PROJECT

HaloDAO Bridge Handler

CLIENT

HaloDAO

DATE

August 2021

REVIEWERS

Andrei Simion

@andreiashu

Daniel Luca

@cleanunicorn

Table of Contents

Details

- **Client** HaloDAO
- **Date** August 2021
- **Lead reviewer** Andrei Simion (@andreiashu)
- **Reviewers** Daniel Luca (@cleanunicorn), Andrei Simion (@andreiashu)
- **Repository:** HaloDAO Bridge Handler
- **Commit hash** `2c54d69a75fddaabc97eb140509c112ec8575828`
- **Technologies**
  - TypeScript

Issues Summary

SEVERITY	OPEN	CLOSED
Informational	0	0
Minor	0	0
Medium	0	0
Major	2	0

Executive summary

This report represents the results of the engagement with **HaloDAO** to review **HaloDAO Bridge Handler**.

The review is part of a broader engagement with HaloDAO that includes several other components from the HaloDAO ecosystem (Halo Rewards, Halo AMM, Halo Bridge, Halo Bridge Handler).

The full review (across above-mentioned repositories) was conducted over the course of **2 weeks** from **16th of August to 27th of August, 2021**. We spent a total of **20 person-days** reviewing the code.

Review progress and strategy

The HaloDAO Bridge sits between Ethereum mainnet and any other EVM compatible chain - Polygon being the first Layer 2 chain targeted for implementation.

We started by going through the Solidity smart contracts inside the HaloDAO Token Bridge repository and reading the TypeScript code within the HaloDAO Bridge Handler repository. To process transfers from Chain A to Chain B (e.g. Ethereum mainnet to Polygon), the Bridge Handler uses a service (HAL) that helps pass Events from a chain to a webhook endpoint. Specifically, when a `DepositReceived` Event is emitted from the `PrimaryBridge` smart contract, the HAL service will call the HaloDAO Bridge Handler endpoint. The Bridge Handler script calls the `mint` function on the `SecondaryBridge` smart contract to create the respective tokens on the destination chain.

Because of the nature of decentralized blockchains, it is critical that the code that interacts with any of the chains involved in the transfer process are aware of *chain reorganization* (or *reorg*).

While discussing with the team, it became clear that the HAL service waits for three block confirmations before it considers a block as finalized. This is nowhere near the desired safety level for a cross-chain Bridge. For example, centralized exchanges use a minimum of 10 confirmations for any Ethereum / ERC20 transfer (we detailed this in issue #1).

The next question we asked ourselves was what should be a safe amount of block confirmations to consider a block as finalized for each of the chains involved. The Polygon chain has much faster block times and therefore requires a higher number of confirmations. We detailed our findings of this in issue #1.

We continued to review the code and communicate with the team while creating an overview of the architecture to help us better understand its trust model.

Trust model

We identified a few essential parts of the system that hold increased trust and are critical to the system's correctness and well behavior.

Reliance on an external party for the safety of transactions and funds

The HAL service has complete control over funds on all chains supported by the HaloDAO Bridge. For example, if HAL is compromised, an attacker **can drain funds** out of the Ethereum mainnet `PrimaryBridge` contract.

Centralised control of smart contracts and transfer handling scripts

Currently, both the blockchain smart contracts and the HaloDAO Bridge Handler scripts are controlled by the HaloDAO team. If the (multi-sig) account that controls the smart contracts is compromised, user funds are at risk.

The Bridge Handler will have to run on a server (AWS infrastructure). Therefore, the cloud account(s) that have access to these scripts are also vulnerable to an attack.

Scope

The initial review focused on the HaloDAO Bridge Handler repository, identified by the commit hash `2c54d69a75fddaabc97eb140509c112ec8575828`.

Includes:

- code/database/migrations/20210709195024_bridge_contracts.ts
- code/database/migrations/20210713001648_mint_requests.ts
- code/database/migrations/20210721125855_burns.ts
- code/database/migrations/20210721130355_redeem_requests.ts
- code/database/migrations/20210729010706_webhook_triggers.ts
- code/database/migrations/20210706173702_deposits.ts
- code/database/knex.ts
- code/database/seeds/bridge_contracts.ts
- code/database/seeds/webhook_triggers.ts
- code/app/middlewares/ListMetadata.ts
- code/app/utils/FeeCalculator.ts
- code/app/utils/logger.ts
- code/app/utils/message.ts
- code/app/utils/RepositoryQueryUtil.ts

