HYPERLEDGER

Linux Foundation

2021 HyperLedger Fabric Penetration Test

Tevora Threat Research Group

Delivered March 5, 2021

Table of Contents

Executive Summary

Purpose

The 2021 HyperLedger Fabric Penetration Test for Linux Foundation was conducted from 11 January, 2021 to 01 March, 2021 to help ensure Linux Foundation resources are secure from advanced threat actors.

Additional objectives for this penetration test were based on industry standard guidelines as follows:

- Identification of vulnerabilities so that they can be remediated prior to being exploited by an attacker
- Direct observation of restricted services or data in the absence of expected access controls
- Compromise of an intermediary device used by privileged users to access secure network zones
- Compromise of the domain used by privileged users
- Sensitive data leakage or exfiltration
- Verification of application logic, session handling, and API security for applications using supplied credentials
- Verification that only authorized services are exposed to the network perimeter
- Verification of network segmentation of non-privileged and privileged networks

Project Participation

The following Linux Foundation and Tevora personnel were directly involved in this assessment.

Sponsors	David Huseby, Linux Foundation
Participants	Ry Jones, Linux Foundation Kevin Dick, Tevora

Scope

This report contains the summary of project scope, findings, and recommendations resulting from the Application Penetration Test conducted by Tevora against the Linux Foundation environment.

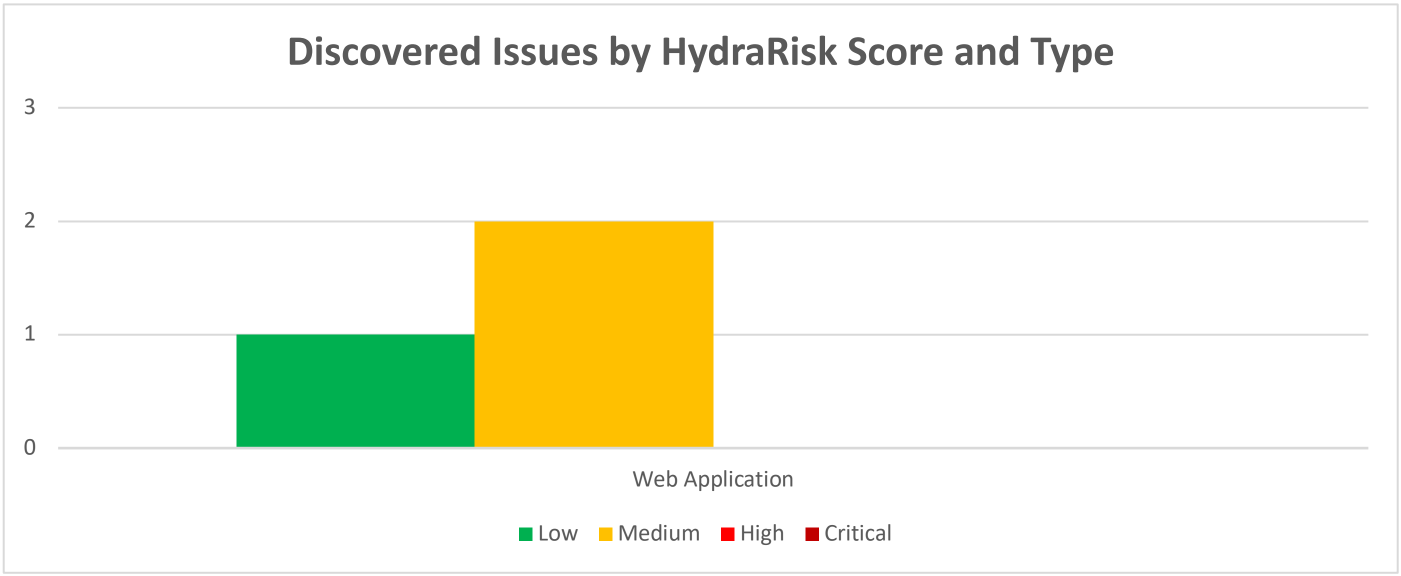
Application Penetration Test

The following items were considered in scope:

HyperLedger Fabric:

- https://github.com/hyperledger/fabric

Findings Overview



Application Penetration Test Results

Tevora performed penetration testing of the Hyperledger Fabric blockchain platform, analyzing the security of Peers, Orderers, Fabric CA, and their intercommunications. Dynamic and static analysis was performed including debugging of critical security code paths, such as identity checking and endorsement validation, as well as testing live test networks' resilience against potential attack scenarios. Tevora discovered some issues during testing, most notably the exposure of Go's PPROF profiling endpoint by Peer and Orderer services, however, no serious issues were found with the core security architecture of the platform nor with implementation of security-critical components.

