Removal of SELFDESTRUCT

An Impact Study on EIP-4758 & EIP-6780

May 23, 2023

Abstract

Dedaub was commissioned by the Ethereum Foundation to perform an impact study of Ethereum Improvement Proposals (EIPs) 4758 and 6780 on existing contracts. EIP-4758 proposes to deactivate `SELFDESTRUCT` by changing it to `SENDALL`, which recovers all funds (in ETH) to the beneficiary without deleting any code or storage. On the other hand, EIP-6780 modifies `SELFDESTRUCT` to work only in the same transaction in which the contract was created, while in all other cases it recovers all funds but does not delete any other account data.

The aim of this study is (i) to help the Ethereum community decide whether to implement, based on the impact of these changes to the ecosystem, EIP-4758 or EIP-6780. In either case we also aimed to (ii) find out which projects are affected and by how much. To evaluate the impact of these proposed changes, we performed comprehensive queries over past on-chain behaviors of smart contacts and queries on code and bytecode of deployed contracts; inspected code manually; checked balances, approvals, and contract proxying state; and informally interviewed developers.

The study was conducted by the equivalent of 2 full-time engineers over 2 weeks.

Summary

For the proposed EIPs, we were able to measure the extent of the impact of these changes. The effects are observed on a handful of known projects, and on many additional smart contracts forming part of (what mostly looks like) MEV bot networks.

From a purely quantitative aspect, the number of specific opcode invocations between blocks 15 - 17.23m reveal the extent of the impact of the removal of `SELFDESTRUCT`. Over 98% of invocations of `SELFDESTRUCT-CREATE2` pairs in known contracts will be unaffected if EIP-6780 is implemented, while the impact of EIP-4758 is less certain. In the latter case, although we know a number of projects will break, many of these can be upgraded. Another observation we made is that metamorphic contracts used for the purposes of upgrades are very rare.

If EIP-4758 were to be deployed today it would break some functionality of the following projects, which can otherwise operate without modifications if EIP-6780 is to be deployed instead. The impact, based on our understanding of the project, is also estimated.

Project Name	**Website**	**Estimated Impact**
AxelarNetwork	https://axelar.network/	High
Sorbet Finance	https://www.sorbet.finance/	Low
Celer	https://celer.network/	Low
Gelato	https://www.gelato.com/	Low
Pine Finance	https://pine.finance/	High
Ricmoo's Wisps	https://github.com/ricmoo/will-o-the-wisp	Low
Revest	https://revest.finance/	High

Chainhop Protocol	https://app.chainhop.exchange/	Low
JPEGd	https://jpegd.io/	High
Thousand Ether Homepage	https://thousandetherhomepage.com/	Moderate

Although EIP-4758 could break the above projects today, we also believe that most of these can be upgraded in time for a deployment of the proposed EIPs to occur. As a result, we are of the opinion that the impact of EIP-4758 and EIP-6780 is manageable and perhaps a net positive due to the simplicity of the implementations of Ethereum Clients, especially if EIP-6780 is selected. We recommend reading the section titled “Opinion“ at the end of this report for more detail and documentation of our subjective opinions. The objective, numeric findings of the study are detailed in the “Experimental Findings and Study“ section.

Experimental Findings and Study

This section describes the experiments conducted during this impact study. The methodology utilized dynamic analysis (observations of past blockchain transactions), and, in order to improve the recall, static program analysis. In addition, we have inspected smart contracts or debugged past transactions to better understand the impact of the EIPs being studied.