- code/app/repositories/WebhookTriggerRepository.ts
- code/app/repositories/MintRequestRepository.ts
- code/app/repositories/DepositRepository.ts
- code/app/repositories/BaseRepository.ts
- code/app/repositories/RedeemRequestRepository.ts
- code/app/repositories/BurnRepository.ts
- code/app/repositories/BridgeContractRepository.ts
- code/app/external-services/SQSService.ts
- code/app/external-services/SecretsManagerService.ts
- code/app/container.ts
- code/app/factories/ParentBridge.ts
- code/app/factories/SecondaryBridge.ts
- code/app/factories/PrimaryBridge.ts
- code/app/factories/ChainProvider.ts
- code/app/factories/TokenContract.ts
- code/app/model/dto/MintRequest.ts
- code/app/model/dto/BridgeContract.ts
- code/app/model/dto/WebhookTrigger.ts
- code/app/model/dto/Burn.ts
- code/app/model/dto/Transaction.ts
- code/app/model/dto/Deposit.ts
- code/app/model/dto/RedeemRequest.ts
- code/app/model/vo/responseVo.ts
- code/app/model/vo/apiQueryVo.ts
- code/app/handler.ts
- code/app/env.ts
- code/app/handlers/SecondaryBridgeHandler.ts
- code/app/handlers/DepositHandler.ts
- code/app/handlers/MintRequestHandler.ts
- code/app/handlers/BurnHandler.ts
- code/app/handlers/RedeemRequestHandler.ts
- code/app/handlers/PrimaryBridgeHandler.ts
- code/app/services/SecondaryBridgeService.ts
- code/app/services/BurnService.ts
- code/app/services/RedeemRequestService.ts
- code/app/services/MintRequestService.ts
- code/app/services/DepositService.ts
- code/app/services/PrimaryBridgeService.ts
- code/tsconfig.json

Issues

Minting to opposing chain can lead to loss of funds

Status Open Severity Major

Description

A Bridge token transfer happens between two EVM compatible blockchains, each having its own finality properties.

When the Bridge Handler is notified of a new transfer from the *source* chain, it will issue a mint call on the *opposing chain*:

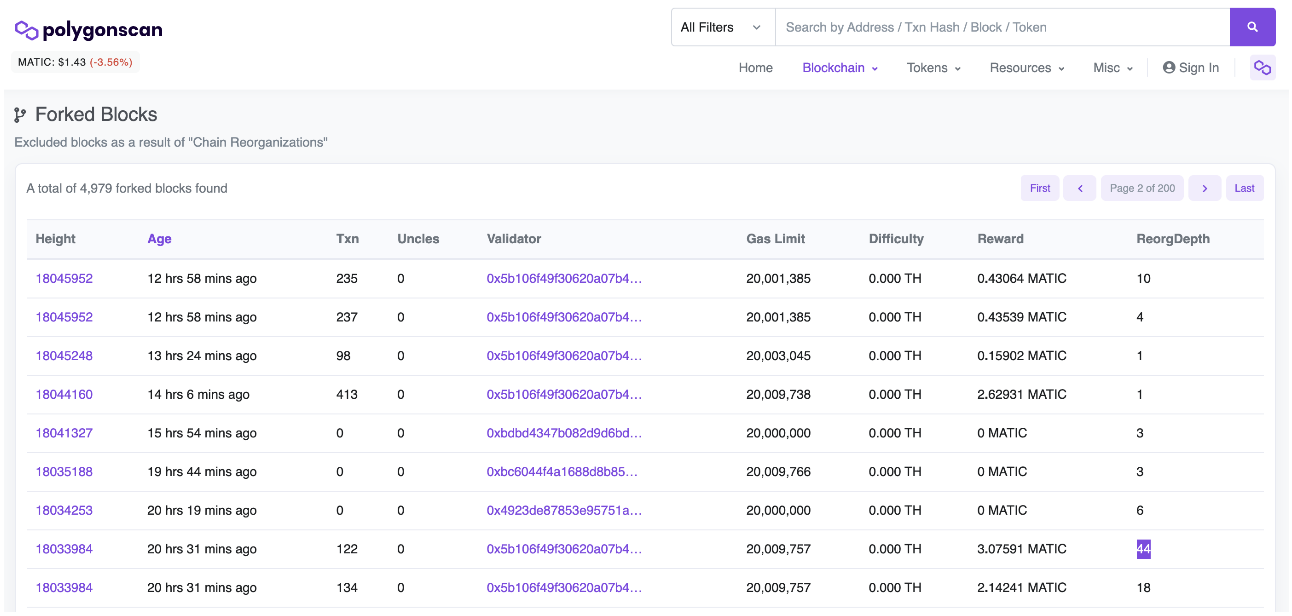
code/app/factories/SecondaryBridge.ts#L34-L48

```
	public async mintToOpposingChain(
		recipientAddress: string,
		amount: number,
	): Promise<ISignedMintTx> {
		const METHOD = '[mintToOpposingChain]'
		log.info(`${TAG} ${this.chainId} ${this.bridgeContractAddress} - ${METHOD}`)

		let result

		try {
			result = await this.bridgeContract.populateTransaction.mint(
				recipientAddress,
				amount,
			)
		} catch (BridgeError) {
```

