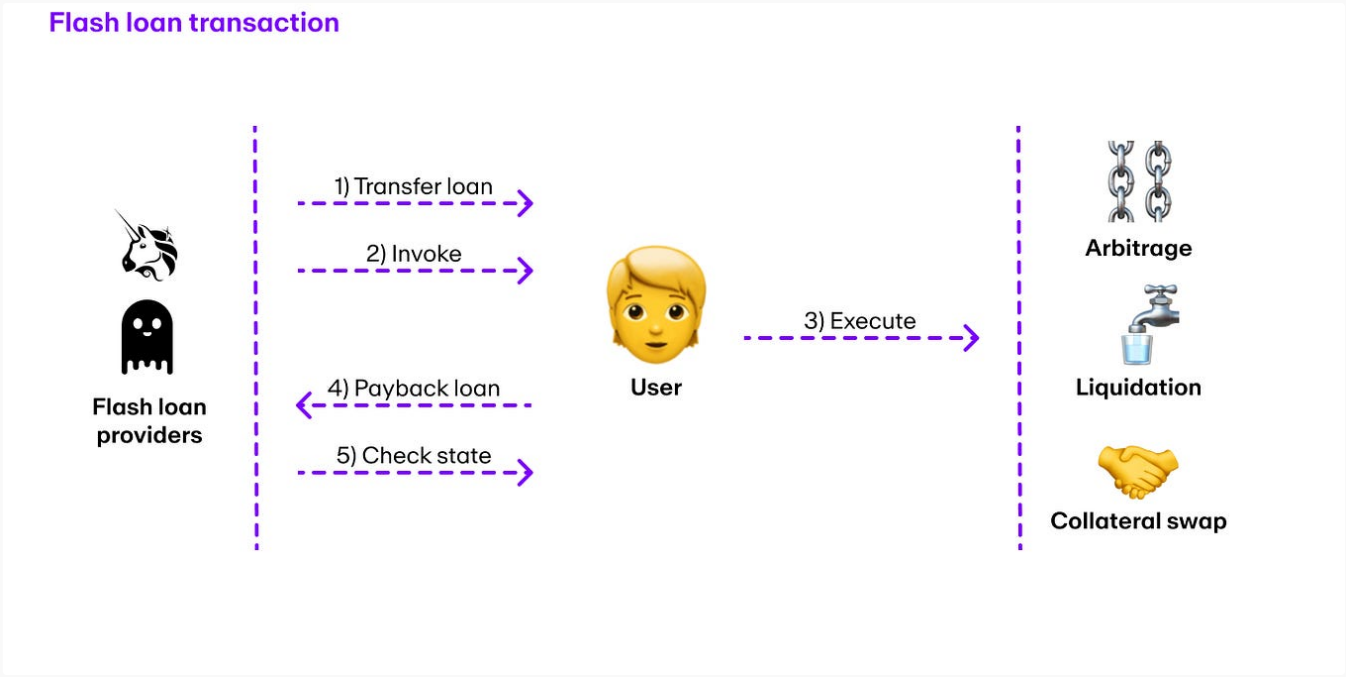
Heuss

Critical NFT Bridge Vulnerability : Potential Theft of Deposited NFTs

Already back with my second post, this vulnerability will be easier to explain (and also was easier to catch as I reported it the same day...

3 min read · Jul 27

68



Seyyed Ali Ayati

Understanding Flash Loan Attacks Through a Practical Example

A Comprehensive Case Study of a Flash Loan Attack

10 min read · Aug 30

2

See more recommendations