Evaluation based on quantitative impact

We performed a quantitative assessment on the use of `SELFDESTRUCT` and the use of metamorphic contract patterns. The experiment queried an analytical Dedaub Watchdog database, a database that is optimized for inspecting fine-grain EVM execution traces at scale. In the experiment we measured the number of specific opcode invocations between blocks 15 - 17.23m, based on both low-level and high-level patterns that may be disallowed by the EIPs in question. The results can be found below:

Pattern	**Instances**
`CREATE` or `CREATE2`	9576237
`CREATE2`	8545615
`SELFDESTRUCT`	549263
Patterns Below Impacted by EIP-4758	
Short-lived smart contract (not impacted by EIP-6780) `TX1: c.CREATE2 ; …; c.SELFDESTRUCT`	402549

Short-lived, metamorphic pattern with address reuse (not impacted by EIP-6780)
 	
`TX1: c.CREATE2 ;…;  c.SELFDESTRUCT`
`TX2: c.CREATE2 ;…;  c.SELFDESTRUCT`
	22041

Long-lived metamorphic pattern, disallowed by either EIP
 	
`TX1: c.SELFDESTRUCT`
`TX2: c.CREATE2`
	735

What really stands out, from a purely quantitative view, is that long-lived metamorphic contract instantiations are very rare. Long-lived refers to the fact that these contracts are created and destroyed in different transactions. Metamorphic refers to the fact that these smart contracts change their bytecode, in-place, over time.

Also to be noted is that *most smart contract creation events* within smart contract code *are performed using CREATE2*, with an order of magnitude fewer SELFDESTRUCTs. Interestingly, it appears that most of these SELFDESTRUCTs are currently performed in tandem with a CREATE or CREATE2, in the same transaction, yielding short-lived smart contracts.

There is more than an order of magnitude reduction in instances of metamorphic contract creations allowed by EIP-6780 than there are short-lived smart contract creations. Furthermore, there are almost 3 orders of magnitude fewer invocations of long-lived metamorphic contracts. This means that, purely quantitatively, the impact of either EIP is small, and even more so for EIP-6780.

Querying historical transactions to check EIP-6780 compliant projects

The goal of this task is the detection of protocols that employ the atomic (same tx) (c.CREATE2 ;..; c.SELFDESTRUCT) pattern, as well as the frequency of this pattern.

In order to answer this question, we constructed a query that looked at historical transactions in the block range 15M - 17.2M (17,230,565 to be exact). At a high level, the query performs the following:

1. Find the selfdestruct call frames, which includes the target of the selfdestruct, i.e., the contract being destroyed, while respecting EVM semantics: if the enclosing callframe is a delegatecall, then the sender address of the callframe

(`from_a` in the query below) is the contract being selfdestructed – otherwise it is the receiver address (`to_a`).

2. With this information, we search for pairs of CREATE2-SELFDESTRUCT events. This boils down to finding CREATE2 call frames that are in the same transaction (aka same block number and transaction index) and precede the selfdestruct events.
3. Finally, in order to cut down on the search space, we collapse the result set, grouping by the top-level callframe/entrypoint address, also requiring that there be more than 1 such transaction for each (entrypoint address, self destruct target) pair we observe. Furthermore, as we want to evaluate how common this pattern is among large protocols, we limit our search to contracts with verified source.

The final query is the following:

```sql
-- Get the selfdestruct call frames
--   -> if it's a delegate call
--     then the selfdestruct target is from_a,
--     else it's to_a
with self_destruct_contracts as (
select
  td.block_number,
  td.transaction_index,
  td.vmstep_start,
  sd.vmstep,
  case
    when td.call_opcode = 'DELEGATECALL' then td.from_a
    else td.to_a
  end as target
from
  self_destructs sd join transaction_detail td on
    (td.block_number, td.transaction_index, td.vmstep_start) =
    (sd.block_number, sd.transaction_index, sd.vmstep_start)
),
addr_groups as (
select
  cm.address,
  count(distinct (sd.block_number, sd.transaction_index, sd.target)) n_txs,
  (array_agg(distinct tx_hash(sd.block_number, sd.transaction_index)))[1] as txs
from
  self_destruct_contracts sd
  -- td is the top-level call-frame
  join transaction_detail td on
    (td.block_number, td.transaction_index, td.vmstep_start) =
    (sd.block_number, sd.transaction_index, 0)
  -- tdc is the creation callframe
  join transaction_detail tdc on
    (tdc.block_number, tdc.transaction_index) =
    (sd.block_number, sd.transaction_index) and tdc.vmstep_start < sd.vmstep
  join contract_metadata cm on cm.address = td.to_a
where
  cm.has_source -- top-level call-frame receiver is a contract with verified source
  and tdc.to_a = sd.target
  and tdc.call_opcode = 'CREATE2'
group by sd.target, cm.address
having count(distinct (sd.block_number, sd.transaction_index, sd.target)) > 1
order by count(distinct (sd.block_number, sd.transaction_index, sd.target)) desc
), event_pairs_collapsed as (
  select address, sum(n_txs) as n_txs, (array_agg(txs))[1] as sample_tx
  from addr_groups
  group by address
)
select contract_name, c.address, n_txs, sample_tx
from event_pairs_collapsed c join contract_metadata using (address)
order by contract_name
```