The issue with this approach is that the code above assumes that the write operation on the *opposing chain* if confirmed, is also finalized. This is not the case. For example, when the opposing chain is Polygon, reorgs happen very frequently (and with high reorg depth, below we can see a 44 depth):



Forked Blocks

Excluded blocks as a result of "Chain Reorganizations"

A total of 4,979 forked blocks found

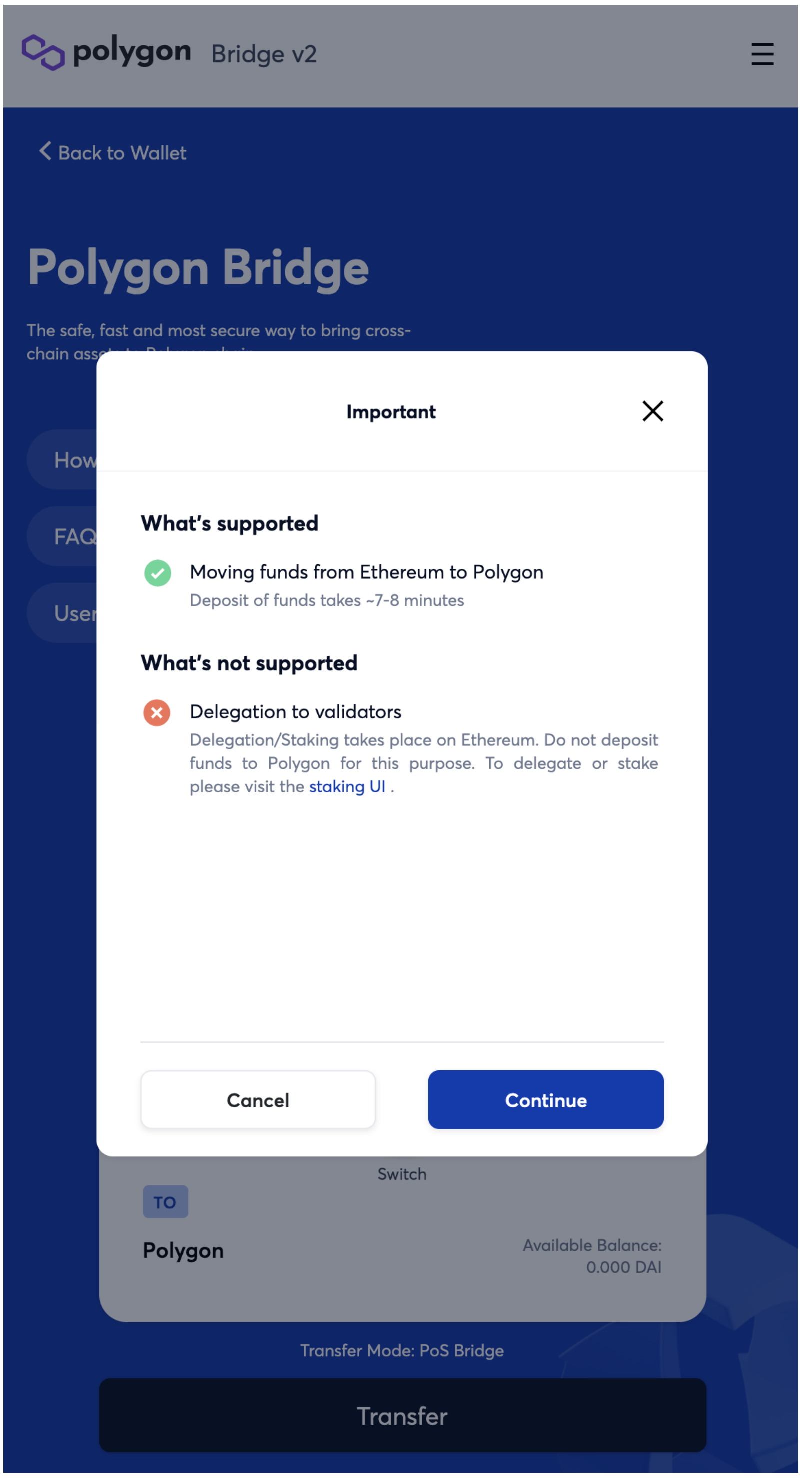
Height	Age	Txn	Uncles	Validator	Gas Limit	Difficulty	Reward	ReorgDepth
18045952	12 hrs 58 mins ago	235	0	0x5b106f49f30620a07b4...	20,001,385	0.000 TH	0.43064 MATIC	10
18045952	12 hrs 58 mins ago	237	0	0x5b106f49f30620a07b4...	20,001,385	0.000 TH	0.43539 MATIC	4
18045248	13 hrs 24 mins ago	98	0	0x5b106f49f30620a07b4...	20,003,045	0.000 TH	0.15902 MATIC	1
18044160	14 hrs 6 mins ago	413	0	0x5b106f49f30620a07b4...	20,009,738	0.000 TH	2.62931 MATIC	1
18041327	15 hrs 54 mins ago	0	0	0xbdbd4347b082d9d6bd...	20,000,000	0.000 TH	0 MATIC	3
18035188	19 hrs 44 mins ago	0	0	0xbc6044f4a1688d8b85...	20,009,766	0.000 TH	0 MATIC	3
18034253	20 hrs 19 mins ago	0	0	0x4923de87853e95751a...	20,000,000	0.000 TH	0 MATIC	6
18033984	20 hrs 31 mins ago	122	0	0x5b106f49f30620a07b4...	20,009,757	0.000 TH	3.07591 MATIC	44
18033984	20 hrs 31 mins ago	134	0	0x5b106f49f30620a07b4...	20,009,757	0.000 TH	2.14241 MATIC	18

