Solana‘s
Wormhole Rekt

Andrej Lukačovič

CONTENTS

LIST OF FIGURES

Solana‘s Wormhole Rekt

I. Introduction

The concept of the wormhole is based on the idea of a tunnel connecting two different points in space-time dimension, allowing objects to travel between them. Similarly, in the context of blockchain, a wormhole serves as a tunnel connecting two different blockchain networks, allowing the transfer of assets between them. When a user sends assets from one blockchain network to another via wormhole, the assets are first locked on the originating network, and then minted on the receiving network. Once the assets have been successfully transferred, they can be traded, used for applications, or withdrawn from the receiving network back to the original one. The Wormhole bridge was hacked on February 2nd, 2022, the attack exploited unpatched Rust programs in Solana that were manipulated into crediting 120k wETH.

II. Architecture Overview

[1]Wormhole is a complex ecosystem with several noteworthy components. Before we go into each component in depth, let's talk about the names of the major pieces and how they fit together.

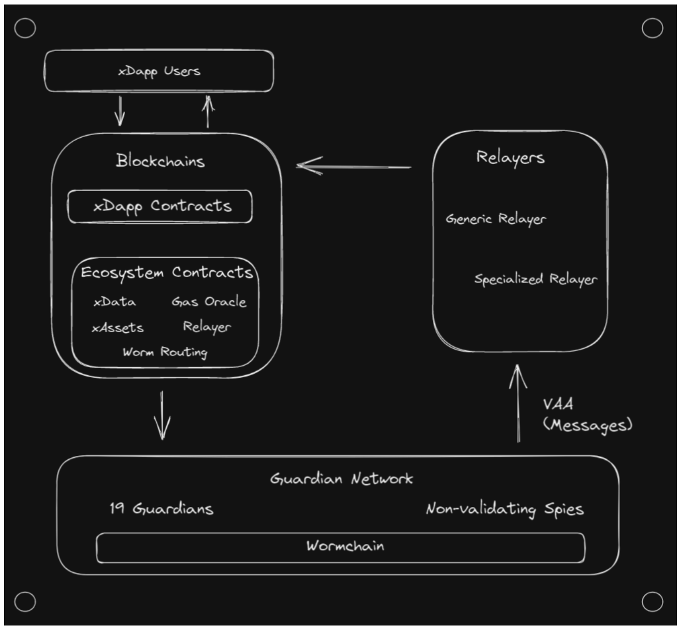

Fig. 1. Architecture Overview

[1]https://book.wormhole.com/wormhole/2_architectureOverview.html

A. On-Chain Components

- **xDapp Contracts** - Contracts developed by xDapp developers. They receive transactions from the end user and then interact with other xDapp contracts and Wormhole Ecosystem Contracts in order to provide their service.
- **Ecosystem Contracts** - Contracts subject to Wormhole governance which live inside the Wormhole Ecosystem. Their job is to provide the feature suite of Wormhole to xDapp developers.
  - **Core Contracts** - Primary ecosystem contracts. These are the contracts that the Guardians observe and which fundamentally allow for cross-chain communication.
  - **xAsset** Contracts - Contracts that allow normal tokens to be converted to xAssets and enable these xAssets to be bridged.
  - **Relay Contracts** - Contracts that allow xDapps to send messages to a specific blockchain via the decentralized Generic Relayer network.
  - **Gas Oracle** - Oracle for recommended fair gas prices across the ecosystem.
  - **Worm Router Contracts** - Contracts that allow developers to make their Dapp an xDapp that users on any Wormhole-supported chain can interact with purely through client-side code.

B. Off-Chain Components

- **Guardian Network** - Validators that exist in their own p2p network. Guardians observe the Core Contract on each supported chain and produce VAAs (signed messages) when those contracts receive an interaction.
- **Guardian** - One of 19 validators in the Guardian Network that contributes to the VAA

multi-signature.

- **Spy** - Validators on the Guardian Network which are not part of the Guardian set. A spy can observe and forward network traffic, which helps scale up VAA distribution.
- **VAAs** - Verifiable Action Approvals (VAAs) are the key piece of data in the Wormhole ecosystem, containing the messages emitted by xDapps along with information such as what contract emitted the message. The VAAs are signed by the Guardians and need 13/19 signatures to be considered authentic.
- **Specialized Relayers** - Relayers that only handle VAAs for a specific protocol or xDapp. They can execute custom logic off-chain, which can reduce gas costs and increase cross-chain compatibility. Currently, xDapp developers are responsible for developing and hosting specialized relayers.
- **Generic Relayers** - A decentralized relayer network which delivers messages that are requested on-chain via the Wormhole Relay Contract.
- **Wormchain** - A purpose-built cosmos blockchain which aids the Guardian Network and allows for formal interaction with the Guardians.

