Planetarium x Theori

NineChronicles.EthBridge Security Audit

: Final Report

August 2021

Theori

Table of Contents

1. Project Overview

1.1 Executive Summary

Starting on August 9th, 2021, Theori assessed the bridge service of NineChronicles for two and a half weeks. The purpose of this audit was to identify security issues and establish the appropriate measures for improvement. For this, we examined the NineChronicles.EthBridge source code to examine the internal process, identify security issues, and give recommendations.

- Source code received: 08/09 (Mon)
- Static code analysis: 08/09 (Mon) ~ 08/24 (Tue)
- Project report: 8/25 (Wed)

We identified attack surfaces by understanding the NineChronicles bridge services and threat modelling, and proceeded with the source code audit. Through this, we were able to identify various issues based on threat scenarios which can occur in NineChronicles Bridge service.

We identified **four security issues,** that include actual threats which can lead to financial damage by exhausting all Ethereum NineChronicles-bridge has.

1.2 Scope of Work

The audit covered smart contracts and back-end source code for bridging NCG and WNCG. Only code within the below source code repository was reviewed. We used static and dynamic analysis to identify issues, mainly focusing on issues that may allow malicious users to make profit.

Github repo	Commit hash
NineChronicles.EthBridge	• https://github.com/planetarium/NineChronicles.EthBridge/commit/49439790c5c6bebfc65f06791fb25e8f03f66bca

Component	Description	Scope
bridge	Bridge back-end (NCG ↔ WNCG)	In scope
contract	WrappedNCG smart contract based on ERC-20	In scope
bridge-www	Bridge front-end for testing	Out of scope

1.3 Summary of Findings

As a result of the audit, We found four issues in the EthBridge. Among which there are high severity issues that can force EthBridge to terminate. Due to the nature of EthBridge that bridges cryptocurrency, immediate patch is required. We recommend considering safety from a long-term perspective for all the patches.

#	Finding ID	Title	CWE Type	Severity
1	FIXED (2021.09.01) BRIDGE-001	**NineChroniclesTransferredEventMonitor Denial of Service (DoS) when receiving malicious transaction with recipient address set to zero**	CWE-248 (Uncaught Exception)	High
2	FIXED (2021.09.01) BRIDGE-002	**Exhausting bridge admin's Ethereum as a transaction fee for user's numerous mint requests due to absence of bridging fee**	CWE-400 (Uncontrolled Resource Consumption)	High
3	FIXED (2021.09.01) BRIDGE-003	**Burn events are ignored when transactions are made with an empty memo in NCGTransferredEventObserver**	CWE-705 (Incorrect Control Flow Scoping)	Medium
4	FIXED (2021.09.01) BRIDGE-004	**NineChroniclesTransferredEventMonitor Denial of Service (DoS) when an attacker sends a malicious transaction with an improperly formatted recipient address**	CWE-248 (Uncaught Exception)	High

1.4 Severity Categories

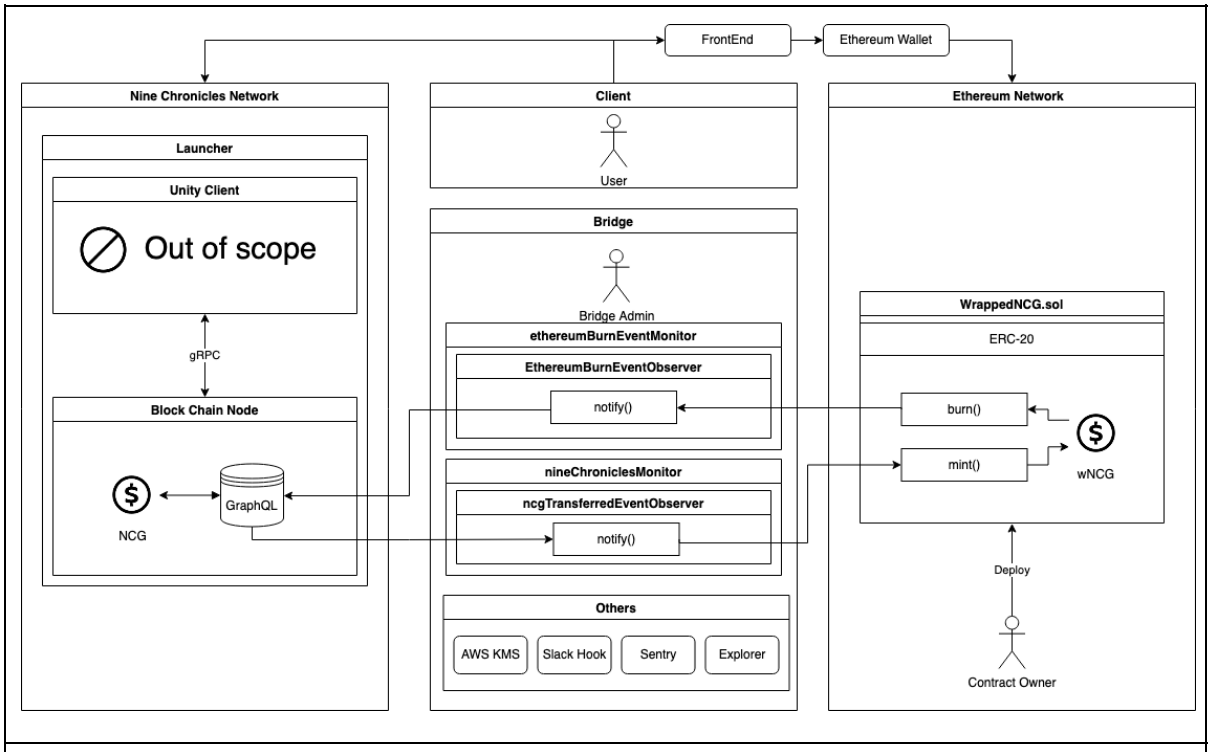
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g. Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.

Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	The issue is not currently recognized as a vulnerability, but may develop into a potential security threat as the service is further developed.

