SPEARBIT

Tezos Etherlink Security Review

Auditors

Lukasz Glen, Lead Security Researcher

Wei Tang, Lead Security Researcher

Report prepared by: Lucas Goiriz

October 29, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Etherlink is an EVM-compatible, non-custodial Layer 2 blockchain powered by Tezos Smart Rollup technology. It enables seamless integration with existing Ethereum tools, including wallets and indexers, and facilitates asset transfers to and from other EVM-compatible chains.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of tezos according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 6 weeks days in total, the Etherlink Team engaged with Spearbit to review the a new version of the Etherlink Kernel. In this period of time a total of **13** issues were found.

Summary

Project Name	Etherlink FA Bridge
Repository	https://gitlab.com/tezos/tezos
Commit	b4f3a1...a6ed
Type of Project	Infrastructure, L2
Audit Timeline	Sep 9 to Oct 16

All follow up changes addressing the issues listed in the table below are included in the commit fbf731e19f51a395d67b1a303d1fb3332db4d58e.

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	7	5	2
Gas Optimizations	0	0	0
Informational	4	1	3
Total	**13**	**8**	**5**

5 Findings

5.1 Medium Risk

5.1.1 Inconsistant stack in `execute_fa_withdrawal`

Severity: Medium Risk

Context: Commit 7e9a0a01

Description: In the new logic of `execute_fa_withdrawal`, on L207, a new execution stack is pushed by `handler.begin_inter_transaction`. It then enters `inner_execute_withdrawal` with Rust's `?` syntax. Later lines we also calls `inner_execute_proxy` also with Rust's `?` syntax.

If an error happens in either function, the function returns directly without any further clean up code, meaning that the `handler.end_inter_transaction` will not be called. In this case, the "transaction stack" will become inconsistent. **At this stage, SputnikVM can guarantee nothing, not even correctly returning an `ExitFatal` error.**

We rate this as medium severity, as we believe that there's a particularly high risks that some forms of the bug may be exploitable during a system upgrade. From the author's experience with other blockchains of similar stacks, it's particularly likely, even in production environment, that some small amount of storage values may become unparsable due to incorrect migrations. At that moment, `read_u256_le_default` in the `TicketTable` will trigger this bug.

Recommendation: Do not use the `?` syntax but handle all errors manually and always remove the entered call stack.

Alternatively, wrap the logic between enter/exit call stack into a separate function and only use `?` syntax there.

Tezos: We agree and implemented the recomendation in MR 15358 (it can be easier to follow the changes in a merge request). It's a bit a spaghetti code because I wanted to have a non-empty withdrawal vector only when the auxiliary function returned `Ok(_)` (which was not enforced before).

It seems that the code is missing a record_cost before beginning the inter transaction, something like:

```diff
diff --git a/etherlink/kernel_evm/evm_execution/src/fa_bridge/mod.rs
↪  b/etherlink/kernel_evm/evm_execution/src/fa_bridge/mod.rs
index 3f0997ed15..466effebf9 100644
--- a/etherlink/kernel_evm/evm_execution/src/fa_bridge/mod.rs
+++ b/etherlink/kernel_evm/evm_execution/src/fa_bridge/mod.rs
@@ -238,6 +238,12 @@ pub fn execute_fa_withdrawal<Host: Runtime>(
     if handler.can_begin_inter_transaction_call_stack() {
         // Create a new transaction layer with 63/64 of the remaining gas.
         let gas_limit = handler.nested_call_gas_limit(None);
+
+
+        if let Err(err) = handler.record_cost(gas_limit.unwrap_or_default()) {
+            todo!()
+        }
+
         handler.begin_inter_transaction(false, gas_limit)?;
```

It has solved the failing tests.

(of course the todo!() needs to be an OutOfGas outcome).

Spearbit: Fix confirmed.

5.1.2 Call of FA bridge proxy contract not respecting the depth limit

Severity: Medium Risk

Context: `handler.execute_call`

Description: After `inner_execute_proxy`, `handler.execute_call` takes over and starts to execute the proxy contract. It notes the call stack depth but does not enforce it.

Recommendation: Properly create a new "transaction data" layer and enforce the call stack depth limit.

Tezos: See MR 15358 where a check was added for call stack before creating a new transaction data layer.

Spearbit: Fixed.

5.2 Low Risk

5.2.1 Tokens on the withdraw precompile address

Severity: Low Risk

Context: withdrawal.rs#L95

Description: When an account calls the withdraw precompile, tokens are transferred and added to the withdraw precompile address. In effect the withdraw precompile's balance declares the total of all withdrawals in the history.

Recommendation: The precompile should remove the received tokens from the balance.

Tezos: See the fix in MR 15281.

Spearbit: Confirmed.

