Important Security Checks Before Cancun Upgrade

As the Cancun upgrade approaches, Salus has prepared essential security measures for developers to review and familiarize ahead of the upgrade.

SINGAPORE, March 1, 2024 /EINPresswire.com/ -- Long story short: The Cancun upgrade is approaching, and this upgrade mainly includes six execution layer changes proposed by EIPs, namely EIP-1153, EIP-4788, EIP-4844, EIP-5656, EIP-6780, and EIP-7516. EIP-4844 is the star of this upgrade, aiming to enhance the scalability of Ethereum, reduce transaction costs and increase transaction speed for L2. The Cancun upgrade has been completed on the Ethereum Goerli, Sepolia, and Holesky testnets on January 17, January 30, and February 7, respectively, and is planned to be activated on the Ethereum mainnet on March 13. Before the upgrade, Salus has organized important security precautions for this upgrade for developers to check by themselves.

EIP Proposal Review

EIP-1153

EIP-1153 introduces transient storage opcodes, which are used to manipulate the state, behave almost the same as storage, but the transient storage will be discarded after each transaction ends. This means transient storage does not deserialize values from storage, nor does it serialize values to storage, and because it does not require disk access, transient storage is cheaper. With two new opcodes, TLOAD and TSTORE (where "T" stands for "transient"), smart contracts can access transient storage. This proposal aims to provide a dedicated and efficient solution for communication between multiple nested execution frames in Ethereum transaction execution.

EIP-4788

EIP-4788 aims to expose the hash tree root of the beacon chain blocks in the EVM, allowing these roots to be accessed within smart contracts. This would allow for trustless access to consensus layer state, supporting use cases such as staking pools, restaking structures, smart contract bridges, MEV mitigation, and more. This proposal stores these roots in a smart contract, and uses a ring buffer to limit storage consumption, ensuring that each execution block only needs constant space to represent this information.

EIP-4844

EIP-4844 introduces a new transaction format called "blob-carrying transactions", aimed at extending Ethereum's data availability in a simple, forward-compatible way. This proposal does this by introducing "blob-carrying transactions" that contain large amounts of data that cannot be accessed by EVM execution but can access its commitments. This format is fully compatible with the format used in full sharding in the future, providing temporary but significant relief for roll-out expansion.

EIP-5656

EIP-5656 introduces a new EVM instruction MCOPY for efficiently copying memory areas. This proposal aims to reduce the overhead of memory copy operations in the EVM by implementing data copy between memory directly through the MCOPY instruction. MCOPY allows source and destination addresses to overlap, considers backward compatibility in its design, and aims to enhance the execution efficiency of various scenarios including data structure construction, efficient access and copying of memory objects.

EIP-6780

EIP-6780 changes the functionality of the SELFDESTRUCT opcode. In this proposal, SELFDESTRUCT will only delete the account and transfer all Ether in the same transaction as contract creation, other than that, when SELFDESTRUCT is executed, the contract will not be deleted, but all Ether will be transferred to the specified target. This change is to adapt to future use of Verkle trees, aiming to simplify EVM implementation, reduce the complexity of state changes, while retaining some common uses of SELFDESTRUCT.

EIP-7516

EIP-7516 introduces a new EVM instruction BLOBBASEFEE to return the value of the blob base-fee of the current block it is executing in. This instruction is similar to the BASEFEE opcode in EIP-3198, the difference being that it returns the blob base fee defined by EIP-4844. This functionality allows contracts to programmatically consider the gas price of blob data, for example, allowing rollup contracts to calculate the cost of blob data use without trust, or to implement blob gas futures based on this to smooth the cost of blob data.

Officially disclosed security considerations.

EIP-1153

Smart contract developers should understand the lifecycle of transient storage variables before use. As transient storage is automatically cleared at the end of a transaction, smart contract developers might try to avoid clearing slots during a call to save Gas. However, this might prevent further interaction with the contract within the same transaction (for example, in the case of re-entrant locks) or lead to other errors, so smart contract developers should be cautious and only keep non-zero values when transient storage slots are reserved, intended for future calls within the same transaction. The behaviour of these opcodes is otherwise identical to SSTORE and SLOAD, so all common safety precautions apply, especially concerning re-entrancy risks.

Smart contract developers might also try to use transient storage as an alternative to memory mapping. They should be aware that transient storage is not discarded like memory when a call returns or reverts, and should favour memory in these use cases to prevent unexpected behaviour on re-entrancy within the same transaction. The cost of transient storage on memory is inevitably high, which should have deterred this usage pattern. Most uses of mappings in memory can be better implemented through sorted entry lists, and memory mappings are rarely needed in smart contracts (i.e., the author knows of no known use cases in production).

EIP-4844

This EIP increases the bandwidth requirement for each beacon block by up to about 0.75 MB. This is 40% larger than the theoretical maximum size of today's blocks (30M Gas / 16 Gas per calldata byte = 1.875M bytes), so it does not significantly increase the worst-case bandwidth. After the merge, block times are static rather than unpredictable Poisson distributions, providing a guaranteed time window for the propagation of large blocks.

Even with limited call data, the continual load of this EIP is much lower than alternatives that could reduce call data costs, as there is no need to store the blob as long as the execution load. This makes it possible to implement strategies where these blobs must be retained for at least some time. The chosen specific value is MIN_EPOCHS_FOR_BLOB_SIDECARS_REQUESTS epochs, about 18 days, much shorter in delay compared to the proposed (but not yet implemented) one-year rotation time for executing effective payload history.