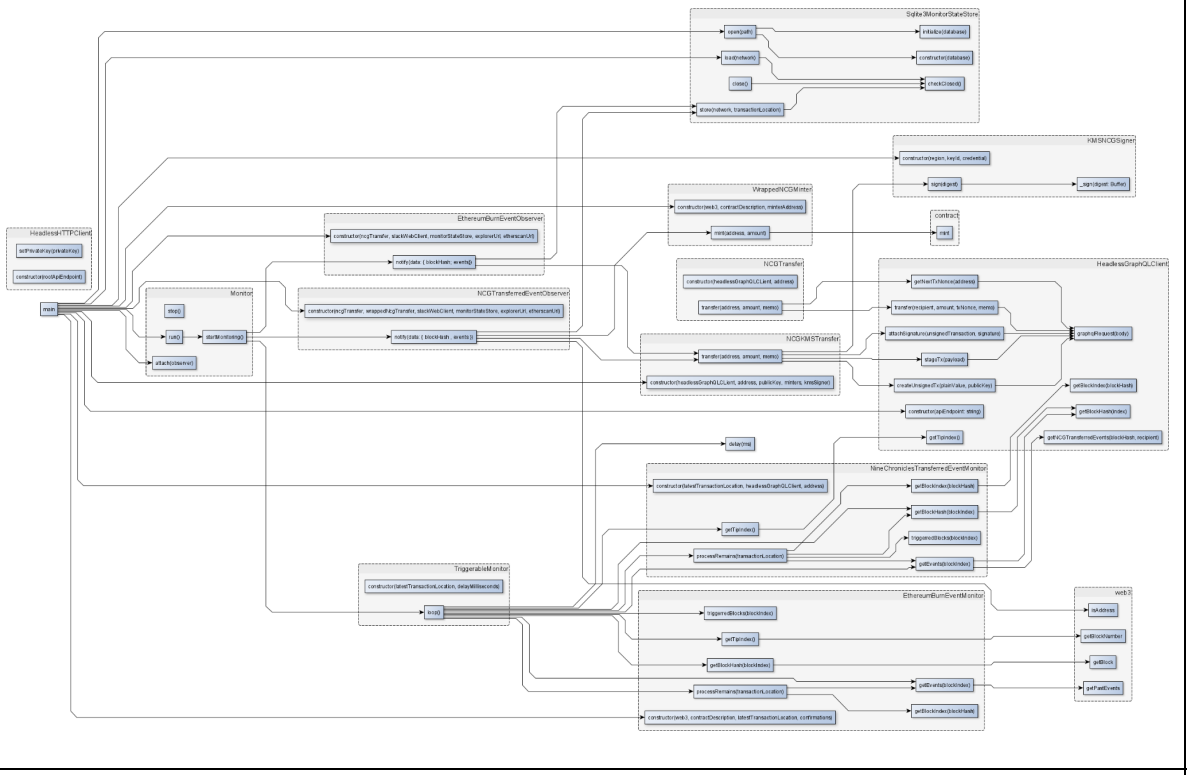
2. Bridge Audit Summary

2.1 Bridge Overview

Before starting the security audit, We drew a schematic of the bridge system. Based on this, we identified the attack surfaces and threat scenarios. Bridge is in charge of exchange and circulation between NCG in Nine Chronicles Network and WNCG that is managed by Ethereum contract written in ERC-20 standard.



Bridge Overview



Bridge call graph

2.2 Attack Surface

Before starting the security audit of NineChronicles Bridge, Theori identified the attack surfaces of the service based on the schematic of the bridge system and call-graph. We categorized the attack surfaces into Ethereum and Nine Chronicles network end-points which can be accessed by a user.

Network	User Input	Description
Ethereum (Smart contract actions)	Amount	WNCG amounts to burn.
	Recipient	NCG recipient address.
Nine Chronicles (Transfer)	Amount	NCG amounts to mint.
	Recipient (memo)	WNCG recipient address.

2.3 Threat Scenario

The following are threat scenarios that can occur on the identified attack surfaces.

Category	Threat Factor	Identified Issue (Corresponding threat factor)
General	Access Control 1. Bypass Permission Validation (Authorization)	N/A
	Credential 1. Hard coded credential 2. Slack Token, KMS, Sentry key exposure 3. Exposure of sensitive information in transactions/contracts/git	N/A
	Programmatic Issue 1. Incorrect calculation of loop 2. Insufficient exception handling 3. Unreachable code 4. SQL Injection (SQLite) 5. Miscellaneous Issue	• BRIDGE-003 - (1) • Recommendation #1 - (3) • Recommendation #2 - (2) • Recommendation #3 - (5) • Recommendation #4 - (1)
Bridge	Fraudulent 'burn' and 'mint' 1. Duplicate 'burn' and 'mint' 2. Miscalculated amount (floating-point, decimal value)	N/A
	Cryptographic issues 1. Arbitrary transaction signing	N/A
	Blockchain 1. Use of proper block confirmation count 2. Invalid handling of block reorganization 3. Invalid address handling on transaction	• BRIDGE-001 - (3) • BRIDGE-004 - (3)
	Bridging Fee 1. Transaction fee abusing (more/less)	• BRIDGE-002 - (1)
Smart Contract	ERC-20 1. Use of non-standard code/functionality	N/A

3. Detail of findings

3.1 BRIDGE-001: NineChroniclesTransferredEventMonitor Denial of Service (DoS) when receiving malicious transaction with recipient address set to zero

Finding ID	Summary	CWE	Severity
BRIDGE-001	When the bridge tries to mint WNCG tokens to an address set to zero, the mint method call will fail and force NineChroniclesTransferredEventMonitor to terminate. Since the attacker can terminate NineChroniclesTransferredEventMonitor, all other user's wrapping events (NCG → WNCG) will be ignored until the bridge restart.	CWE-248 (Uncaught Exception)	High

Finding Details

[Introduction]

