Protocol Audit Report

Version 1.0

mgnfy-view

July 17, 2024

Protocol Audit Report

mgnfy-view

July 17, 2024

Prepared by: mgnfy-view

Table of Contents

    * [I-01] Use `safeTransfer()` instead of `transfer()` function in `GNATI_BRIDGE::transfer()` function
    * [I-02] Avoid using magic numbers in `GNATI_BRIDGE::swapToBridge()`
  - Gas
    * [G-01] Use external functions instead of public functions to save gas
    * [G-02] Use custom errors instead of `require` statements in `GNATI_BRIDGE::swapToBridge()`

Disclaimer

mgnfy-view makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

The findings described in this document correspond the following commit hash:

```
c1ef69aeae2efe701c701b70f63ceb2ab86e1c8a
```

Scope

```
./contracts/
#-- Proxy.sol
```

Protocol Summary

The Nati bridge serves as a proxy to the Illuminati token contract on Ethereum Mainnet. It brings down the supply of $NATI from 33 trillion to 3.3 billion, and allows the tokens to be bridged to the Verus chain via the Verus Ethereum bridge. The proxy locks up users $NATI tokens on Ethereum Mainnet, scales it down by a factor of 1e4, mints tokens to the Verus Ethereum bridge and creates an ERC20 export to an r-address on Verus. While going from Verus to Ethereum, the proxy tokens are burned, scaling takes place and the user's locked $NATI is freed up and available for circulation.

Executive Summary

The Nati bridge correctly defines the idea to bring down the supply and bridge tokens to the Verus chain, however, it fails to handle rounding issues that occur during scaling.

Issues Found

Severity	Number of issues found
High	2
Medium	0
Low	2
Info	2
Gas	2
Total	8

High

[H-01] Bridging $NATI token amounts lesser than 1e14 will cause tokens to be lost and stuck in GNATI_BRIDGE

Summary

The `GNATI_BRIDGE::swapToBridge()` function is used to bridge $NATI tokens from the Ethereum mainnet to the Verus chain. However, it lacks checks to see if the calculated amount to bridge is equal to zero or not, which can lead to token loss for the user.

Vulnerability Details

The `GNATI_BRIDGE` is an intermediate proxy which reduces the total market cap of the $NATI tokens from 33 trillion (33e30, with 18 decimals) to 3.3 billion (33e26 with 18 decimals). Thus, any tokens that need to be bridged are first scaled down by 1e4. All coins on Verus use 8 decimals, so further scaling down from 18 decimals to 8 decimals takes place by dividing by 1e10. This can be seen from the code segment below,

```
1      function swapToBridge(
2          uint256 _amountToSwap,
3          address addressTo,
4          uint8 addressType,
5          address bridgeAddress,
6          address destinationCurrency,
7          address feecurrencyid
8      ) public payable {
9          // more code here
10
11         // amount to mint of proxy token that only the bridge accepts
12 @>      uint256 amountToMint = _amountToSwap / multiplier;
13
14         // more code here
15
16 @>      uint64 verusAmount = uint64(amountToMint / SATS_TO_WEI_STD); //
       from 18 decimals to 8
17
18         VerusBridge(bridgeAddress).sendTransfer{value: msg.value}(
19             buildReserveTransfer(verusAmount, addressTo, addressType,
                   destinationCurrency, feecurrencyid)
20         );
21     }
```

However, while bridging the tokens there is no check to see if the `verusAmount` is 0 or not. If the amount to bridge is less than 1e14 (and has 4 trailing zeros to bypass the `_amountToSwap % multiplier == 0` check), then the tokens will be locked up in the proxy contract forever, unable to be recovered. This is because the Verus Ethereum Bridge does not revert for 0 amount transfers.

Impact

If any amount smaller than 1e14 is bridged, tokens will be stuck in the `GNATI_BRIDGE` proxy forever.

Proof of Concept

The following fork test (Ethereum Sepolia, with Foundry) demonstrates how tokens can be lost.