The query yielded 16 entrypoint contracts matching our criteria. Upon manual inspection, the following were deemed most interesting:

Contract Name	Entrypoint Address	#txs	Sample tx	Protocol/ Project
AxelarGatewayProxyMultisig	0x4f449524...	2521	0x001b0786...	AxelarNetwork
AxelarDepositServiceProxy	0xc1dcb196...	933	0x00051ec5...	AxelarNetwork
OptimizedTransparentUpgradeableProxy*	0x4066d196...	1	0x00fb1d5f...	Celer
OptimizedTransparentUpgradeableProxy*	0xed8877f8...	1	0x36112b61...	Chainhop
Gelato	0x3caca7b4...	6	0x01a5eeaa...	Gelato Network
TransparentUpgradeableProxy	0x4e5f305b...	2	0x02f813ff...	JPEGd
TransparentUpgradeableProxy	0xd636a2fc...	4	0x01ad27a4...	JPEGd

Contract Name	Entrypoint Address	#txs	Sample tx	Protocol/ Project
TransparentUpgradeableProxy	0x923a36f8...	4	0x055b6424...	JPEGd
TransparentUpgradeableProxy*	0x2be665ee...	1	0x146debc1...	JPEGd
TransparentUpgradeableProxy	0x7bc8c4d1...	4	0x12ac08d6...	JPEGd
PineCore	0xd412054c...	4	0x1bc54f70...	Pine Finance
RevestA4	0x9f551f75...	1	0x0128b590...	Revest
Springboard	0x50bece19...	1608	0x000834e4...	Ricmoo's Wisps
Springboard	0x7c0cc141...	13	0x734be821...	Ricmoo's Wisps
GelatoPineCore	0x36049d47...	410	0x00c65301...	Sorbet Finance
Thousand Ether Homepage Ad*	0x7bb952ab...	1	0x140ae6dd...	Thousand Ether Homepage

*In this task we ignored (`target.create2()`, `target.selfdestruct()`) pairs that only appear once for a given target, as it's a good indicator that target address is not being reused. However, we included some of these instances due to their connection to known protocols/projects.

It should be noted that there are a few projects in the above list–e.g. JPEGd contracts–that are upgradable, which means that it's quite likely they can update their implementation to eliminate the affected pattern, if EIP-4758 is chosen over EIP-6780.

A project that stands out in the above list is Axelar Network, with its AxelarGatewayProxyMultisig contract having a TVL of over $85M. However, similarly to JPEGd, the protocol appears to be upgradeable, and thus any disruption in the protocol's operation can be resolved by updating the implementation.

On the other hand, Pine Finance's PineCore and Sorbet Finance's GelatoPineCore, despite their relatively modest TVL relative to that of Axelar (around $11.5K and $16.6K respectively, for these specific contracts), are not upgradeable and their operation will be irreversibly affected if EIP-4758 is implemented instead of EIP-6780.