The NineChroniclesTransferredEventMonitor in the bridge service will be terminated by the attacker's malicious transaction (with the recipient address set to zero). The code below is the bridge minting WNCG to the recipient address on the WrappedNCG contract in the Ethereum network. Note that the recipient address is the user's Ethereum wallet address, so the address is a user-controllable value.

```
// File: bridge/src/observers/nine-chronicles.ts
// Lines: 30 ~ 53

async notify(data: { blockHash: BlockHash, events: (NCGTransferredEvent &
TransactionLocation)[] }): Promise<void> {
    const { blockHash, events } = data;

    if (events.length === 0) {
        await this._monitorStateStore.store("nineChronicles", { blockHash,
txId: null });
    }

    for (const { blockHash, txId, sender, amount: amountString, memo:
```

```
recipient, } of events) {
        const amount = new Decimal(amountString).mul(new Decimal(10).pow(18));
        if (recipient === null || !isAddress(recipient) || !amount.isFinite()
|| amount.isNaN()) {
            const nineChroniclesTxId = await this._ncgTransfer.transfer(sender,
amountString, "I'm bridge and you should transfer with memo having ethereum
address to receive.");
            console.log("Valid memo doesn't exist so refund NCG. The
transaction's id is", nineChroniclesTxId);
            return;
        }

        const { transactionHash } = await
this._wrappedNcgTransfer.mint(recipient, amount); // Minting.
        console.log("Receipt", transactionHash);
        await this._monitorStateStore.store("nineChronicles", { blockHash, txId
});
        await this._slackWebClient.chat.postMessage({
            channel: "#nine-chronicles-bridge-bot",
            ...new WrappedEvent(this._explorerUrl, this._etherscanUrl, sender,
recipient, amountString, txId, transactionHash).render()
        });
    }
}
```

NCGTransferredEventObserver::notify()

In the minting process, the mint() function will be called with `address` and `amount` values.

```
// File: bridge/src/wrapped-ncg-minter.ts
// Lines: 22 ~ 28

async mint(address: string, amount: Decimal): Promise < TransactionReceipt > {
    //NOTICE: This can be a problem if the number of digits in amount exceeds
9e+14.
    //more detail: https://mikemcl.github.io/decimal.js/#toExpPos
    Decimal.set({ toExpPos: 900000000000000 });
    console.log(`Minting ${amount.toString()}
${this._contractDescription.address} to ${address}`);
    return this._contract.methods.mint(address,
this._web3.utils.toBN(amount.toString())).send({ from: this._minterAddress });
// [!]
}
```

WrappedNCGMinter::mint()

The WrappedNCG contract is based on the ERC-20 contract. Since, the ERC-20 contract cannot mint any token to the zero address, the minting process raises an error (*UnhandledPromiseRejectionWarning: Error: execution reverted: ERC20: mint to the zero address*) and the NineChroniclesTransferredEventMonitor will be terminated by an unhandled exception.

```
// File:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/to
```

```
ken/ERC20/ERC20.sol#L252
// Line: 252

function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address"); // [!]
    _beforeTokenTransfer(address(0), account, amount);
    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);
    _afterTokenTransfer(address(0), account, amount);
}
```

_mint() at ERC-20

[Steps]

1. Setup environments for sending transaction (Bridge, Contracts, Node)
2. Send the malicious transaction using GraphQL query.

[PoC]

```
mutation{
    transfer(recipient: "[Bridge address]", amount: "0.01", txNonce: 83, memo:
"0x0000000000000000000000000000000000000000")
}
```

A malicious transaction with the recipient (memo) address set to zero.

[Result]

Transaction Details

Id
ceeadd972729e96f828120002de7e64081def2f71252601dc5b6705e13765560
Nonce
37
Public Key
03406bfe8bcffbe4b4755a7cd744cb9c5a88ff77148125c138c90a30c03d8c13f2
Signature
304402205202b6a67bc6a8110ed49c2613f2f6eafe908345d7f5703045620c1a0b6c051f02201075f8d24c83e84b3f3c275ddda6d10d5c626b6a0e79cad73bdb56cdb3c185bc
Signer
0x9bDaA64d1b33D1FCB81CCf4C6E048cef48Ea1204
Timestamp
오후 05:20
Updated Addresses
0x9bDaA64d1b33D1FCB81CCf4C6E048cef48Ea1204
0xD03523408F26F80f8C5aC0eB15485A7FA82f2C66
Block Reference
Actions

```
type_id: "transfer_asset"
▼ values: {} 4 keys
  ▼ amount: [] 2 items
    ▼ 0: {} 3 keys
        decimalPlaces: "<binary> 02"
      ▼ minters: [] 1 item
          0: "<binary> 47d082a115c63e7b58b1532d20e631538eafadde"
        ticker: "NCG"
      1: 1
  memo: "0x00000000000000000000000000000000000000000000"
  recipient: "<binary> d03523408f26f80f8c5ac0eb15485a7fa82f2c66"
  sender: "<binary> 9bdaa64d1b33d1fcb81ccf4c6e048cef48ea1204"
```

Malicious transaction details

```
[NineChroniclesTransferredEventMonitor] Execute triggerred block # 2158886
Minting 10000000000000000 0xFFeF90A0CD599e61c835cECdd1e1952b2F7272E0 to
0x0000000000000000000000000000000000000000
(node:28637) UnhandledPromiseRejectionWarning: Error: execution reverted:
ERC20: mint to the zero address
0x08c379a0000000000000000000000000000000000000000000000000000000000000002000000
0000000000000000000000000000000000000000000000000000000000001f45524332303a206d696e
7420746f20746865207a65726f206164647265737300
    at Object._fireError
(/home/attacker/NineChronicles.EthBridge/bridge/node_modules/web3-utils/lib/ind
ex.js:49:17)
    at sendTxCallback
(/home/attacker/NineChronicles.EthBridge/bridge/node_modules/web3-core-method/l
ib/index.js:539:30)
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/web3-core-requestma
nager/lib/index.js:299:20
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/web3-provider-engin
e/index.js:232:9
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/internal/once
.js:12:16
```

```
    at replenish
(/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/internal/eac
hOfLimit.js:61:25)
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/internal/each
OfLimit.js:71:9
    at eachLimit
(/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/eachLimit.js
:43:36)
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/internal/doLi
mit.js:9:16
    at end
(/home/attacker/NineChronicles.EthBridge/bridge/node_modules/web3-provider-engi
ne/index.js:211:5)
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/internal/once
.js:12:16
    at next
(/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/waterfall.js
:21:29)
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/internal/only
Once.js:12:16
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/internal/once
.js:12:16
    at next
(/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/waterfall.js
:21:29)
    at
/home/attacker/NineChronicles.EthBridge/bridge/node_modules/async/internal/only
Once.js:12:16
(Use `node --trace-warnings ...` to show where the warning was created)
(node:28637) UnhandledPromiseRejectionWarning: Unhandled promise rejection.
This error originated either by throwing inside of an async function without a
catch block, or by rejecting a promise which was not handled with .catch(). To
terminate the node process on unhandled promise rejection, use the CLI flag
`--unhandled-rejections=strict` (see
https://nodejs.org/api/cli.html#cli_unhandled_rejections_mode). (rejection id:
9)
(node:28637) [DEP0018] DeprecationWarning: Unhandled promise rejections are
deprecated. In the future, promise rejections that are not handled will
terminate the Node.js process with a non-zero exit code.
[EthereumBurnEventMonitor] Try to check trigger at 10860449
[EthereumBurnEventMonitor] Execute triggerred block # 10860447
[EthereumBurnEventMonitor] Try to check trigger at 10860450
[EthereumBurnEventMonitor] Skip check trigger current: 10860449 / tip: 10860449
[EthereumBurnEventMonitor] Try to check trigger at 10860450
```