Recommendation

Ensure that code that writes to the opposing chain is aware of the difference between a transaction that has been confirmed and one that can be considered finalized.

Regarding how many confirmations to wait before a block should be considered finalized, it's best to err on the side of caution. We recommend finding out what other bridge platforms or centralized exchanges that work with each of the opposing chains do.

The Matic/Polygon Bridge will wait for up to 8 minutes before minting a transfer from Ethereum mainnet to Polygon chain - given that the average block time on Polygon is around 2.5 seconds this would mean waiting for a **minimum of 192 blocks depth** before considering a transaction in it as finalized.


polygon Bridge v2

Back to Wallet

Polygon Bridge

The safe, fast and most secure way to bring cross-chain ass

Important

What's supported

Moving funds from Ethereum to Polygon

Deposit of funds takes ~7-8 minutes

What's not supported

Delegation to validators

Delegation/Staking takes place on Ethereum. Do not deposit funds to Polygon for this purpose. To delegate or stake please visit the staking UI .

References

- Polygon Blocks Forked
- Reorgs and Other Blockchain Quirkiness: Polyroll Update v2.01
- Matic/Polygon Bridge
- Polygon Block Time Chart

HAL service might cause a double-spend in the Bridge - can lead to loss of funds

Status Open Severity Major

Description

The HaloDAO bridge uses the HAL service in order to initiate a token transfer from the source chain to a destination chain.

When the source chain is Ethereum mainnet, the Halo team informed us that the HAL service uses 3 block confirmations before it fires a notification to the Halo Bridge webhook handler.

The issue here is that 3 confirmations are not nearly enough to consider a block as final. It was trivial to find a block with a Re-Org depth of 3 blocks within the last 2 months.

Granted, the Ethereum PoW blockchain only gives Probabilistic Finality - the probability that a transaction will not be reverted increases as the block which contains that transaction sinks deeper into the chain. Therefore we want to aim for a much higher block confirmation number than what HAL service currently has.

The more general issue is that the HAL service, with its current settings, is targeted more towards end-user and non-critical types of applications, where an extra event fired for a block that does not end up in the main chain, can be handled gracefully/ignored. In the case of a cross-chain bridge, this type of issue can cause a double-spend transaction with the consequence of **loss of funds of its users**.

Recommendation

A cross-chain bridge, within the context of block finality, is not different than a centralized exchange: security needs to be paramount.

Therefore block finality should be increased to match what other reputable exchanges use for ERC20 token deposits:

Coinbase	Kraken	Binance	FTX
35	20	12	10

Additionally, we recommend having a Service Level Agreement (SLA) with a service provider like HAL, that clearly outlines the availability and quality of the service provided. As an example, separate infrastructure from the end-user HAL infrastructure might mean that HALO Bridge is not affected when HAL experiences load from their free / end-user accounts.

References

- Finality in Blockchain Consensus
- Re-Org depth of 3 blocks
- Kraken Cryptocurrency deposit processing times

- Coinbase Uniswap deposit processing times
- Binance Reduces the Number of Confirmations Required for Deposits & Withdrawals on BTC and ETH Networks
- FTX Blockchain Deposits and Withdrawals

License