[READY TO SIGN] Runbook: Support EAS

Aug 31, 2023

Context

Optimism deployed SchemaRegistry and EAS contracts (part of Ethereum Attestation Service) to Predeploys (see here). We are adding the same setup to Base to keep feature parity and also since several teams are requesting it.

The signing task requires upgrading two of the reserved predeploy addresses on the Base network to point to EAS related implementations. Specifically the `0x4200000000000000000000000000000000000020` address will point SchemaRegistry and the `0x4200000000000000000000000000000000000021` address will point to the EAS contract.

Steps

1. Pull main branch of base-org/contract-deployments:

   ```
   cd contract-deployments
   git pull origin main
   ```

2. Ledger setup:
   a. Ledger needs to be connected and unlocked
   b. Ethereum application needs to be opened on Ledger with the message "Application is ready"

3. Signing:

   The script you will run will test that the multisig you are in can successfully complete a contract upgrade (therefore proving ownership). It does so using test contracts, so that we can simulate *as if* the Safe was the owner before actually transferring ownership.

   First get to the directory for this task:

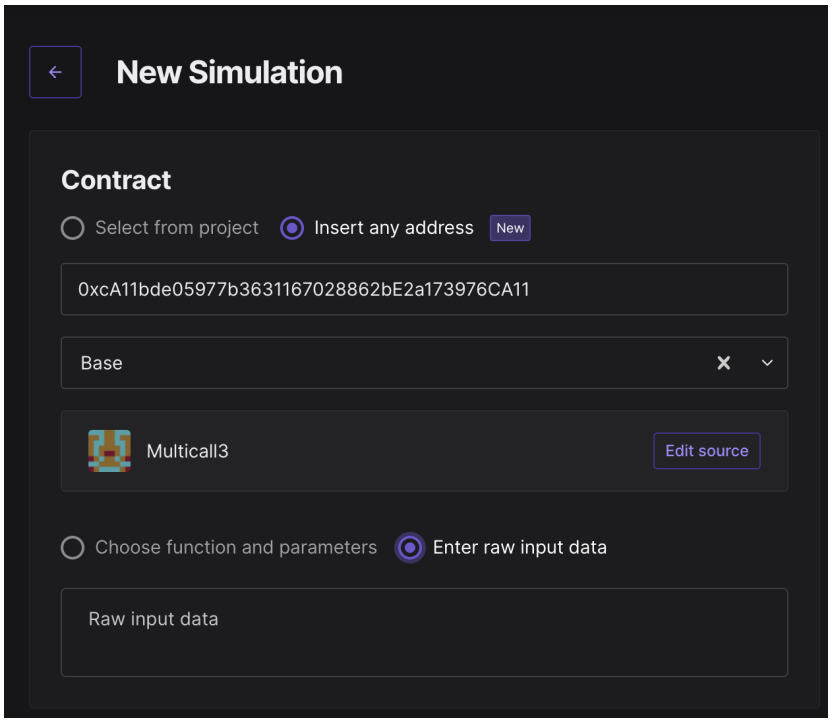
   ```
   cd mainnet/2023-08-15-support-eas
   ```

   Make sure your ledger is still unlocked and run the following (note that it might take a minute to install dependencies). Before actually signing with your ledger, go to next step.

   ```
   make sign-cb  # Or make sign-op for Optimism signers
   ```

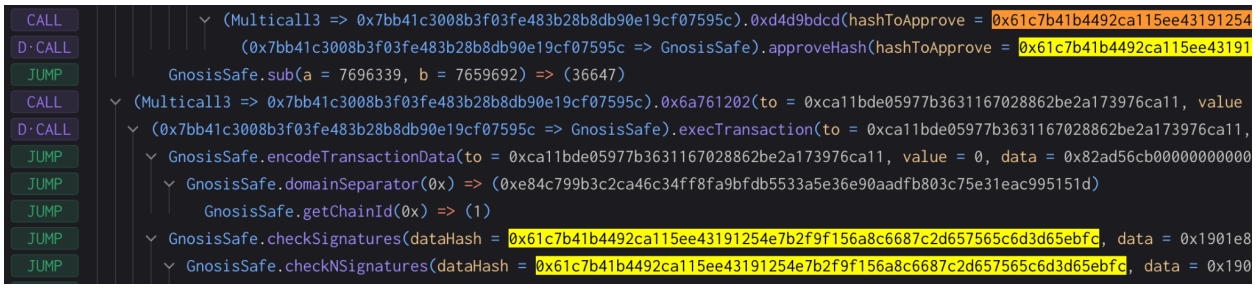
4. Validations:

   During signing, a Tenderly simulation link will be outputted. Paste this in your browser. You may see a message like this in your terminal - `Insert the following hex into the 'Raw input data' field`. If so, in Tenderly select the “Enter raw input data option” and enter the data there.

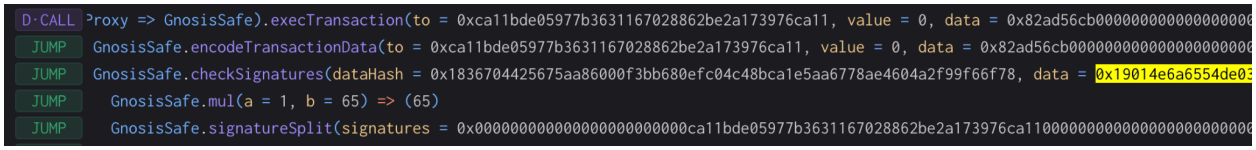


   Validate the following:

   a. The Nested hash: the value printed to the terminal matches the Tenderly `hashToApprove` under approveHash and the dataHash when checking the signatures for the actual tx execution. Example where hash is `0x61c74`…:



   b. The Data to sign: value printed to the terminal (and showed on the ledger screen) matches the data field when checking the signatures for the approveHash tx. Example where hash is `0x19014e`…:



c. The state changes include
    i. Upgrading 0x4200000000000000000000000000000000000020 to the new implementation [0x75505a97BD334E7BD3C476893285569C4136Fa0F]
    ii. Upgrading 0x4200000000000000000000000000000000000021 to the new implementation [0xbEb5Fc579115071764c7423A4f12eDde41f106Ed]

If all validations check out, sign the payload with your ledger.

5. Send output to Facilitator(s). Note: Nothing occurred onchain - these are offchain signatures which will be collected by Facilitators for execution. Execution can occur by anyone once a threshold of signatures are collected, so a Facilitator will do the final execution for convenience.

    Format should be something like this:

    ```
    Data:  <DATA>
    Signer: <ADDRESS>
    Signature: <SIGNATURE>
    ```

6. Congrats, you are done! 🔐