Monitor terminated log

Risk

- An attacker can terminate NineChroniclesTransferredEventMonitor. Thus, the user's wrapping events (NCG → WNCG) will be ignored until bridge restart.

Root cause and solution
[Root cause] • Due to insufficient address validation, the bridge tries to mint WNCG tokens to the zero address via WrappedNCG contract based on ERC-20. • The bridge does not handle any exceptions about transaction sending. [Solution] • Handling exceptions when sending transactions. • WNCG minting should not be allowed to a zero address.
Fix
Planetarium fixed the issue by adding the validation of zero address. `// File: bridge/src/observers/nine-chronicles.ts` `// Lines: 18 ~ 20` `function isValidAddress(address: string): boolean {` `  return address.startsWith("0x") && isAddress(address) && address !== ZERO_ADDRESS;` `}` BRIDGE-001 fixed code

3.2 BRIDGE-002: Exhausting bridge admin's Ethereum as a transaction fee for user's numerous mint requests due to absence of bridging fee

Finding ID	Summary	CWE	Severity
BRIDGE-002	The bridge does not get fees from users when they request to mint WNCG. However, when the bridge mints WNCG at WrappedNCG contract on the Ethereum network, it will consume some fee (gas) from the bridge. So, Ethereum of the bridge can be depleted by an attacker by multiple mint requests.	CWE-400 (Uncontrolled Resource Consumption)	High

Finding Details

[Introduction]

In the code below, there are two check routines. The routines check whether the amount in the transaction is a finite number and whether it is NaN, respectively. However, the bridge does not have a routine for checking the minimum price for the bridge to pay the gas fee, which allows it to mint WNCG without considering Ethereum network transaction fee.

```
// File: bridge/src/observers/nine-chronicles.ts
// Lines: 30~45

async notify(data: { blockHash: BlockHash, events: (NCGTransferredEvent &
TransactionLocation)[] }): Promise<void> {
    const { blockHash, events } = data;

    if (events.length === 0) {
        await this._monitorStateStore.store("nineChronicles", { blockHash, txId:
null });
    }

    for (const { blockHash, txId, sender, amount: amountString, memo: recipient,
} of events) {
        const amount = new Decimal(amountString).mul(new Decimal(10).pow(18));
        if (recipient === null || !isAddress(recipient) || !amount.isFinite() ||
amount.isNaN()) {
            const nineChroniclesTxId = await this._ncgTransfer.transfer(sender,
amountString, "I'm bridge and you should transfer with memo having ethereum
address to receive.");
            console.log("Valid memo doesn't exist so refund NCG. The
transaction's id is", nineChroniclesTxId);
            return;
        }
```

```
        const { transactionHash } = await
this._wrappedNcgTransfer.mint(recipient, amount);
```

notify() function to handle the mint event.

As of today (Aug 19th, 2021), the gas used to transmit the mint transaction is worth about 0.56 USD. But bridge does not get fees from users when they request to mint WNCG. So, an attacker can request minting repeatedly to inflict financial damage (0.56 USD/transaction) to the bridge.

```python
from math import floor

NCG_PRICE = 4.38  # Token price USD (Aug 19th, 2021)
GAS_MINT = 0.56  # USD (Average 28 gwei @ mainnet)
GAS_BURN = 0.48  # USD (Average 28 gwei @ mainnet)
NCG_TRANSACITON_MIN = 0.01  # NCG


def getLoss(attacker_ncg):
    # NCG to WNCG
    global NCG_TRANSACITON_MIN, GAS_MINT
    tx_count = floor(attacker_ncg / NCG_TRANSACITON_MIN)
    return (tx_count, tx_count * GAS_MINT)  # Count, loss USD


def recoverNCG(attacker_ncg):
    # WNCG to NCG
    global NCG_PRICE, GAS_BURN
    revert_ncg = (round(attacker_ncg, 2) * NCG_PRICE) - GAS_BURN
    return revert_ncg / NCG_PRICE  # NCG


if __name__ == "__main__":
    total_tx_count = 0
    total_loss = 0
    attacker_ncg = 10  # Attacker's NCG (43.8 USD)
    while attacker_ncg > NCG_TRANSACITON_MIN:
        loss = getLoss(attacker_ncg)
        total_tx_count += loss[0]
        total_loss += loss[1]
        attacker_ncg = recoverNCG(attacker_ncg)
    print(f"[#] Count: {total_tx_count}")
    print(f"[#] Loss: {total_loss} USD")

    # python ./simulate.py
    # [#] Count: 45955
    # [#] Loss: 25734.8 USD
```

Loss simulation code (Assumption: an attacker has 10 NCG)

An attacker can make the bridge consume more gas by repeating the process of minting NCG and burning the generated WNCG back to NCG.
As a result of the above test code, if an attacker spent 10 NCG, it could inflict a loss of 25,734

USD to the bridge.