Tevora focused much of the initial testing on validating Fabric's trust model with the default managed service provider (MSP), ensuring that authentication and authorization checks were enforced as expected. Tevora forged certificates and sent a variety of malformed requests to evaluate the security posture of authentication and authorization implementations. Tevora found that, in the default MSP, validation was performed as expected and was resilient to forgery attempts. This may be due in part to Fabric's use of standard mechanisms such as .x509 certificates and proven public/private trust to authenticate components. Tevora found the default MSP to be well designed, and the flexibility in MSP implementation a welcome feature.

Additionally, a number of potential attack scenarios were explored including compromised peers, transaction replay, unauthenticated denial of service, gossip replication exploitation, unauthorized chaincode installation, and endorsement spoofing. Tevora performed both experimental analysis and direct debugging of security critical code paths. Tevora performed experiments using modified peers and clients to launch attack scenarios then subsequently used instrumentation such as the Hyperledger Blockchain Explorer to analyze the results of the attempted attacks. Tevora also stepped through security-critical code paths in a debugger to analyze directly how validation checks are performed. In all cases, Tevora did not observe any deviation from expected behavior, nor were any attack paths apparent after analysis of the execution flow directly. Lastly, Tevora extended testing significantly beyond the planned timeframe to ensure good coverage of attack scenarios and an understanding of the validation behavior was achieved.

Tevora performed enumeration of and dynamic testing against exposed endpoints on Fabric components, including gRPC endpoints, Fabric CA REST APIs, and Peer health web endpoints. The majority of complexity was, as expected, in the gRPC endpoints, which were thoroughly tested. Other endpoints were found to present a minimal attack surface. The most critical of the discovered web services endpoints is the Fabric CA REST APIs. These were found to be resilient to attack and observed to implement recommended hardening such as restriction of failed login attempts to prevent password brute-forcing. No serious issues were observed here, and many of these endpoints can be restricted in production deployments, leaving only gRPC communications open to other organizations.

Tevora discovered Go's PPROF endpoint to be opened on 0.0.0.0:6060 by default for Peers and Orderer nodes, potentially exposing these components to denial of service or informational leakage risks. In many cases, production deployments will likely block access to these ports at the network layer, however, the service being exposed by default increases the risk of insecure deployments. In general, Tevora recommends hardening default configurations, with debug functionality being opt-in.

During testing, Tevora was alerted to a dependency confusion issue in the fabric-chaincode-node SDK reported via Hyperledger's bug bounty program. Tevora validated this issue was present, and performed a sweep of that node SDK, and other Fabric related SDKs for this or other issues. In-depth testing on all SDKs was not performed to the degree of the core fabric repo, however, Tevora found few issues in the sweep of the SDKs.

Overall, Tevora found Fabric to be a robust, well-architected, and similarly well-implemented platform. Fabric has reached a mature state of security and functionality and is well-positioned for updates should any flaws be discovered in the underlying industry-standard crypto used for the default MSP and validation, partly due to Fabric's modular and open-source nature. The issues discovered by Tevora all relate to increasing hardening of default configurations, in order to avoid insecure deployments of Fabric in a production environment. In general, Tevora anticipates the highest likelihood of attack targets to be production implementation errors such as weak passwords exposed debug interfaces, or trust misconfigurations. The Hyperledger Fabric team may want to add security guidelines and considerations into the "deploying a production network" documentation, because the example networks, although quite well documented and complete, have some insecure defaults, like setting weak administrative passwords.