Code

```
1  contract ProxyTest is Test {
2      struct ProxyConfig {
3          address linkedERC20;
```

```
4           address thisTokeniAddress;
5           address destinationCurrency;
6           address vETHiAddress;
7       }
8
9       ProxyConfig public s_config;
10      address public s_proxy;
11      address public s_verusBridge;
12      address public s_natiWhale;
13      address public s_receiver;
14
15      uint256 private s_sepoliaFork;
16      uint256 private s_rollTo;
17
18      function setUp() public {
19          s_config = ProxyConfig({
20              linkedERC20: 0xA23DFcF889e9544fa8d7DC8e3774b979F4Ca5bA1,
21              thisTokeniAddress: 0
                    xE73b92E469c49A4651AeFf204ecFE920a78022DB,
22              destinationCurrency: 0
                    xffEce948b8A38bBcC813411D2597f7f8485a0689,
23              vETHiAddress: 0x67460C2f56774eD27EeB8685f29f6CEC0B090B00
24          });
25
26          s_proxy = 0xc363C4eda3bEF13984F1E170a6840E0d8Bc777aA;
27          s_verusBridge = 0xCaA98A4eC79dAC8A06Cb3BfDcF5351b6576d939f;
28          s_natiWhale = 0x4a7C219FB111982C81Ce777F2edBD692663e0A34;
29
30          s_rollTo = 6304433;
31          s_receiver = 0x55F51a22c79018A00CEd41e758560F5dF7d4d35d;
32
33          string memory SEPOLIA_RPC_URL = vm.envString("SEPOLIA_RPC_URL")
                ;
34          s_sepoliaFork = vm.createSelectFork(SEPOLIA_RPC_URL);
35          vm.rollFork(s_rollTo);
36      }
37
38      function testBridgingSmallAmountsLeadsToTokenLoss() public {
39          uint256 dealAmount = 0.003 ether;
40          deal(s_natiWhale, dealAmount);
41
42          uint8 addresstype = 2;
43          uint256 amountToBridge = 5e13;
44
45          uint256 bridgeProxyTokenBalanceBefore = IERC20(s_proxy).
                balanceOf(s_verusBridge);
46          uint256 whaleNatiBalanceBefore = IERC20(s_config.linkedERC20).
                balanceOf(s_natiWhale);
47          uint256 proxyNatiTokenBalanceBefore = IERC20(s_config.
                linkedERC20).balanceOf(s_proxy);
48
```

```
49          vm.startPrank(s_natiWhale);
50          IERC20(s_config.linkedERC20).approve(s_proxy, amountToBridge);
51          IProxy(s_proxy).swapToBridge{value: dealAmount}(
52              amountToBridge, s_receiver, addresstype, s_verusBridge,
                    s_config.destinationCurrency, s_config.vETHiAddress
53          );
54
55          uint256 bridgeProxyTokenBalanceAfter = IERC20(s_proxy).
                balanceOf(s_verusBridge);
56          uint256 whaleNatiBalanceAfter = IERC20(s_config.linkedERC20).
                balanceOf(s_natiWhale);
57          uint256 proxyNatiTokenBalanceAfter = IERC20(s_config.
                linkedERC20).balanceOf(s_proxy);
58
59          assertEq(bridgeProxyTokenBalanceBefore,
                bridgeProxyTokenBalanceAfter);
60          assertEq(whaleNatiBalanceBefore - whaleNatiBalanceAfter,
                amountToBridge);
61
62          console.log("The proxy tokens held by the verus bridge contract
                 before: ", bridgeProxyTokenBalanceBefore);
63          console.log("The proxy tokens held by the verus bridge contract
                 after: ", bridgeProxyTokenBalanceAfter);
64          console.log("The NATI tokens held by user before bridging: ",
                whaleNatiBalanceBefore);
65          console.log("The NATI tokens held by user after bridging: ",
                whaleNatiBalanceAfter);
66          console.log("Tokens stuck in the proxy: ",
                proxyNatiTokenBalanceAfter - proxyNatiTokenBalanceBefore);
67      }
68  }
```