[Steps]

1. Send mint request which converts NCG to WNCG
2. Check transaction log

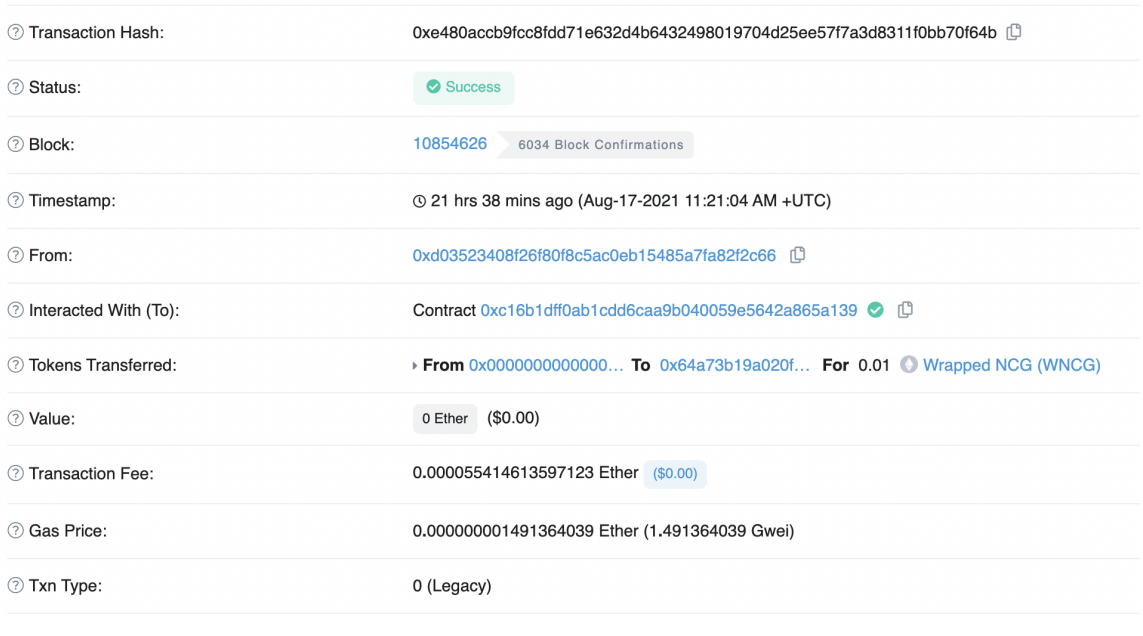
[PoC]

The below GraphQL query is a mint request which converts NCG to WNCG. If an attacker sends more mint requests, this results in the bridge spending more Ethereum as fee (gas).

```
mutation{
 transfer(recipient: "[Bridge address]", amount: "0.01", txNonce: 64, memo:
"[Attacker's ETH wallet address]")
}
```

Mint request which converts NCG to WNCG

[Result]



Etherscan transaction log of PoC (https://ropsten.etherscan.io/tx/0xe480accb9fcc8fdd71e632d4b6432498019704d25ee57f7a3d8311f0bb70f64b)

Risk

- Ethereum of the bridge can be depleted by an attacker if the attacker makes multiple mint requests while consuming a transaction fee (gas) that is greater than the market price of NCG.

Root cause and solution

[Root cause]

- The bridge mints WNCG without considering Ethereum network transaction fee.
- There is no limit on the minimum amount of NCG to mint.

[Solution]

- Design an incentive model. For example, (1) A user needs at least 1 NCG (≈ 4 USD) for minting. (2) Bridge owner takes 0.5 NCG (≈ 2 USD) from the user's amount per transaction as a fee.

Fix

Planetarium mitigated the issue by adding a bridging fee and limiting the range of mint amounts.

```
// File: bridge/src/observers/nine-chronicles.ts
// Lines: 102 ~ 111

// If exchangeFeeRatio == 0.01 (1%), it exchanges only 0.99 (= 1 - 0.01 = 99%)
of amount.
const fee = new Decimal(limitedAmount.mul(this._exchangeFeeRatio).toFixed(2));
const exchangeAmount = limitedAmount.sub(fee);
const ethereumExchangeAmount = exchangeAmount.mul(decimals);

console.log("limitedAmount", limitedAmount);
console.log("fee", fee);
console.log("exchangeAmount", exchangeAmount);

const { transactionHash } = await this._wrappedNcgTransfer.mint(recipient,
ethereumExchangeAmount);
```

BRIDGE-002 fixed code (Bridging fee)

```
// File: bridge/src/observers/nine-chronicles.ts
// Lines: 60 ~ 123

const amount = new Decimal(amountString);
const minimum = new Decimal(this._limitationPolicy.minimum); // 100 NCG
const maximum = new Decimal(this._limitationPolicy.maximum); // 100,000 NCG
const transferredAmountInLast24Hours = new Decimal(await
this._exchangeHistoryStore.transferredAmountInLast24Hours("nineChronicles",
sender));
// ...

if (amount.eq(0)) {
    // error
}

if (amount.cmp(minimum) === -1) {
    // minimum check
}
```

```
if (transferredAmountInLast24Hours.cmp(maximum) >= 0) {
   // 24 Hours maximum amount check
}

// ...

const { transactionHash } = await this._wrappedNcgTransfer.mint(recipient,
ethereumExchangeAmount);

// Logging amount
await this._exchangeHistoryStore.put({
   network: "nineChronicles",
   tx_id: txId,
   sender,
   recipient,
   timestamp: new Date().toISOString(),
   amount: limitedAmount.toNumber(),
});
```

BRIDGE-002 fixed code (Min/max amount check)

3.3 BRIDGE-003: Burn events are ignored when transactions are made with an empty memo in NCGTransferredEventObserver

Finding ID	Summary	CWE	Severity
BRIDGE-003	If the event handle loop in NCGTransferredEventObserver.notify() meets a transfer event without a recipient, the function returns immediately without processing the remaining events. Thus, the attacker can make the bridge ignore burn events.	CWE-705 (Incorrect Control Flow Scoping)	Medium

Finding Details

[Introduction]