As the table subtext suggests, we mainly focused on contracts that are seen to be created and selfdestructed more than once. Despite this, we included some contracts in the table that displayed only one `target.CREATE2-target.SELFDESTRUCT` pair:

- Celer, Chainhop: Both of these are really similar, both in their objective (facilitating cross-chain functionality), as well as in the way they use the `CREATE2-SELFDESTRUCT` pattern. Upon closer inspection, we found that the salt used in the CREATE2 instruction was message ID/nonce which appears to be unique. Assuming this uniqueness property holds, address uniqueness is impossible and thus the projects are unaffected by either EIP.
- Interestingly, in both Chainhop and Celer we found that "Pocket" contracts are deployed, to receive cross chain funds. When funds are claimed, these Pockets are destroyed, which also burns any remaining ERC20 tokens within them. If SELFDESTRUCT does not remove smart contract code, any unaccounted for tokens could potentially be claimed by anyone. Although this is a change in semantics for these "Pockets", we don't think this poses a security risk.
- Thousand Ether Homepage: Looking at the source code, we can see that the `CREATE2` salt is a function of the (`_idx`, `_owner`) parameters, which we believe is definitely possible to see occur twice, at least in theory, as there is nothing in the contract code that prevents this. However, it's highly unlikely that this will happen, which explains why we haven't observed any address reuse.
- Revest: In this project, the `CREATE2-SELFDESTRUCT` pattern is used as part of a fund withdrawal operation, which involves the burning of some NFTs, the ID of which is used as the `CREATE2` salt. We've confirmed with the developers of Revest that It is possible to issue 2 ERC1155 tokens to two different accounts. In this case if one account withdraws from that vault some of the assets the other account cannot access the assets if EIP-4758 goes through.

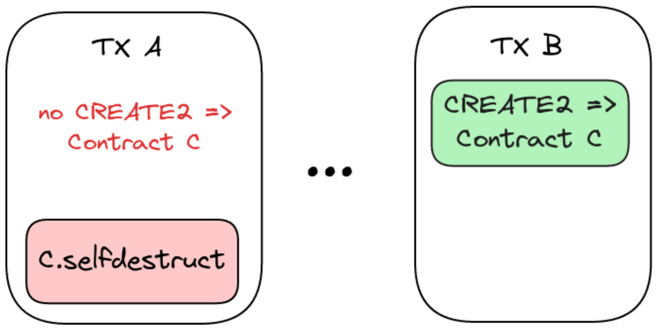
An interesting set of contracts that were not included in the above table are the MultV5 contracts:

Contract Name	Entrypoint Address	#txs	Sample tx
MultV5	`0x7478a6bf...`	738	`0x5b67eb15...`
MultV5	`0x3365889c...`	14	`0x0a3eefd5...`
Mult_V5	`0x6139eab2...`	236	`0x385045d7...`

These appear to make frequent use of the CREATE2-SELFDESTRUCT pattern. Fortunately, they seem to be mostly inactive (latest tx is 1 month ago for the 0x6139... contract) and hold either no or low value in tokens.

Querying historical transactions to detect projects that employ non-atomic SELFDESTRUCT, CREATE2 pairs.

The goal of this query was to identify projects that would break with both EIP-4758 and EIP-6780. This essentially boils down to the detection of SELFDESTRUCT-CREATE2 pairs that do not happen in the same transaction. We additionally require that no CREATE2 event took place in the same transaction as the SELFDESTRUCT, to exclude EIP-6780 compliant pairs, and that more than one c.SELFDESTRUCT-c.CREATE2 pair is observed. As before, our search considers transactions/events in the block range 15M - 17.2M.