Attack Map

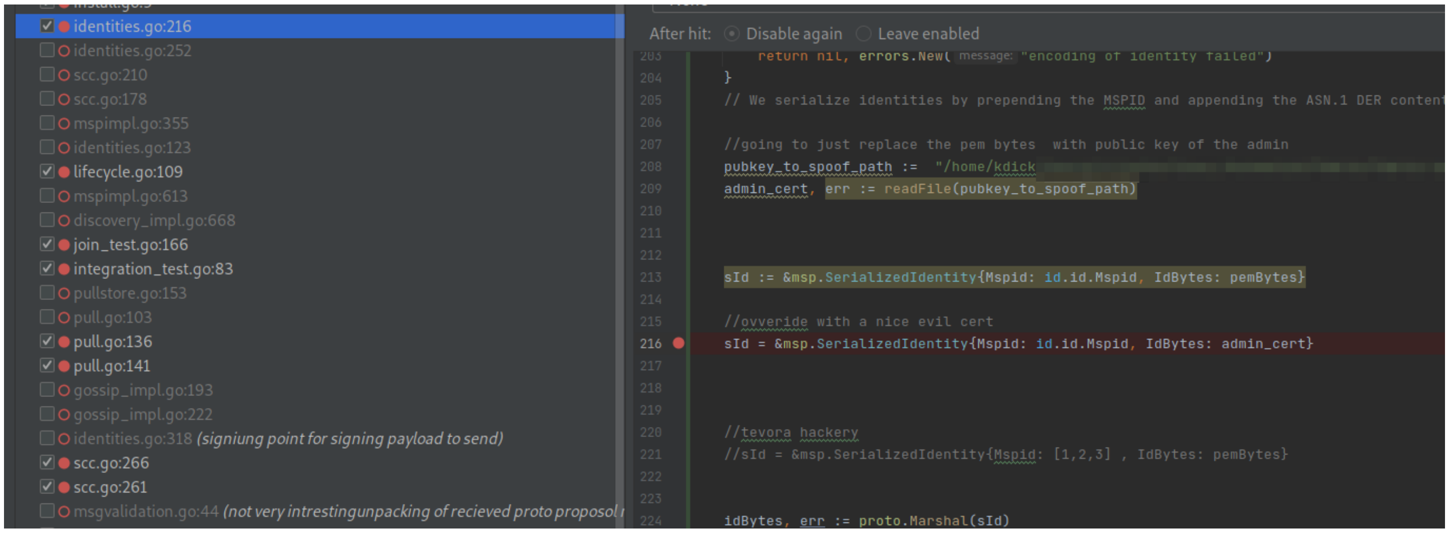
Tevora analyzed the footprint and expected deployments of the fabric platform to develop potential attack scenarios and test cases to evaluate these scenarios. Tevora did not discover major application logic flaws or design flaws outside of legacy (1.4) chaincode installation processes. Tevora performed many application logic tests on the fabric platform and has highlighted the attack scenarios below that Tevora believes to be most critical to validating Fabric's security model. In all cases, Tevora found Fabric to be resilient to attack and spoofing attempts.

Identify Spoofing

A key tentpole of Fabric is trust, with all subsequent security requiring the trust mechanisms to be correctly implemented. Unlike public blockchain systems using proof of work to establish trust, Fabric relies on being able to identify and authenticate organizations and their components. Because of this, a large amount of testing was focused on fabric Membership Service Providers (MSPs) and how components authenticate to each other.

Tevora analyzed how peers communicate and authenticate with certificates, stepping through the default certificate-based MSP provider to ensure identification and validation of clients is done correctly. Tevora both created spoofed certificate tests cases, and stepped through the MSP validation code in a debugger to analyze behavior for areas of potential weakness. Tevora used modified peers to both test how data is handled by interesting functions directly, and to modify Peer CLI commands to send test cases to Peer servers.

Example snippet of modified peer used as a scratchpad to test the handling of unexpected cert data in various stages of the validation pipeline