The test passes with the following logs,

```
1  Ran 1 test for test/Proxy.t.sol:ProxyTest
2  [PASS] testBridgingSmallAmountsLeadsToTokenLoss() (gas: 617452)
3  Logs:
4    The proxy tokens held by the verus bridge contract before:
       32000500000000000000
5    The proxy tokens held by the verus bridge contract after:
       32000500000000000000
6    The NATI tokens held by user before bridging:
       99999669996000000000000000000
7    The NATI tokens held by user after bridging:
       99999669995999950000000000000
8    Tokens stuck in the proxy:
       50000000000000
9
10 Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.67s
     (6.11ms CPU time)
```

Recommended Mitigation

Add a check in the GNATI_BRIDGE contract to see if the amount to bridge is greater than zero or not,

```
    function swapToBridge(
        uint256 _amountToSwap,
        address addressTo,
        uint8 addressType,
        address bridgeAddress,
        address destinationCurrency,
        address feecurrencyid
    ) public payable {

        // more code here

        // amount to mint of proxy token that only the bridge accepts
        uint256 amountToMint = _amountToSwap / multiplier;

        // more code here

        uint64 verusAmount = uint64(amountToMint / SATS_TO_WEI_STD); //
            from 18 decimals to 8

+       require(amountToMint > 0 && verusAmount > 0, "Insufficient
    amount to bridge");

        VerusBridge(bridgeAddress).sendTransfer{value: msg.value}(
           buildReserveTransfer(verusAmount, addressTo, addressType,
           destinationCurrency, feecurrencyid));
    }
```

[H-02] $NATI tokens can be lost and stuck in the contract due to rounding down while dividing

Summary

The GNATI_BRIDGE::swapToBridge() function has an initial check to see if the amount to bridge does not undergo truncation while dividing by 1e4, which is to scale down the amount to go from a 33 trillion to a 3.3 billion market cap. However, truncation due to rounding down is also possible while adjusting the amount to 8 decimals (for Verus) by dividing by 1e10. This can lead to some tokens being stuck in GNATI_BRIDGE.

Vulnerability Details

Consider the following code segment from GNATI_BRIDGE::swapToBridge(),

```
    function swapToBridge(
        uint256 _amountToSwap,
```

```
3           address addressTo,
4           uint8 addressType,
5           address bridgeAddress,
6           address destinationCurrency,
7           address feecurrencyid
8       ) public payable {
9           // more code here
10
11          // make sure amount being sent is a multiple of the multiplier
                to stop wei being lost in truncation
12  @>      require(_amountToSwap % multiplier == 0, "not divisable by
        1000000");
13
14          // more code here
15
16  @>      uint64 verusAmount = uint64(amountToMint / SATS_TO_WEI_STD); //
         from 18 decimals to 8
17
18          VerusBridge(bridgeAddress).sendTransfer{value: msg.value}(
19              buildReserveTransfer(verusAmount, addressTo, addressType,
                    destinationCurrency, feecurrencyid)
20          );
21      }
```

The first check can be bypassed with any amount with 4 trailing zeros (multiplier value is 1e4). However, lack of such a check for the 14 trailing zeros implies that any amount upto 1e14 can be lost and stuck in `GNATI_BRIDGE` while bridging.

Impact

While bridging tokens, it is possible that any amount upto 1e14 can be lost.

Proof of Concept

Consider a user who wants to bridge 1000050000000000000 $NATI tokens to the Verus chain.

1. The `GNATI_BRIDGE` transfers 1000050000000000000 $NATI from the user to itself.
2. The amount is first scaled down by 1e4, leaving us with 100005000000000.
3. Now, to convert the amount to 8 decimals, we divide by 1e10. This leaves us with 10000 tokens.
4. 1e4 tokens will be brideged over to Verus. However, while bridging back to Ethereum mainnet, the user will only be able to claim 1000000000000000000 $NATI (scaling up by 1e14).
5. The amount of tokens lost by user while bridging is 1000050000000000000 - 1000000000000000000 = 5e13.