The SQL query that was used to detect the target of these transaction pairs is the following:

```
with self_destruct_contracts as (
  select
    td.block_number,
    td.transaction_index,
    td.vmstep_start,
    sd.vmstep,
    case
      when td.call_opcode = 'DELEGATECALL' then td.from_a
      else td.to_a
    end as target
from self_destructs sd join transaction_detail td on
  (td.block_number, td.transaction_index, td.vmstep_start) =
  (sd.block_number, sd.transaction_index, sd.vmstep_start)
),
self_destructs_no_create_same_tx as (
  select sd.*
  from self_destruct_contracts sd
  where
    not exists(
      select 1
      from transaction_detail td
      where
        (td.block_number, td.transaction_index) =
        (sd.block_number, sd.transaction_index)
        and td.to_a = sd.target
        and td.call_opcode = 'CREATE2'
    )
)
select
  sd.target,
  count(*),
  array_agg(distinct tx_hash(sd.block_number, sd.transaction_index)),
  array_agg(distinct tx_hash(tdc.block_number, tdc.transaction_index))
from
  self_destructs_no_create_same_tx sd
  join transaction_detail tdc on
    (tdc.block_number, tdc.transaction_index) >
    (sd.block_number, sd.transaction_index)
where
  tdc.to_a = sd.target
  and tdc.call_opcode = 'CREATE2'
group by sd.target
having count(*) > 1
order by count(*) desc
```

88 such addresses were detected. Upon manual inspection, two contracts stood out, while a third one was found using other methods:

Address	Sample SELFDESTRUCT TX	Sample CREATE2 TX	Note
0x1ac01ebe...	0x000be41d...	0x0277e45c...	The top-level contract of the selfdestruct transaction was found to belong to the

			Sorore NFT project (tagged on Etherscan)
0xd1742b3c...	0x2eae9ed6...	0x62a1a4cc...	This contract is being used by the crypto trading firm Seawise, and we've detected 28 SELFDESTRUCT-CREATE2 pairs.
HopImplV2	Not found	Not found	Part of the Hop protocol. It's used to deploy a disabled contract at destroyed route to handle it gracefully.

The rest of the contracts appear to mainly be MEV bots, written in low-level code.

Detection of contracts that hold tokens but can't currently transfer them to a 3rd party.

In this task, we wanted to detect if there are metamorphic contracts that hold tokens but have no way of transferring them to any other address with their current implementation (motivated by this Ethereum Magicians post).

A mix of static analysis and querying of our contract/token balance database was used. More specifically, we started by analyzing the entire corpus of deployed contracts to detect possible CREATE2 factories.

We started from a query in our database for all possible factories (i.e. contracts that have deployed other contracts). This query returned 28403 distinct smart contract bytecodes. We then analyzed these contracts using the gigahorse framework to find out which of them perform CREATE2 operations. This analysis had a 99.7% success rate (timed out for 75 contracts) and returned a dataset of 3243 unique smart contracts bytecodes that use CREATE2.

Using this dataset, we can find all contracts created by CREATE2, with a simple query in our database:

```
-- Get addresses of marked create2 factories
with deployers as (
  select address
  from contracts dc join create2_factories f on
    dc.md5_bytecode = f.md5_bytecode
),
-- Get deployed contracts from said factories
targets as (
  select c.address
  from contracts c join deployers cd on
    c.deployer = cd.address
),
-- If deployed contract is a proxy, get implementation address, else keep the
address as is
implementations as (
  select
    coalesce(implementation_address, t.address) as address,
  from targets t left join proxy_implementation on proxy_address = t.address
)
select distinct on (md5_bytecode) md5_bytecode, bytecode_hex
from implementations join contracts using (address)
     join bytecode using (bytecode_md5)
```

This yields a superset of metamorphic contracts, but is a good starting point for our task.

Note the final CTE in the query. In the case we know that an address is a proxy, we fetch the implementation contract instead. This will allow us to analyze the code that matters and not the proxy boilerplate, which is practically useless to our static analysis.
Our analysis for these contracts intended to find those that could be destroyed but had no way to withdraw ERC20 tokens:

```
MoneyCanBeStuckInContract():-
  SELFDESTRUCTInContract(),
  not ERC20TransferOpInContract(),
  not ArbitraryCallInContract(), // i.e contract.call(_data)
  not DELEGATECALLInContract().
```

The analysis flagged around 448 distinct bytecode md5s, which corresponds to around 11.308.693 contract addresses. While the vast majority (more than 10.000.000) are CHI Gas Token relics, in order to reduce the number of contracts for manual inspection we further required that the total value held by the contract be at least 1K. This additional filtering was enough to reduce the list to around 17 contracts which can be inspected manually.