III. How does it work?

The Core Bridge Contract provides functions for emitting messages. The 19 guardians each observe messages emitted by the Core Contracts continuously and sign on the messages (e.g., a message like "Alice just sent an ETH to Wormhole on Ethereum, and she wanted Wormhole to send an ETH to Bob on Solana). The guardians each hold equal weight. When a supermajority (13/19) of them sign a message, the guardian network produces a Verifiable Action Approval (VAA), which serves as proof for Wormhole to deliver the same message on the target chain.

A. Two VAA components

- Header: is used by the Core Contract to determine the authenticity of the VAA (using signatures), but can generally be ignored by other consumers.

```
byte         version
u32          guardian_set_index
u8           len_signatures
[][66]byte   signatures
```

- Body: The payload byte array contains the message content. For example, a token Transfer, includes the transfer amount, token address, token chain, the recipient address, the target chain ID, transfer fee, etc

```
u32          timestamp
u32          nonce
u16          emitter_chain
[32]byte     emitter_address
u64          sequence
u8           consistency_level
[]byte       payload
```

Once a VAA is produced, it will be stored in the guardian network. Each VAA is uniquely indexed by its emitterChain, emittedAddress, and sequence, and can be obtained by querying a guardian node (via an RPC API) with this information. Anyone can query a guardian node, and the returned VAA bytes can be submitted by anyone to the target chain to complete the message.

B. Exemplary token transfer of 0.1 ETH from Ethereum to Solana

1) Transaction (Ethereum), the user sends 0.1 ETH to the Wormhole Token Bridge by calling wrapAndTransferETH with parameters specifying the recipient chain (Solana 0x01), recipient, arbiterFee (i.e., the relayer fee if used) and nonce.
2) This transaction invokes Wormhole Core Bridge internally and emits a message containing sequence, nonce, payload, and consistencyLevel.
3) The guardians observe the message emitted above and produce a VAA.
4) The VAA is retrieved from the guardian network and used to call Wormhole Core Bridge Contract on Solana.
5) Transaction (Solana), the VerifySignatures function on the Solana Wormhole Core Bridge is invoked by a signer with the VAA to create a SignatureSet.
   - **This transaction also invokes the precompiled Secp256k1 SigVerify Precompile to verify the guardian signatures in the VAA.**
6) Transaction (Solana), after all, signatures in the VAA are verified, the PostVAA function can be invoked to create a message account, which uniquely identifies the transferred message.
7) Transaction (Solana) Finally, the CompleteWrapped function on the Wormhole Token Bridge is invoked to complete the transfer. As we can see amount 0.1 of ETH - Ether (Portal), is assigned to Address.

IV. So how did the attack happen?

A. General idea

The general idea behind the attack is spoofing. Clearly in the example above, at some point there is a need to check all provided signatures by calling verify_signatures. This function call (at the time of the attack) already deprecated function load_instruction_at, to check if the instruction is for the secp256k1 program. But as I mentioned above the function was deprecated on January 13th, by looking into commits, because it did not check the signature verification was performed by the correct system address.

B. What did the attacker do?

From the transaction, we can observe that the provided **Account3** from Input Accounts corresponds to spoofed address and not to Sysvar: Instructions as shown for example in this transaction. The attacker abused this to inject his own fake data mimicking a validation result from the **Secp256k1** program. The verify_signatures function is meant to take a set of signatures provided by the guardians and pack it into a **SignatureSet**. But it doesn't actually do any of the verification itself. Instead, it delegates that to the **Secp256k1** program. This meant that the attacker could create his own account which stored the same data that the Sysvar: Instructions would have stored, and substituted that account in the vulnerable part mentioned above. That's exactly what the attacker did. Hours earlier, he created the account which contained **a single serialized instruction corresponding to a call to the Secp256k1 contract**. Once he had the fake ‘SignatureSet‘, it was trivial to use it to generate a valid VAA and trigger an unauthorized mint to his own account.