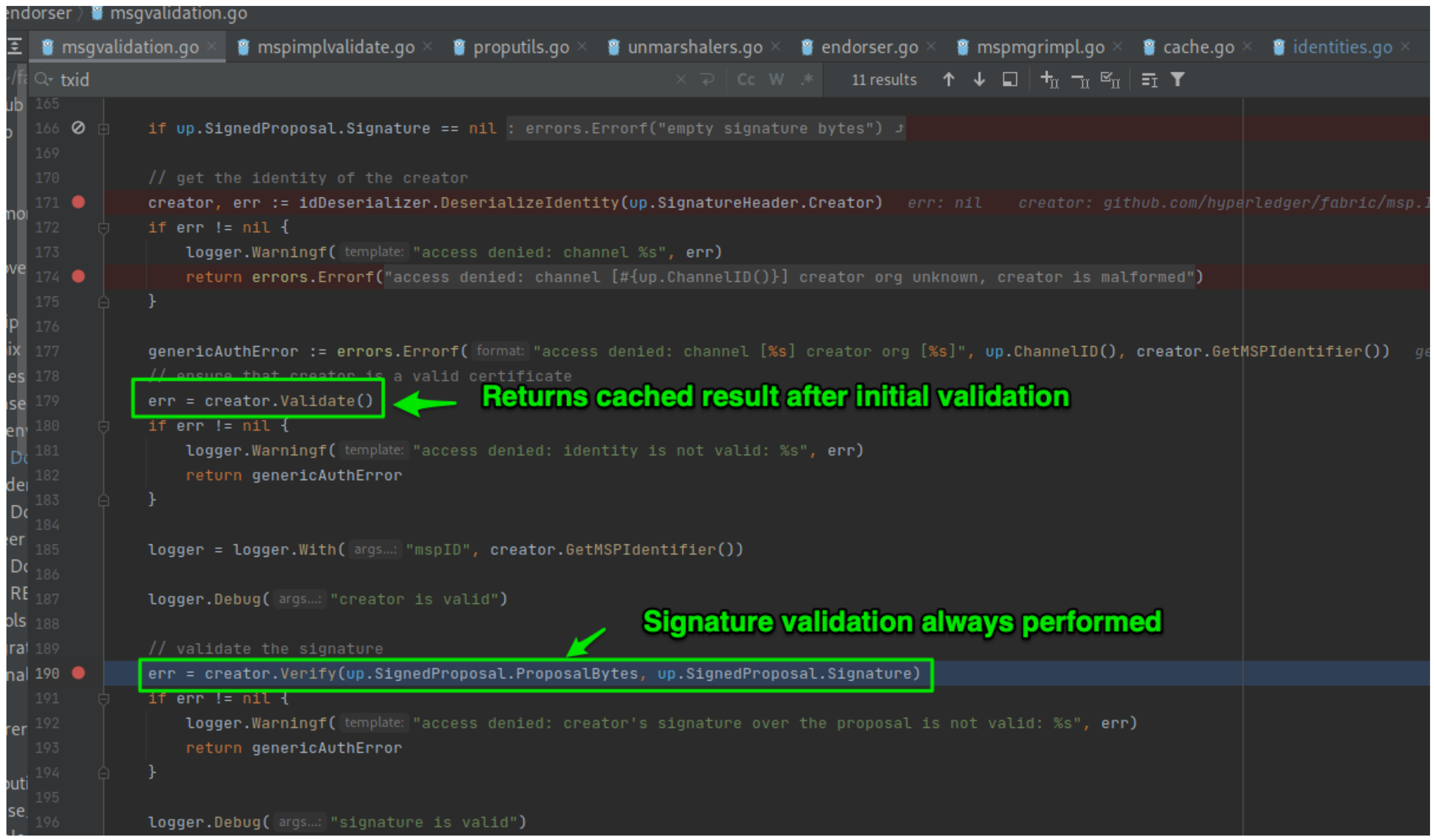


✓ ● identities.go:216
☐ ○ identities.go:252
☐ ○ scc.go:210
☐ ○ scc.go:178
☐ ○ mspimpl.go:355
☐ ○ identities.go:123
✓ ● lifecycle.go:109
☐ ○ mspimpl.go:613
☐ ○ discovery_impl.go:668
✓ ● join_test.go:166
✓ ● integration_test.go:83
☐ ○ pullstore.go:153
☐ ○ pull.go:103
✓ ● pull.go:136
✓ ● pull.go:141
☐ ○ gossip_impl.go:193
☐ ○ gossip_impl.go:222
☐ ○ identities.go:318 *(signiung point for signing payload to send)*
✓ ● scc.go:266
✓ ● scc.go:261
☐ ○ msgvalidation.go:44 *(not very intrestingunpacking of recieved proto proposol*