When the event handle loop in NCGTransferredEventObserver.notify() meets an invalid event, the function must properly handle it and continue the loop. However, the function returns immediately without processing the rest of the events.

```
// File: bridge/src/observers/nine-chronicles.ts
// Lines: 30 ~ 53

async notify(data: { blockHash: BlockHash, events: (NCGTransferredEvent &
TransactionLocation)[] }): Promise<void> {
   const { blockHash, events } = data;

   if (events.length === 0) {
       await this._monitorStateStore.store("nineChronicles", { blockHash, txId:
null });
   }

   for (const { blockHash, txId, sender, amount: amountString, memo: recipient,
} of events) {
       const amount = new Decimal(amountString).mul(new Decimal(10).pow(18));
       if (recipient === null || !isAddress(recipient) || !amount.isFinite() ||
amount.isNaN()) {
           const nineChroniclesTxId = await this._ncgTransfer.transfer(sender,
amountString, "I'm bridge and you should transfer with memo having ethereum
address to receive."); // Refund NCG to attacker
           console.log("Valid memo doesn't exist so refund NCG. The
transaction's id is", nineChroniclesTxId);
           return; // [!] Ignore events behind the malicious transaction.
       }
```

```
        const { transactionHash } = await
this._wrappedNcgTransfer.mint(recipient, amount);
        console.log("Receipt", transactionHash);
        await this._monitorStateStore.store("nineChronicles", { blockHash, txId
});
        await this._slackWebClient.chat.postMessage({
            channel: "#nine-chronicles-bridge-bot",
            ...new WrappedEvent(this._explorerUrl, this._etherscanUrl, sender,
recipient, amountString, txId, transactionHash).render()
        });
    }
}
```

NCGTransferredEventObserver.notify()

NCGTransferredEventObserver.notify() function gets 'events' data and handles it. In the 'for' loop, the function enumerates events and checks whether the recipient of an event is not null. In this 'if' statement, the bridge refunds a user's NCG, and the 'notify()' function will be exited through the 'return' statement. As a result, the function skips all remaining events.

[Steps]

1. Make two transactions as follows.
    a. Transfer NCG to bridge minter without recipient (memo).
    b. Transfer NCG to bridge minter like a normal user's transaction that has a valid recipient.
2. Make sure that the first transaction (1-a) and second transaction (1-b) are mined from the same block.
3. Check the balance of the recipient's Ethereum address. Second transaction (1-b) might be ignored due to (1-a).

[PoC]

```
mutation{
   transfer(recipient: "0xD03523408F26F80f8C5aC0eB15485A7FA82f2C66", amount:
"2", txNonce:116 , memo: "")
}
 // and then
 mutation{
transfer(recipient: "0xD03523408F26F80f8C5aC0eB15485A7FA82f2C66", amount: "1",
txNonce:117 , memo: "0xEf3ea14080d8f0357F45abB7Fb0eeC6B4B8442a4")
}
```

Transactions 1-a and 1-b

[Result]

- First transaction: https://explorer.libplanet.io/9c-audit/transaction/?29da15b6ab94df5887330eb88d41a42d0104a16091e11f48d9705092036980a6

- Second transaction:
  https://explorer.libplanet.io/9c-audit/transaction/?1789ffb4c9c26953153b3820222e7fd4fb4785211a75a9713ec014155fcc6da1

```
[NineChroniclesTransferredEventMonitor] Execute triggerred block # 2159029
[NineChroniclesTransferredEventMonitor] Execute triggerred block # 2159030
Valid memo doesn't exist so refund NCG. The transaction's id is
ad38a27a16d8385da2d88b788fe02d08ab596ea1b146a753cf820ddb148cec36
[NineChroniclesTransferredEventMonitor] Try to check trigger at 2159031
[NineChroniclesTransferredEventMonitor] Skip check trigger current: 2159030 /
tip: 2159030
[EthereumBurnEventMonitor] Try to check trigger at 10861000
[EthereumBurnEventMonitor] Execute triggerred block # 10860999
[EthereumBurnEventMonitor] Try to check trigger at 10861001
[EthereumBurnEventMonitor] Skip check trigger current: 10861000 / tip: 10861000
[EthereumBurnEventMonitor] Try to check trigger at 10861001
[EthereumBurnEventMonitor] Execute triggerred block # 10861000
[EthereumBurnEventMonitor] Try to check trigger at 10861002
[EthereumBurnEventMonitor] Skip check trigger current: 10861001 / tip: 10861001
[NineChroniclesTransferredEventMonitor] Try to check trigger at 2159031
[NineChroniclesTransferredEventMonitor] Skip check trigger current: 2159030 /
tip: 2159030
[EthereumBurnEventMonitor] Try to check trigger at 10861002
[EthereumBurnEventMonitor] Execute triggerred block # 10861001
[EthereumBurnEventMonitor] Try to check trigger at 10861003
[EthereumBurnEventMonitor] Execute triggerred block # 10861002
[NineChroniclesTransferredEventMonitor] Try to check trigger at 2159031
[NineChroniclesTransferredEventMonitor] Skip check trigger current: 2159030 /
tip: 2159030
[EthereumBurnEventMonitor] Try to check trigger at 10861004
[EthereumBurnEventMonitor] Execute triggerred block # 10861003
[EthereumBurnEventMonitor] Try to check trigger at 10861005
[EthereumBurnEventMonitor] Skip check trigger current: 10861004 / tip: 10861004
[NineChroniclesTransferredEventMonitor] Try to check trigger at 2159031
[NineChroniclesTransferredEventMonitor] Try to check trigger at 2159032
```

Bridge skips second transaction(There is no 'Receipt' log)

Risk

- Bridge skips events behind the event without a recipient.

Root cause and solution

[Root cause]

- When NCGTransferredEventObserver.notify() meets invalid transfer event, it immediately returns, without processing the rest of the events.

[Solution]

- Use 'continue' instead of 'return'.

Fix

Planetarium fixed the issue by replacing ‘return’ with ‘continue’.

```
// File: bridge/src/observers/nine-chronicles.ts
// Lines: 48 ~ 54

if (recipient === null || !isValidAddress(recipient) || !amount.isFinite() ||
amount.isNaN()) {
   const nineChroniclesTxId = await this._ncgTransfer.transfer(sender,
amountString, "I'm bridge and you should transfer with memo having ethereum
address to receive.");
   console.log("Valid memo doesn't exist so refund NCG. The transaction's id
is", nineChroniclesTxId);
   continue;
}
const { transactionHash } = await this._wrappedNcgTransfer.mint(recipient,
amount);
```

BRIDGE-003 fixed code

3.4 BRIDGE-004: NineChroniclesTransferredEventMonitor Denial of Service (DoS) when an attacker sends a malicious transaction with an improperly formatted recipient address

Finding ID	Summary	CWE	Severity
BRIDGE-004	The internal library ('ethers') throws an error when the bridge tries to mint WNCG tokens to improperly formatted recipient addresses. Thus, the attacker can terminate NineChroniclesTransferredEventMonitor, and all other user's wrapping events (NCG → WNCG) will be ignored until the bridge restarts.	CWE-248 (Uncaught Exception)	High