5.2.2 LinkedList: removing the middle element from 3 element list

Severity: Low Risk

Context: Delivered LinkedList patch `8c3d50e74dbc44ec6004666a21b3ee36`

Description: When removing the middle element from 3 element list, the back element of list is set incorrectly.

Recommendation: Handle the case where both the front and the back must be updated after removing an intermediate element.

Tezos: Solved in MR 15206.

Spearbit: Confirmed.

5.2.3 The proxy may return a magic value instead of simply success

Severity: Low Risk

Context: fa_bridge/mod.rs#L316

Description: When handling FA deposit and FA withdrawal, a proxy contract is called. The call must succeed in order to make sure that the target contract understands the operation and tokens are handled properly.

Some enhancement is to require the call not only to succeed but also return a magic value - a specified constant 4 bytes. That increases sureness that a called code is intended to handle deposits or withdrawals. Motivation and inspiration is `ERC721TokenReceiver`.

An example of contract that may successfully handle calls having unexpected semantics is WETH9. WETH9 contract has the default call handler that performs ETH deposit.

Recommendation: The function `deposit()` and `withdraw()` of the proxy may return specified constant 4 bytes and the caller should check the returned value. Note that such change is not backward compatible.

Tezos: This unfortunately hasn't been prioritized, but I agree that it makes more sense. In order to have a future breaking change we could do a `deposit_bis()` and `withdraw_bis()` (with better names of course) that requires these magic bytes.

Spearbit: Acknowledged.

5.2.4 A silent success on reentrancy to a circular withdrawal call

Severity: Low Risk

Context: MR 14545

Description: In order to prevent circular calls (reentrancy), the entry `apply_fa_deposit` has got removed withdrawal precompiles from the set of precompiles. The point is that withdrawal precompile addresses have no bytecode, as contract bytecode. A call to no bytecode address always succeeds, so a call to a withdrawal address always succeed. For the record. It is not always the case that reentrancy is a malicious action of proxy. Making an external call may be a part of design. For instance, the standard ERC-777 is an extension of ERC-20 and it calls a contract receiver upon a token transfer.

Recommendation: Instead of a silent success, revert. Instead of removing a withdrawal precompiles from the set of precompiles, provide an implementation that always reverts.

Tezos: MR 15360 should fix the issue.

Spearbit: Fixed,. Just two comments:

- The flag `enable_fa_withdrawals` is not checked when binding `revert_precompile` to `FA_BRIDGE_PRECOMPILE_ADDRESS`, this is still fine since the revert precompile is used only when `enable_fa_withdrawals` is set.
- No ticks are registered, this is still fine since a call to the precompile generates a gas cost.

5.2.5 Checks Effects Interactions pattern in `execute_fa_deposit`

Severity: Low Risk

Context: fa_bridge/mod.rs#L119

Description: The flow of the function `execute_fa_deposit` is: call the proxy, update the ticket table, add EVM log. The safety practice is to have contract calls in the end of operations sequence.

The point to discuss is: if the deposit and FA deposit functions and withdraw and FA withdraw precompiles would follow the Checks Effects Interactions pattern, the system would be protected from reentrancy attacks. In particular, we should consider what resources are to be protected. It seems that the resources are: the ticket table, tez balances, EVM logs and outbox messages. Note that FA deposit and FA withdrawal have only one call to the proxy what makes things even more clear.

Note that the issue "Potential incorrect order of outbox messages" also refers to the Checks Effects Interactions pattern as adding a message to the outbox is Effects. And this issue should be fixed in order to provide seal reentrancy protection.

Recommendation: Follow the Checks Effects Interactions pattern, the flow should be: add EMV log, update the ticket table, call the proxy. Even if Reentrancy Guard is introduced, it is recommended to change the order.

Tezos: Acknowledged.

Spearbit: Acknowledged.

5.2.6 Shared gasometer in FA bridge proxy contract invocation

Severity: Low Risk

Context: `fa_bridge/mod.rs/inner_execute_proxy`

Description: When the FA bridge executes the proxy contract, it calls `handler.execute_call` directly. This call does not create a new "transaction data layer". The gasometer is shared between the `fa_bridge` precompile and the proxy contract.

The SputnikVM engine does not guarantee reuse of gasometer to be valid, especially after an OOG.

Recommendation: Create a new "transaction data" layer when invoking the proxy contract.

Tezos: In MR 15358 we created a new inter transaction that takes 63/64th of the gas remaining from the last transaction layer.

Spearbit: The same logic needs to be applied to `inner_execute_deposit`.

Tezos: `inner_execute_deposit` is wrapped in an `begin_initial_transaction` and `end_initial_transaction`, wouldn't it be enough?

Spearbit: In the commit, the new transaction layer spans on both `inner_execute_withdrawal` and `inner_execute_proxy`. Only the latter should be. Nonetheless, it is fine. It does not provide any danger. It just looks unexpected.