After hit: [✓ Disable again] Leave enabled

```
        return nil, errors.New( message: "encoding of identity failed")
    }
    // We serialize identities by prepending the MSPID and appending the ASN.1 DER conten

    //going to just replace the pem bytes  with public key of the admin
    pubkey_to_spoof_path :=  "/home/kdick[redacted]
    admin_cert, err := readFile(pubkey_to_spoof_path)



    sId := &msp.SerializedIdentity{Mspid: id.id.Mspid, IdBytes: pemBytes}

    //ovveride with a nice evil cert
    sId = &msp.SerializedIdentity{Mspid: id.id.Mspid, IdBytes: admin_cert}



    //tevora hackery
    //sId = &msp.SerializedIdentity{Mspid: [1,2,3] , IdBytes: pemBytes}


    idBytes, err := proto.Marshal(sId)
```

Tevora noted that some caching is performed of MSP cert validation, presumably to optimize performance. Tevora evaluated this to ensure a spoofed certificate could not be sent after an initial cache, but found that signatures are always validated. The cache key is a SHA digest of the entire raw public cert; any modification to the certificate will cause a cache invalidation, and the certificate is always used to validate the signature. There were no issues identified with caching.



```
    if up.SignedProposal.Signature == nil : errors.Errorf("empty signature bytes") ⮐

    // get the identity of the creator
    creator, err := idDeserializer.DeserializeIdentity(up.SignatureHeader.Creator)   err: nil    creator: github.com/hyperledger/fabric/msp.
    if err != nil {
        logger.Warningf( template: "access denied: channel %s", err)
        return errors.Errorf("access denied: channel [#{up.ChannelID()}] creator org unknown, creator is malformed")
    }

    genericAuthError := errors.Errorf( format: "access denied: channel [%s] creator org [%s]", up.ChannelID(), creator.GetMSPIdentifier())
    // ensure that creator is a valid certificate
    err = creator.Validate()
    if err != nil {
        logger.Warningf( template: "access denied: identity is not valid: %s", err)
        return genericAuthError
    }

    logger = logger.With( args...: "mspID", creator.GetMSPIdentifier())

    logger.Debug( args...: "creator is valid")

    // validate the signature
    err = creator.Verify(up.SignedProposal.ProposalBytes, up.SignedProposal.Signature)
    if err != nil {
        logger.Warningf( template: "access denied: creator's signature over the proposal is not valid: %s", err)
        return genericAuthError
    }

    logger.Debug( args...: "signature is valid")
```

No other notable leads were identified while stepping through the code flow, and the validation process remained resilient to sanity test cases and experiments with forged certificates, modified proposals, and modified headers. The use of industry-standard public/private key cryptography as well as a concise and effective validation flow benefit Fabric and provide no observed avenues of attack other than potential vulnerabilities in the underlying cryptography used.

In the case of RSA, ECDSA, or ED25519 implementations being broken (or underlying algorithm being broken i.e. in RSA's case an algorithm for fast factoring primes), Hyperledger would be far from the only platform to be affected by this, and new solutions would likely be easy to plug in, most likely with a simple update to Go's x509 library. Similarly, vulnerable methods can be deprecated and made unsupported.

Additionally, Tevora observed a flexible MSP interface that allows other implementations to be developed relatively seamlessly. Although this could lead to untested and unsecured deployments with vulnerable MSP implementations, the most Hyperledger may want to do here is warn users of the diligence required when rolling their own MSP.

Lastly, Fabric also makes optional use of TLS wrappers for connections between components. This extra layer of security encrypts gRPC traffic in transit and mitigates the risk of any potential future man in the middle (MiTM) attacks against the MSP itself since the communication occurs within the protection of the TLS tunnel.

Malicious ChainCode Install

Because chaincode, by design, executes defined code on targeted peers (typically in a Docker container), Tevora wanted to ensure that unauthenticated clients or malicious evil peers in a network could not deploy malicious chain code to the network. Any vectors for code execution are a valuable target for attackers, so it is important that these chaincodes are only deployable with proper administrator authorization.

The 1.4 chaincode lifecycle does allow peers from other organizations to replicate potentially malicious chaincode across org boundaries. Other security measures, such as isolation of executed chaincode to docker containers, are in place, however, this could lead to denial of service or network based attacks from the compromised container. Fabric is already aware of these risks, as evidenced by documentation and public presentations (https://www.youtube.com/watch?v=XvEMDScFU2M&t=1407s). Tevora recommends migrating to 2.0 chaincode lifecycles for existing implementations.