Here's a fork test (Ethereum Sepolia, with Foundry),

Code

```
1  contract ProxyTest is Test {
```

```
2       struct ProxyConfig {
3           address linkedERC20;
4           address thisTokeniAddress;
5           address destinationCurrency;
6           address vETHiAddress;
7       }
8
9       ProxyConfig public s_config;
10      address public s_proxy;
11      address public s_verusBridge;
12      address public s_natiWhale;
13      address public s_receiver;
14
15      uint256 private s_sepoliaFork;
16      uint256 private s_rollTo;
17
18      function setUp() public {
19          s_config = ProxyConfig({
20              linkedERC20: 0xA23DFcF889e9544fa8d7DC8e3774b979F4Ca5bA1,
21              thisTokeniAddress: 0
                    xE73b92E469c49A4651AeFf204ecFE920a78022DB,
22              destinationCurrency: 0
                    xffEce948b8A38bBcC813411D2597f7f8485a0689,
23              vETHiAddress: 0x67460C2f56774eD27EeB8685f29f6CEC0B090B00
24          });
25
26          s_proxy = 0xc363C4eda3bEF13984F1E170a6840E0d8Bc777aA;
27          s_verusBridge = 0xCaA98A4eC79dAC8A06Cb3BfDcF5351b6576d939f;
28          s_natiWhale = 0x4a7C219FB111982C81Ce777F2edBD692663e0A34;
29
30          s_rollTo = 6304433;
31          s_receiver = 0x55F51a22c79018A00CEd41e758560F5dF7d4d35d;
32
33          string memory SEPOLIA_RPC_URL = vm.envString("SEPOLIA_RPC_URL")
                ;
34          s_sepoliaFork = vm.createSelectFork(SEPOLIA_RPC_URL);
35          vm.rollFork(s_rollTo);
36      }
37
38      function testRoundingDownCausesTokensToBeStuck() public {
39          uint256 dealAmount = 0.003 ether;
40          deal(s_natiWhale, dealAmount);
41
42          uint8 addresstype = 2;
43          uint256 amountToBridge = 1000050000000000000;
44
45          uint256 bridgeProxyTokenBalanceBefore = IERC20(s_proxy).
                balanceOf(s_verusBridge);
46          uint256 proxyNatiTokenBalanceBefore = IERC20(s_config.
                linkedERC20).balanceOf(s_proxy);
47
```

```
48          vm.startPrank(s_natiWhale);
49          IERC20(s_config.linkedERC20).approve(s_proxy, amountToBridge);
50          IProxy(s_proxy).swapToBridge{value: dealAmount}(
51              amountToBridge, s_receiver, addresstype, s_verusBridge,
                    s_config.destinationCurrency, s_config.vETHiAddress
52          );
53
54          uint256 bridgeProxyTokenBalanceAfter = IERC20(s_proxy).
                balanceOf(s_verusBridge);
55          uint256 proxyNatiTokenBalanceAfter = IERC20(s_config.
                linkedERC20).balanceOf(s_proxy);
56
57          uint256 tokensLost = (proxyNatiTokenBalanceAfter -
                proxyNatiTokenBalanceBefore)
58              - (bridgeProxyTokenBalanceAfter -
                    bridgeProxyTokenBalanceBefore) * 1e4;
59          console.log("The amount of $NATI tokens stuck in the contract
                and cannot be recovered: ", tokensLost);
60      }
61  }
```