Finding Details

[Introduction]

The NineChroniclesTransferredEventMonitor in the bridge service will be terminated by the attacker's malicious transaction with an improper recipient address. The code below is the bridge minting WNCG to the recipient address on the WrappedNCG contract at the Ethereum network. Note that, the recipient address is the user's Ethereum wallet address, so the address is a user-controllable value.

```
// File: bridge/src/observers/nine-chronicles.ts
// Lines: 30 ~ 53

async notify(data: { blockHash: BlockHash, events: (NCGTransferredEvent &
TransactionLocation)[] }): Promise<void> {
    const { blockHash, events } = data;

    if (events.length === 0) {
        await this._monitorStateStore.store("nineChronicles", { blockHash,
txId: null });
    }

    for (const { blockHash, txId, sender, amount: amountString, memo:
recipient, } of events) {
        const amount = new Decimal(amountString).mul(new Decimal(10).pow(18));
        if (recipient === null || !isAddress(recipient) || !amount.isFinite()
|| amount.isNaN()) { // Address validation.
            const nineChroniclesTxId = await this._ncgTransfer.transfer(sender,
amountString, "I'm bridge and you should transfer with memo having ethereum
```

```
address to receive.");
            console.log("Valid memo doesn't exist so refund NCG. The
transaction's id is", nineChroniclesTxId);
            return;
        }

        const { transactionHash } = await
this._wrappedNcgTransfer.mint(recipient, amount); // Minting.
        console.log("Receipt", transactionHash);
        await this._monitorStateStore.store("nineChronicles", { blockHash, txId
});
        await this._slackWebClient.chat.postMessage({
            channel: "#nine-chronicles-bridge-bot",
            ...new WrappedEvent(this._explorerUrl, this._etherscanUrl, sender,
recipient, amountString, txId, transactionHash).render()
        });
    }
}
```

NCGTransferredEventObserver::notify()

Bridge validates the Ethereum recipient address using the `web3.util.isAddress()` function.

```
// File:
https://github.com/ChainSafe/web3.js/blob/436e77a8eaa061fbaa183a9f73ca590c2e1d7
697/packages/web3-utils/src/utils.js
// Lines: 87 ~ 98

var isAddress = function (address) {
   // check if it has the basic requirements of an address
   if (!/^(0x)?[0-9a-f]{40}$/i.test(address)) {
       return false;
       // If it's ALL lowercase or ALL upppercase
   } else if (/^(0x|0X)?[0-9a-f]{40}$/.test(address) ||
/^(0x|0X)?[0-9A-F]{40}$/.test(address)) {
       return true;
       // Otherwise check each case
   } else {
       return checkAddressChecksum(address);
   }
};
```

web3.util.isAddress()

The point here is that the `web3.util.isAddress()` function returns true when the recipient address satisfies the regex condition (all letters are lowercase or uppercase). Here are some examples of addresses for which `web3.util.isAddress()` function returns true.

- */^(0x|0X)?[0-9a-f]{40}$/*
  - 0xaaaaaaaaaabbbbbbbbbbccccccccccddddddddddd
  - 0Xaaaaaaaaaabbbbbbbbbbccccccccccddddddddddd
  - aaaaaaaaaabbbbbbbbbbccccccccccddddddddddd
- */^(0x|0X)?[0-9A-F]{40}$/*

- 0xAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDD
- 0XAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDD
- AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDD

In other words, **(1) "0x{bytes20}" (2) "0X{bytes20}" (3) "{bytes20}"** addresses which satisfy the above regex condition returns true.

```
// File: bridge/src/wrapped-ncg-minter.ts
// Lines: 22 ~ 28

async mint(address: string, amount: Decimal): Promise < TransactionReceipt > {
    //NOTICE: This can be a problem if the number of digits in amount exceeds
9e+14.
    //more detail: https://mikemcl.github.io/decimal.js/#toExpPos
    Decimal.set({ toExpPos: 900000000000000 });
    console.log(`Minting ${amount.toString()}
${this._contractDescription.address} to ${address}`);
    return this._contract.methods.mint(address,
this._web3.utils.toBN(amount.toString())).send({ from: this._minterAddress });
// [!]
}
```

WrappedNCGMinter::mint()

```
//
https://github.com/ethers-io/ethers.js/blob/ba404ffb0b77f95f37172a5e52da0f7949a
7f406/packages/address/lib/index.js#L68-L100
export function getAddress(address: string): string {
   let result = null;
   // ...

   if (address.match(/^(0x)?[0-9a-fA-F]{40}$/)) { // [1]
       // ...

   } else if (address.match(/^XE[0-9]{2}[0-9A-Za-z]{30,31}$/)) {
       // ...
   } else {
       logger.throwArgumentError("invalid address", "address", address);
   }

   return result;
}
```

@ethersproject/address.getAddress()

However, unlike the `web3.util.isAddress()` function, the *ethers* `getAddress` function that is invoked within the `wrappedNcgTransfer.mint()` function approves addresses in **"0x{bytes20}"** and **"{bytes20}"** format. Therefore, if a recipient address starts with letters '0X' , the address will not satisfy the above two conditions and the `getAddress` function will throw the *invalid address* error

```
//
https://github.com/ethers-io/ethers.js/blob/v5.0.7/packages/bignumber/src.ts/bignumber.ts#L201-L214
static from(value: any): BigNumber {
   if (value instanceof BigNumber) { return value; }

   if (typeof(value) === "string") {
       if (value.match(/^-?0x[0-9a-f]+$/i)) { // [1]
           return new BigNumber(_constructorGuard, toHex(value));
       }

       if (value.match(/^-?[0-9]+$/)) { // [2]
           return new BigNumber(_constructorGuard, toHex(new BN(value)));
       }

       return logger.throwArgumentError("invalid BigNumber string", "value", value);
   }
```

@ethersproject/bignumber.from()

Furthermore, the ethers library recognizes an address as a string and casts it to a BigNumber in the `bignumber.from()` function. In this function's code, if the recipient address is neither a hexadecimal that starts with '0x' nor a only numeric, `bignumber.from()` function throws the *invalid BigNumber string* error

Eventually, an attacker can terminate the NineChroniclesTransferredEventmonitor by passing a recipient address in the form of **"0X{bytes20}"** or **"{bytes20}"** so that `web3.util.isAddress()` function returns true.