Note that the new transaction layer spans on the event log in `inner_execute_withdrawal` and the external call in `inner_execute_proxy`. But the point was that the proxy was called directly without a subtransaction as it would be in case of a call within a smart contract. Here it takes a bit more. But it is not significant.

5.2.7 Use of ticket table not respecting EIP-161 empty account rule

Severity: Low Risk

Context: `fa_bridge/ticket_table.rs`

Description: The FA bridge has the concept of a ticket table. The ticket table is stored, under a special path, of the account of `SYSTEM_ACCOUNT_ADDRESS`. After EIP-161, an account is considered empty (and may be removed by the EVM engine) if an account has empty code, empty nonce, and empty balance. The empty check rule does not take storage values or custom storage into account.

The EIP-161 rule states that an account, after being touched, shall be removed if it becomes empty. The author has not specifically checked whether this rule applies to the `SYSTEM_ACCOUNT_ADDRESS` when being invoked as the `fa_bridge` precompile, due to this being a rare case, explained below. However, we want to note that if the EVM engine is EIP-161 conforming (which according to previous tests, it was), there is a possibility that it will invoke the removal of the account.

On the other hand, during FA deposit, the execution is done as if it's a transaction from `SYSTEM_ACCOUNT_ADDRESS`, thus it will increase the nonce, eliminating the possibility that it is considered an empty account.

Recommendation: Manually increase the nonce of `SYSTEM_ACCOUNT_ADDRESS` during system upgrade, before any other transaction is allowed.

Tezos: Fixed in MR 15197.

Spearbit: Fix confirmed.

5.3 Informational

5.3.1 FA Deposit log should be first on the list

Severity: Informational

Context: fa_bridge/mod.rs#L149

Description: In the function `execute_fa_deposit` first is a call to a proxy and then the evm log Deposit is added. If a call to the proxy creates evm logs, then the Deposit log is the last on the list. It is preferred to have more informative logs first on the list. The Deposit log is the main log and it should be first.

Recommendation: Add the Deposit log first, then call the proxy. If the call fails, revert the log too and add a new Deposit log because a ticker owner is changed.

Tezos: Acknowledged.

Spearbit: Acknowledged.

5.3.2 Potential incorrect order of outbox messages

Severity: Informational

Context: fa_bridge/mod.rs#L212

Description: The flow of `execute_fa_withdrawal` goes as follows:

- Update the ticket table.
- Add the EVM log.
- Call the proxy.
- The withdrawal message is added to the outbox.

The last step is executed outside out the function when the message is returned to the calling function.

The point is the if we allow the reentrancy, a safe reentrancy, the order of EVM logs can be different than as the order of messages in the outbox. More general, if the call to the proxy adds a message to the outbox, it is before the withdrawal message in the outbox.

Note that this issue can be of medium severity if the reentrancy is allowed.

Recommendation: Add the withdrawal message to the outbox before calling the proxy.

Tezos: We would prefer not to follow the suggestion, it would break an abstraction. The order of outbox messages is not really relevant. What we need is to have a clear association between an L2 transaction and its withdrawals, and it's done using the `withdrawal_id`.

Spearbit: Acknowledged.

5.3.3 Incorrect logic when checking the `Transfer` object in `fa_bridge` precompile

Severity: Informational

Context: `precompiles/fa_bridge.rs#124`

Description: If a `Transfer` object does not exist, then it's not a normal `CALL`. So the logic here is incorrect. However, the severity of this is "informational", because another branch also checks for `DELEGATECALL` and `CALLCODE` above.

Recommendation: Change here to be `unwrap_or(true)`.

Tezos: Fixed in MR 14950.

Spearbit: Fix confirmed.

5.3.4 Backward compatibility of fixed gas limit in custom contract calls

Severity: Informational

Context: Description: During FA bridge deposit phrase, a fixed gas limit `FA_DEPOSIT_PROXY_GAS_LIMIT` is used to execute the proxy contract. We would like to note a possible backward compatibility issue that needs to be taken consideration every time the EVM gas configuration is changed.

In many situations, especially due to more expensive storage access costs when the state grows larger, the EVM gas configurations have often been raised. In this scenario, if a proxy contract was previously just below `FA_DEPOSIT_PROXY_GAS_LIMIT`, after the upgrade, it will suddenly become not-executable. The value `FA_DEPOSIT_PROXY_GAS_LIMIT` must also be changed at the same time.

Recommendation: An on-chain analysis of contracts is recommended to be conducted every time when the EVM gas configurations are changed. Especially if gas costs are raised, the value of `FA_DEPOSIT_PROXY_GAS_LIMIT` is also recommended to be raised to avoid backward compatibility issues.

Tezos: Acknowledged. It's worth to add comments around the `FA_DEPOSIT_PROXY_GAS_LIMIT` to explain this.

Spearbit: Acknowledged.