The test passes with the following logs,

```
1  Ran 1 test for test/Proxy.t.sol:ProxyTest
2  [PASS] testRoundingDownCausesTokensToBeStuck() (gas: 617367)
3  Logs:
4    The amount of $NATI tokens stuck in the contract and cannot be
       recovered:  50000000000000
5
6  Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.74s
     (5.09ms CPU time)
7
8  Ran 1 test suite in 6.74s (6.74s CPU time): 1 tests passed, 0 failed, 0
     skipped (1 total tests)
```

Recommended Mitigation

Make the following changes in `GNATI_BRIDGE::swapToBridge()` function,

```
1      function swapToBridge(
2          uint256 _amountToSwap,
3          address addressTo,
4          uint8 addressType,
5          address bridgeAddress,
6          address destinationCurrency,
7          address feecurrencyid
8      ) public payable {
9          // more code here
10
11         // make sure amount being sent is a multiple of the multiplier
               to stop wei being lost in truncation
```

```
12 -        require(_amountToSwap % multiplier == 0, "not divisable by
       1000000");
13 +        require(_amountToSwap % (multiplier * SATS_TO_WEI_STD) == 0, "
       Not divisible by 1e14");
14
15          // more code here
16      }
```

This will ensure that users will bridge an amount that never undergoes truncation due to division, first by 1e4, and then by 1e10.

Low

[L-01] Missing event emission in `GNATI_BRIDGE::transfer()` and `GNATI_BRIDGE::swapToBridge()` functions

Vulnerability Details

The `GNATI_BRIDGE` lacks event emission in two major functions `GNATI_BRIDGE::transfer()` and `GNATI_BRIDGE::swapToBridge()`. The former is used to bridge tokens from Verus to Ethereum, and the latter bridges tokens in the opposite direction.

Impact

Events are used by off-chain services to track on-chain activities. It is also useful for debugging purposes.

Recommended Mitigation

Make the following changes in `GNATI_BRIDGE`,

```
1  contract GNATI_BRIDGE is ERC20 {
2      // more code here
3
4  +   event Bridged(bool fromEthereumToVerus, uint256 amountFrom, uint256
       amountTo, address receiver);
5
6      // more code here
7
8      function transfer(address to, uint256 amount) public virtual
          override returns (bool) {
9          _burn(msg.sender, amount);
10
11         //send the scaled up amount back to the user on ETH
12         ERC20(linkedERC20).transfer(to, (amount * multiplier));
13
14 +       emit Bridged(false, amount, (amount * multiplier), to);
15
```

```
16          return true;
17      }
18
19      function swapToBridge(
20          uint256 _amountToSwap,
21          address addressTo,
22          uint8 addressType,
23          address bridgeAddress,
24          address destinationCurrency,
25          address feecurrencyid
26      ) public payable {
27
28          // more code here
29
30          uint64 verusAmount = uint64(amountToMint / SATS_TO_WEI_STD); //
                from 18 decimals to 8
31
32          VerusBridge(bridgeAddress).sendTransfer{value: msg.value}(
33              buildReserveTransfer(verusAmount, addressTo, addressType,
                    destinationCurrency, feecurrencyid)
34          );
35
36  +       emit Bridged(true, _amountToSwap, verusAmount, addressTo);
37      }
38
39      // more code here
40
41  }
```

[L-02] Allowing user supplied bridge address in GNATI_BRIDGE can cause anomalies, breaking protocol invariant

Summary

With an arbitrary bridge address supplied by users, they can get hold of proxy tokens without actually bridging them to Verus. Also, a lack of bridge address implies a lack of access control for `GNATI_BRIDGE::transfer()` function.

Vulnerability Details

The `GNATI_BRIDGE::swapToBridge()` function allows the user to supply an arbitrary bridge address. Users can pass a custom contract address they own which has a `sendTransfer()` function. This will allow them to transfer the proxy tokens to themselves without actually bridging $NATI tokens to Verus, breaking the core invariant – proxy tokens should only be held by the bridge, and a scaled down amount should be freed up on Verus.

