

Security Review of Gnosis Safe

November 11, 2019

Overview

G0 Group was engaged to perform a security review of Gnosis Safe v1.1.0. G0 Group was contracted for an eight person-week effort to that end. The primary subjects of this review were the smart contracts which implement the Gnosis Safe: an extensible, multi-signature smart contract wallet. This review was initially performed on <https://github.com/gnosis/safe-contracts/commit/1a9e5ce768e134c556770ea50e114fd83666b8a8>.

Files in Scope

```
contracts/  
  base/  
    Executor.sol  
    FallbackManager.sol  
    Module.sol  
    ModuleManager.sol  
    OwnerManager.sol  
  common/  
    Enum.sol  
    EtherPaymentFallback.sol  
    MasterCopy.sol  
    SecuredTokenTransfer.sol  
    SelfAuthorized.sol  
    SignatureDecoder.sol  
  handler/  
    DefaultCallbackHandler.sol  
  interfaces/  
    ERC1155TokenReceiver.sol  
    ERC721TokenReceiver.sol  
    ERC777TokensRecipient.sol  
    ISignatureValidator.sol  
  libraries/  
    CreateAndAddModules.sol  
    CreateCall.sol  
    MultiSend.sol  
  modules/  
    DailyLimitModule.sol  
    SocialRecoveryModule.sol  
    StateChannelModule.sol  
    WhitelistModule.sol
```

```
proxies/  
  DelegateConstructorProxy.sol  
  PayingProxy.sol  
  Proxy.sol  
  ProxyFactory.sol  
GnosisSafe.sol
```

Result Summary

During the course of this review, 6 issues were discovered and reported. None of these issues constitute an immediately exploitable security vulnerability; however, users should be aware of them as they concern additional precautions users should take to ensure predictable behavior of the safe. Further developing client side tools to verify the state history and providence of the safe, as discussed below, would make using the safe securely easier.

No further issues were discovered in

<https://github.com/gnosis/safe-contracts/commit/78494bcd61b3db52308a25f0556c42cf656ab1> (v1.1.0)

Issues

1. Safe state integrity can be maliciously corrupted through delegatecall, leading to a lack of predictability

Type: security / Severity: dependant on use

There are multiple ways through which a quorum of owners can induce a `delegatecall` from the safe contract that can lead to an arbitrary modification of the contract's state. Most significantly, this can lead to addition of hidden entries in the `owners` mapping structure defined on the `line 15` of `OwnerManager.sol` which holds the record of authorised owners of the contract and the `modules` mapping structure defined on the `line 18` of `ModuleManager.sol`. These entries are not only invisible through a standard getter functions of the contract's interface, but can also be added in a way that makes the state entry unidentifiable and undecodable until it is activated through submitting a message authorised through the owner's address or private key. Any unexplained state modification that occurred as a result of a `delegatecall` is suspect, and results in a loss of transparency regarding the contract's ownership. This means that any owner that hasn't taken part (at least as an observer) in all of the contract's past calls can only verify the contract's ownership structure after carefully examining all past state changes. This makes the contract potentially ill-suited for use cases where dynamic ownership is expected: as this burden of verification could be relatively high.

2. Fragile code segment in StateChannelModule.sol can lead to creation of reentrancy vulnerability in the future

Type: security / Severity: potential issue (fragile code)

`checkHash()` function call in `StateChannelModule.sol` on `line 45` is positioned in between check that ensures identical call has not been already executed and state update that marks the current call as executed, if there's an update to the `checkHash()` function or the downstream functions in the future that introduces an external call to an untrusted address, it will allow an attacker to re-enter the contract to execute the same call multiple times. To prevent this possibility the call should be moved two lines down, under the state update.

3. Safe transfer not used in DailyLimitModule.sol

Type: security / Severity: low

Safe transfer is not used for token transfers in the `DailyLimitModule` contract, potentially leading to certain malformed tokens being incorrectly marked as spent for the day even if no actual transfer occurred.

4. Notes on deployment

Type: note

It's necessary to ensure that masterCopies aren't controlled by anyone, and can't be maliciously `selfdestructed` or replaced (via `create2`). This has to be achieved by correct deployment. Ideally, this would be easily verifiable by users. In the case of the current iteration of the safe contract, users would verify that the provided masterCopy was setup without: accessible owners (e.g. Gnosis intends to use 0x02 & 0x03), any modules, or fallback manager set. This ensures no further transactions will be executed on said deployment, since the ownership of the safe has been given to inaccessible accounts; and that no unexpected state changes occurred during setup. In general, specific attention should be paid to masterCopies which have the ability to make arbitrary delegate calls: to ensure that this functionality is not accessible to potential attackers.

5. In SocialRecoveryModule.sol, an identically configured recovery can't be executed multiple times

Type: usability

In the unlikely event that an identical owner replacement needs to be executed multiple times, it isn't possible to do it directly in the current version of the contract: it has to be done through an intermediate address because of the implemented protections against replay attacks.

6. Note on hardcoded storage addresses

Type: note

It's important for users to verify that any hard-coded storage address like the one in `FallbackManager` is generated in a way that precludes intentional collision with a storage slot that is used by another state variable. The employed technique of hashing english strings seems like a good way to ensure that.