EIP-5656

Clients should ensure their implementations do not use intermediate buffers (for example, the C stdlibmemmove function does not use an intermediate buffer), as this is a potential denial of service (DoS) vector. Most language built-in/standard library functions for moving bytes have the correct performance characteristics here.

Apart from this, the analysis of DoS and memory exhaustion attacks is the same as for other memory-touching opcodes, as memory expansion follows the same pricing rules.

EIP-6780

The following applications of SELFDESTRUCT will be broken, and it's no longer safe for applications to use it in this way:

Where CREATE2 is used to redeploy contracts at the same location to make the contract upgradeable. This functionality is no longer supported and should be replaced with ERC-2535 or other types of proxy contracts.

If a contract relies on burning Ether by SELFDESTRUCTing a contract as a beneficiary that was not created in the same transaction.

Risks associated with smart contracts

EIP1153

Two scenarios are imagined using the opcodes TLOAD and TSTORE:
- The called contract uses this opcode
- The calling contract uses this opcode

Risk 1:

Compared to the traditional SSTORE and SLOAD, the added transient storage mainly changes the storage duration of the data. The data stored by tstore is read by tload, and the data will be released after a transaction execution ends, rather than being permanently recorded as with sstore. Developers should understand the characteristics of this opcode to avoid misuse that may lead to data not being correctly written into the contract, causing losses. Also, the data of tstore is a private variable, which can only be accessed by the contract itself. If you want to use this data externally, you can only pass it as a parameter or temporarily store it in a public storage variable.

Risk 2:

Another potential risk is that if smart contract developers do not properly manage the life cycle of transient storage variables, data may be cleared or wrongly retained when it shouldn't be. If the contract expects to use data stored in transient storage in subsequent calls of the transaction, but fails to properly manage the life cycle of these data, data may be wrongly shared or lost between different calls, leading to logical errors or security vulnerabilities. Considering that data like balance or allowance in projects like Token not being properly stored will lead to errors in contract logic, causing losses. Or using this opcode when setting the owner address will cause the privileged address to not be correctly recorded, losing the modification of important contract parameters.

Consider a smart contract that uses transient storage to temporarily record trade prices on a cryptocurrency trading platform. The contract updates the price after each transaction and allows users to query the latest price in a short time. However, if the contract design does not take into account the characteristic of transient storage being automatically cleared when the transaction ends, then in the time between the end of one transaction and the start of the next, users may get a wrong or outdated price. This could not only lead to users making decisions based on incorrect information, but also be maliciously exploited, affecting the platform's reputation and user asset security.

EIP-6780

This proposal changes the behavior of the previous selfdestruct opcode, does not destroy the contract, only transfers tokens, and only contracts created in the same transaction with self-destruction will be destroyed. The impact of this EIP is relatively large.

Using create2 to redeploy contracts at the same address to make the contract upgradeable. This feature is no longer supported and should be replaced with ERC-2535 or other types of proxy contracts. (This may affect the security of chain contracts that use create2 to implement upgradeable contracts)

In smart contracts, the SELFDESTRUCT operation allows contracts to be destroyed and sends the contract balance to a specified target address. In this case, the contract uses SELFDESTRUCT to destroy Ether and sends the destroyed Ether to the contract. But the contract can only be a contract created in the same transaction (a contract created by this contract or other contracts in the same transaction). Otherwise, only Ether will be transferred without destroying the contract (for example, self-destruction and the beneficiary is the self-destructing contract, this will not make any change). This will affect all contracts that depend on selfdestruct for withdrawal or other operations.

A Gas Token similar to the 1inch CHI Token works by maintaining an offset, always executing CREATE2 or SELFDESTRUCT at this offset. After this update, if the contract at the current offset has not been correctly self-destructed, the subsequent CREATE2 will not be able to successfully deploy the contract.

The implementation of this proposal cannot directly cause attacks on the contract, but it will damage the normal logic of the originally deployed contracts that rely on the selfdestruct operation (contracts that only rely on self-destruction for fund transfer are not affected, but if subsequent operations require self-destructed contracts to be deleted, they are affected), leading to unexpected contract operations. From the perspective of the contracts and users, this could lead to contract strikes, losses of funds, and other damages (for example, the original use of create2 to deploy a new contract at the original address, self-destruct the original contract for upgrading, can no longer be successfully deployed). In the long run, modifying the functionality of an opcode could lead to centralization issues.

For example, there is a vault contract to be updated:

- The create2 temporary storage contract is used to temporarily reserve funds for the vault.
- Self-destruction of the vault contract, funds are transferred to the temporary contract (only transferring funds without destroying the contract).
- A new vault contract is created at the original address using create2 (fails because the original vault contract was not destroyed).
- The temporary contract is self-destructed to return the funds to the vault (loss of funds, as the vault contract was not created).

Extended Reading

The Cancun upgrade will further enhance Ethereum's competitive advantage. However, the changes to the core smart contract layer brought about by this upgrade carry risks, which could affect the secure operation of existing DApps. These changes and potential risks also need to be highly regarded during the process of smart contract development. You can contact Salus for risk checks or audit support, or you can understand the changes by reading related content further.

- Cancun Network Upgrade Specification
- EIP-1153
- EIP-4788
- EIP-4844
- EIP-5656
- EIP-6780
- EIP-7516
- Metapod contract
- GasToken2 contract

Shawn
Salus
pr@salusec.io
Visit us on social media:
Twitter
LinkedIn