Additionally, the `GNATI_BRIDGE::transfer()` function should be used exclusively by the Verus

Ethereum bridge to transfer a scaled up amount to a receiver once $NATI tokens are bridged from Verus to Ethereum. Users should not hold the proxy tokens directly, and thus should not be exposed to the transfer function.

Impact

Users can pass a custom contract address they own which has a `sendTransfer()` function, and get hold of proxy tokens without actually bridging to Verus. This breaks two protocol invariants:

1. Proxy tokens should only be held by the Verus Ethereum bridge.
2. If proxy tokens are minted on Ethereum, then a scaled down amount should be freed up and in circulation on Verus.

Additionally, since the `GNATI_BRIDGE::transfer()` function lacks access control, an attacker can drain the bridge if they can find a way to mint more tokens.

Recommended Mitigation

Do not allow users to supply an arbitrary bridge address. Set a `verusBridge` address variable in `GNATI_BRIDGE` and make direct calls to it. Since the bridge address is susceptible to change in the future (confirmed by the Verus community), set a trusted admin for the `GNATI_BRIDGE` contract who can change the `verusBridge` address as required.

Also, add access control to the `GNATI_BRIDGE::transfer()` function so that it is only callable by the Verus Ethereum bridge.

Informational

[I-01] Use `safeTransfer()` instead of `transfer()` function in `GNATI_BRIDGE::transfer()` function

The `GNATI_BRIDGE::transfer()` function sends the scaled up $NATI tokens to the receiver on bridging from Verus to Ethereum using the `IERC20::transfer()` function. It is recommended that `safeTransfer()` from Openzeppelin's `SafeERC20` library is used to ensure token transfers do not fail silently.

[I-02] Avoid using magic numbers in `GNATI_BRIDGE::swapToBridge()`

Magic number is used for the ETH amount to be supplied for bridging tokens, as can be seen below,

```
1     function swapToBridge(
2         uint256 _amountToSwap,
3         address addressTo,
```

```
4           uint8 addressType,
5           address bridgeAddress,
6           address destinationCurrency,
7           address feecurrencyid
8       ) public payable {
9   @>      require(msg.value == 0.003 ether, "0.003 ETH required");
10
11          // more code here
12
13      }
```

Avoid doing this. Always use constants. Make the following changes,

```
1  contract GNATI_BRIDGE is ERC20 {
2
3      // more code here
4
5  +   uint256 public constant fee = 0.003 ether;
6
7      function swapToBridge(
8          uint256 _amountToSwap,
9          address addressTo,
10         uint8 addressType,
11         address bridgeAddress,
12         address destinationCurrency,
13         address feecurrencyid
14     ) public payable {
15 -       require(msg.value == 0.003 ether, "0.003 ETH required");
16 +       require(msg.value == fee, "0.003 ETH required");
17
18         // more code here
19
20     }
21 }
```

Gas

[G-01] Use external functions instead of public functions to save gas

The `GNATI_BRIDGE::transfer()` and `GNATI_BRIDGE::swapToBridge()` functions should be marked with external visibility to save gas.

[G-02] Use custom errors instead of `require` statements in `GNATI_BRIDGE::swapToBridge()`

Custom errors use a lot less gas than require statements. Make the following changes,

```
contract GNATI_BRIDGE is ERC20 {

    // more code here

+   event InsufficientFeesSupplied();
+   event IncorrectInputAmount();

    function swapToBridge(
        uint256 _amountToSwap,
        address addressTo,
        uint8 addressType,
        address bridgeAddress,
        address destinationCurrency,
        address feecurrencyid
    ) public payable {
-       require(msg.value == 0.003 ether, "0.003 ETH required");
+       if (msg.value != 0.003 ether) revert InsufficientFeesSupplied()
    ;

        // make sure amount being sent is a multiple of the multiplier
           to stop wei being lost in truncation
-       require(_amountToSwap % multiplier == 0, "not divisable by
    1000000");
+       if (_amountToSwap % multiplier != 0) revert
    IncorrectInputAmount();

        // more code here

    }
}
```