All the contracts we inspected were found to be false positives, and can be split in two categories:

1. Contracts that are extremely low-level and optimized. These include MEV bots that contain hand-written and non-standard patterns that our analysis couldn't resolve. All of these appear to have a way to drain the funds of the contract.
2. Contracts that hold only ETH. This is not an issue, because the `SELFDESTRUCT` instruction (`SENDALL` post EIP) is essentially an ETH drain, so there is no risk of ETH getting stuck.

To summarize:

1. We created a dataset of contracts created by factories that contain `CREATE2`. This should be a superset of metamorphic contracts.
2. We wrote a static analysis to detect contracts that can selfdestruct and have no way to withdraw ERC20 tokens.
3. We inspected the contracts marked by the analysis that hold more than 1K in value—no contract that was at risk of locked funds was found.

Impact Opinion

As elucidated in previous sections, several projects utilize metamorphic contract patterns, in particular for safe transfer of Ether, however, within many of these projects the impact is moderate. If the EIP-6780 implementation is selected, most known projects will not be impacted. The only exception being some low-level MEV bots and a couple of obscure projects.

Evidence of long-lived metamorphic pattern misuse. Interestingly, over the course of the study we also noticed anecdotal evidence of metamorphic patterns disallowed by

EIP-6780 being detrimental to the security of Ethereum projects. In particular, while conducting this study a hacker exploited the opaqueness of the semantics of SELFDESTRUCT to propose a governance proposal on Tornado Cash. The executor contract of this proposal was subsequently recreated, after the proposal passed. The smart contract executor was created using the CREATE instruction, so any casual reader would not have immediately suspected that the contract is metamorphic (note that most metamorphic patterns in the wild utilize CREATE2). However, SELFDESTRUCT can be used to reset the nonce of a smart contract, as in the case of the hacker's transaction. Hence, any contract created using a metamorphic contract factory can itself be metamorphic. Therefore, currently, to ascertain that a contract is not metamorphic the deployment transaction needs to be inspected in order to see whether it was created by the CREATE2 opcode. This needs to be carried out also on the deployer, recursively, until an EOA is found at the end of the contract→deployer chain.

Evidence of metamorphic pattern usage over time. Over the course of blocks 15m - 17.2m we haven't seen any statistically significant evidence of the use of CREATE2 and SELFDESTRUCT changing. This was established even after normalizing for the activity happening on the Ethereum blockchain. Therefore, we think that awareness in itself will not detract developers from utilizing SELFDESTRUCT in a destructive manner if the EVM allows it.

Upgradeability of affected projects. If EIP-4758 is implemented, we noticed that many of the affected protocols can be upgraded. In particular, Axelar, Chainhop, Celer, Jpegd, Gelato & Sorbet can be upgraded, according to their smart contracts. We haven't however established whether the addresses controlling these smart contracts are reachable or whether their governance is functional.

Conclusion

In our comprehensive study of the consequences of introducing EIP-4758 and EIP-6780 on existing contracts, we have observed the impact to be largely manageable. Both EIPs are projected to affect only a limited portion of the known projects and certain MEV bots. This is especially so for EIP-6780, which affects no “well-known” projects, meaning that most operations will remain largely untouched. We can anticipate that the EIP-6780 would only impact obscure projects utilizing metamorphic contract patterns.

Our study also references instances of misuse of the metamorphic pattern that may be curtailed by either EIP. These cases have demonstrated potential security risks associated with the pattern, exemplifying the need for vigilance and thorough validation during contract deployment.

Consequently, we categorize the impact of EIP-6780 as “low and manageable”, while for EIP-4758 the impact is “moderate and manageable”. Finally, all of the findings in this report have to be consumed with the understanding that any interpretation of our findings can vary between people.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.