[Steps]

1. Send mint request that contains invaild recipient address
2. Check the terminal log on bridge process

[PoC]

The below PoC terminates the NineChroniclesTransferredEventMonitor. The recipient address of the first request starts with '0X', and the recipient address of the second request does not have '0x' in the first two characters. Both of them cause an exception.

```
mutation{
 transfer(recipient: "0xD03523408F26F80f8C5aC0eB15485A7FA82f2C66", amount: "1",
txNonce:104 , memo: "0XC1912FEE45D61C87CC5EA59DAE31190FFFFF232D")
}

// ...

mutation{
 transfer(recipient: "0xD03523408F26F80f8C5aC0eB15485A7FA82f2C66", amount: "1",
txNonce:104 , memo: "0Xc1912fee45d61c87cc5ea59dae31190fffff232d")
}
```

Mint request with '0X' character in the recipient address

```
mutation{
   transfer(recipient: "0xD03523408F26F80f8C5aC0eB15485A7FA82f2C66", amount:
"1", txNonce:104 , memo: "C1912FEE45D61C87CC5EA59DAE31190FFFFF232D")
}
```

Mint request without '0x' character in the recipient address

[Result]

```
[NineChroniclesTransferredEventMonitor] Execute triggerred block # 2160580
Minting 1000000000000000000 0xc16b1dff0ab1cdd6caa9b040059e5642a865a139 to
0XC1912FEE45D61C87CC5EA59DAE31190FFFFF232D
(node:59056) UnhandledPromiseRejectionWarning: Error: invalid address
(argument="address", value="0XC1912FEE45D61C87CC5EA59DAE31190FFFFF232D",
code=INVALID_ARGUMENT, version=address/5.4.0)
```

Result of mint request with '0X' character in recipient address
9c-audit transaction log

```
[NineChroniclesTransferredEventMonitor] Execute triggerred block # 2160619
Minting 1000000000000000000 0xc16b1dff0ab1cdd6caa9b040059e5642a865a139 to
c1912fee45d61c87cc5ea59dae31190fffff232f
(node:71332) UnhandledPromiseRejectionWarning: Error: invalid BigNumber string
(argument="value", value="c1912fee45d61c87cc5ea59dae31190fffff232f",
code=INVALID_ARGUMENT, version=bignumber/5.4.1)
```

Result of mint request without '0x' character in recipient address
9c-audit transaction log

[Reference]

- web3.utils — web3.js 1.0.0 documentation

Risk

- An attacker can terminate NineChroniclesTransferredEventMonitor. Thus, other user's minting events (NCG → WNCG) will be ignored until the bridge restarts.

Root cause and solution

[Root cause]

- Due to improper validation of recipient address, the bridge tries to mint WNCG tokens to an invalid recipient address.
- The bridge does not handle any exceptions which occur when sending transactions.

[Solution]

- Handling exceptions when sending transactions.
- Make sure the recipient address is properly validated.
    - Make sure the recipient address starts with "0x" and then validate with `web3.util.isAddress()` function.

Fix
Planetarium fixed the issue by validating the address starts with “0x” and then validate with `web3.util.isAddress()` function. ``` // File: bridge/src/observers/nine-chronicles.ts // Lines: 18 ~ 20 function isValidAddress(address: string): boolean {   return address.startsWith("0x") && isAddress(address) && address !== ZERO_ADDRESS; } ``` BRIDGE-004 fixed code

4. Recommendations

These are recommendations to consider for improving the security of NineChronicles bridge. These do not impose any immediate security impacts.

Recommendation ID	Title
Recommendations #1	**Remove unreachable codes**
Recommendations #2	**Handle Exceptions**
Recommendations #3	**Ensure that bridge runs exclusively**
Recommendations #4	**Incorrect block calculation**

4.1 Remove unreachable codes

We found some functions in the repository that appear to be unreachable. They should be deleted to increase the efficiency of code management and maintenance. Below are file names of unreachable code.

- bridge/src/headless-http-client.ts
- bridge/src/ncg-transfer.ts

4.2 Handle Exceptions

If the bridge service is terminated by some exceptions, it may cause financial damage. Identify the functions which can raise exceptions, and handle the exceptions to reduce downtime of service and to ensure availability. For example, slackWebClient can raise exceptions when the slack server has a network issue. It must be handled with the `Promise.error()` function.

4.3 Ensure that bridge runs exclusively

Create a system so that only one bridge runs at a time. In the current version, multiple bridges can run at the same time. So, if multiple bridges are run by mistake, the bridges may handle mint or burn events several times, which lead to financial damage. We suggest using an external server that runs Redis service to save the information about the bridge that is currently running, and check whether there is any information about the currently-running bridge, before running another bridge.

4.4 Incorrect block calculation

```
// File: bridge/src/monitors/nine-chronicles-transferred-event-monitor.ts
// Lines: 23

// Test code below.
const AUTHORIZED_BLOCK_INTERVAL = 50;
var blockIndex = 50;
const authorizedBlockIndex = Math.floor((blockIndex + AUTHORIZED_BLOCK_INTERVAL)
/ AUTHORIZED_BLOCK_INTERVAL) * AUTHORIZED_BLOCK_INTERVAL;
console.log('result : ' + authorizedBlockIndex)

// result : 100
```

Edge case at calculation of authorizedBlockIndex.

When calculating the remaining blocks in the `processRemains` function, the calculation of `authorizedBlockIndex` is wrong. The authorized block index is 50 when bridge processes the 50th block, but `processRemains` function returns authorized block index as 100. Every 50th block has the possibility to be affected by this issue. This may cause critical issues when the block is reorganized.
(This issue was found in the target commit version of our scope, but it has been fixed with an added commit during the audit.)

5. Conclusion

This project report was prepared based on the code security audit conducted by two professional researchers of Theori from August 9th to August 23th, 2021 for a total of two and a half weeks. As a result of the audit, four issues were identified. Among the issues discovered, some issues could lead to critical issues such as forced termination of the bridge (BRIDGE-001, BRIDGE-003, BRIDGE-004) and depletion of the contract's Ethereum balance (BRIDGE-002).
To improve the security of the service, we recommend prioritizing the patch of findings according to severity level and then proceeding with the patch.