On Parity's proposed changes to SELFDESTRUCT behaviour

or, retroactive changes to invariants considered harmful

Nick Johnson

Follow

--

4

Listen

Share

When analysing a system, one of the critical things to examine is its invariants

: that is, what guarantees does it offer?

For instance, a well written token contract has the invariant that sum(balance(account) for all accounts) = totalSupply

— that is, the total supply is equal to the sum of all account balances. If that's no longer the case, it's a sign something is seriously wrong.

Invariants are particularly important because people designing on top of a system often implicitly assume they are true, even when they're not explicitly described anywhere. That means that any time we are considering breaking or modifying an invariant, we should look very carefully to see whether anything could be explicitly or implicitly depending on it.

Ethereum has a number of invariants, some of them more critical to the system — and hence more relied upon — than others. To name a few:

1. The sum of gas used by all transactions in a block is less than or equal to the block's gas limit.
2. The sender of a transaction is the account whose address is returned by calling ecrecover

with the hash of the transaction data and the signature.

1. Before executing the transaction, the sender must have at least enough funds to cover value + gasPrice * gasLimit

.

1. The nonce of a transaction must match the nonce stored in the state trie.
2. The state root hash in a block is the result of executing the block's transactions against the state represented by the parent block's hash.

The last example was broken by the DAO hard fork, by inserting an 'irregular state change'; the new version reads something like:

The state root hash in a block is the result of executing the block's transactions against the state represented by the parent block's hash, except if the block number is 1,920,000, in which event additionally make the following changes…

Breaking or changing invariants isn't always a bad thing, but it always needs to be approached with great caution. A one-off change to an invariant, like the DAO example above, is a lot easier to evaluate than an ongoing change — even a seemingly harmless one.

When considering a proposed change, we need to consider:

1. The nature of the change.
2. What benefits the change grants us.
3. The consequences of the change, and how certain we can be about them.

We should also require a comprehensive analysis of the effects a change will have, so we can be certain about the impact of the decision we're making. One example of this was prior to the DoS hard forks where gas cost repricing was required; several individuals undertook analyses to determine if there were contracts that depended directly on the specific gas cost of operations; the results of this lead to the "63/64 rule" to permit such contracts to continue to operate correctly. Where an

analysis isn't possible or isn't done, we should err on the side of caution, since we can't be certain of all its consequences.

All of which brings us to Parity's recently published proposal to make some changes to the SELFDESTRUCT opcode to provide facilities for recovering funds made inaccessible due to destroyed contracts. The raison d'etre for this, naturally, is the recent parity wallet issue.

They present four variants of the same basic proposal:

1. Add a builtin, CREATEAT

, which creates an arbitrary contract at an address derived from the caller's address and a specific past nonce.

1. Add a builtin, CLAIM

, with the same effect as 1, but with a 500,000 block delay before the created contract is usable.

1. Add a builtin, PROXY

, with the same effect as 1, but requires the caller to first explicitly approve the proxy call. Additionally, transfer all the funds at the newly created proxy contract's address to the caller.

1. Add a builtin MULTI_PROXY

, which creates a proxy contract at an address derived from the caller's address and a specific past nonce, and allows any caller of that to configure the address calls are forwarded to. Additionally, transfer all the funds at the newly created proxy contract's address to the caller.

I won't drill down on the specific detail of the different variants, because these proposals all have one thing in common: they violate a couple of implied invariants in Ethereum. Roughly stated, the invariants are:

1. The code at a contract's address will never change, except to be deleted if the contract self destructs.
2. A contract's creator has no special access or privilege in regards to the created contract.

It's worth noting that the "except" in the first invariant is the reason for Parity's issue in the first place — the authors of the Parity library code assumed the contract's code was truly immutable, and failed to allow for the one special case where it was not.

The second invariant is also crucial to the trustless nature of Ethereum; we don't need to know anything at all about the creator of a contract unless something about them is coded into the contract.

Now Parity are proposing eliminating the second invariant, and replacing the first invariant with an altogether weaker one:

The code at a contract's address may be deleted if the contract self destructs, after which it may be changed to code chosen by the original creator (option 1) or a standard proxy contract (options 2, 3, 4).

Unlike the DAO change, this change is ongoing — and critically, it's retroactive as well, affecting contracts that were written and deployed under the previous set of assumptions.

Because the current invariant is more implied than explicit, it's difficult to fully analyze how people may be depending on it. In my opinion, it's likely that many existing contracts implicitly depend on this invariant, potentially in very nonobvious ways. Some examples that spring to mind:

1. Systems like Etherdelta upgrade themselves by deploying a new contract, migrating users to the new contract, then destroying the old one. This gives users an opportunity to examine the updated code if they wish. With some variants of this change, the authors of Etherdelta could immediately destroy and replace the contract with any code they wish, without users' knowledge.
2. Reviewers may have made assumptions about the incentives for the owner of a contract. If an owner can destroy a contract, but by doing so can only lock funds up, they have much less incentive to do so than if they can then replace the contract to steal the funds instead.
3. Parity's analysis focuses on assets (Ether and tokens), but these are not the only effects a contract can have; for instance, a smart contract can also have permissions granted by other contracts. With this change, a contract with ERC20 approvals could be destroyed and replaced by one that steals the tokens it's approved to access. See the Etherdelta example in point 1 for an example of where this could happen.

Further, I'm unconvinced that this change grants a useful new capability to smart contracts that they did not previously possess; instead, it appears to add another edge-case contracts wanting to guarantee trustlessness must work around.

So, looking at our 3 point criteria above:

1. Nature of the change:

A weakening of the invariant relating to contract code immutability, and eliminating the invariant that a contract's creator has no special privileges.

1. Benefits:

Recovery of funds locked in Parity wallets, and an unknown quantity of other funds.

1. Consequences:

Unknown and difficult to quantify due to the quantity of code already deployed that may call SELFDESTRUCT.

To the best of my knowledge, nobody has yet attempted a thorough assessment of the consequences of this change to existing contracts — and doing so would be very difficult, given the number of existing contracts that contain SELFDESTRUCT, and the fact that many of them will have no published source code.

Because of the risks and the level of uncertainty surrounding them, I personally can't recommend any of the four variants of this proposal for adoption. I'm skeptical there is any variant of this proposal that would sufficiently alleviate the risks involved to be worthwhile, though I'm naturally willing to consider new variants with an open mind.