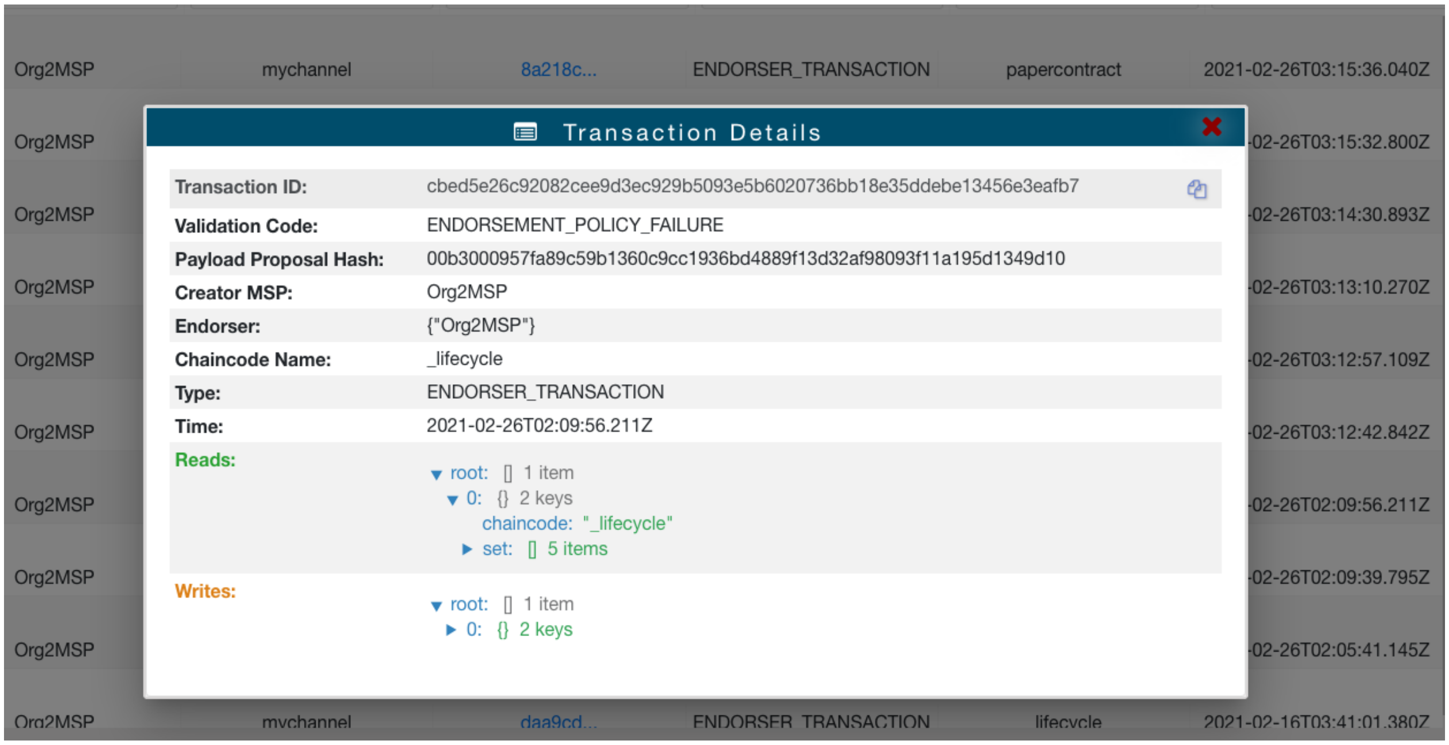
The 2.0 model of chaincode installation was found to be designed with both security and scalability in mind, requiring organization admins to install specific code bundles to peers and not replicating the code packages across peers. In addition to the major security improvements, 2.0 chaincode lifecycle allows different chaincode to be implemented for the same endorsed interface greatly increasing the flexibility of chaincode operations by allowing per organization customizations. Tevora did not identify any ways to bypass authentication to install chaincode to peers without having the requisite admin access, nor was chaincode observed to be replicated as in the 1.4 protocol.

Endorsement Spoofing

Tevora validated Fabric's distributed endorsement validation architecture by simulating malicious peers and client applications modified to attempt to commit and replicate transactions that have failed the approved endorsement policy. Tevora used the default majority endorsement policy for these tests, modifying evil peers and clients to propose and commit to the ledger transactions not meeting the endorsement policy.

Tevora observed that the unmodified peers successfully enforced their approved endorsement policy on chaincode execution and replication.

Example attempted _lifecycle chaincode endorsement replication attempt result

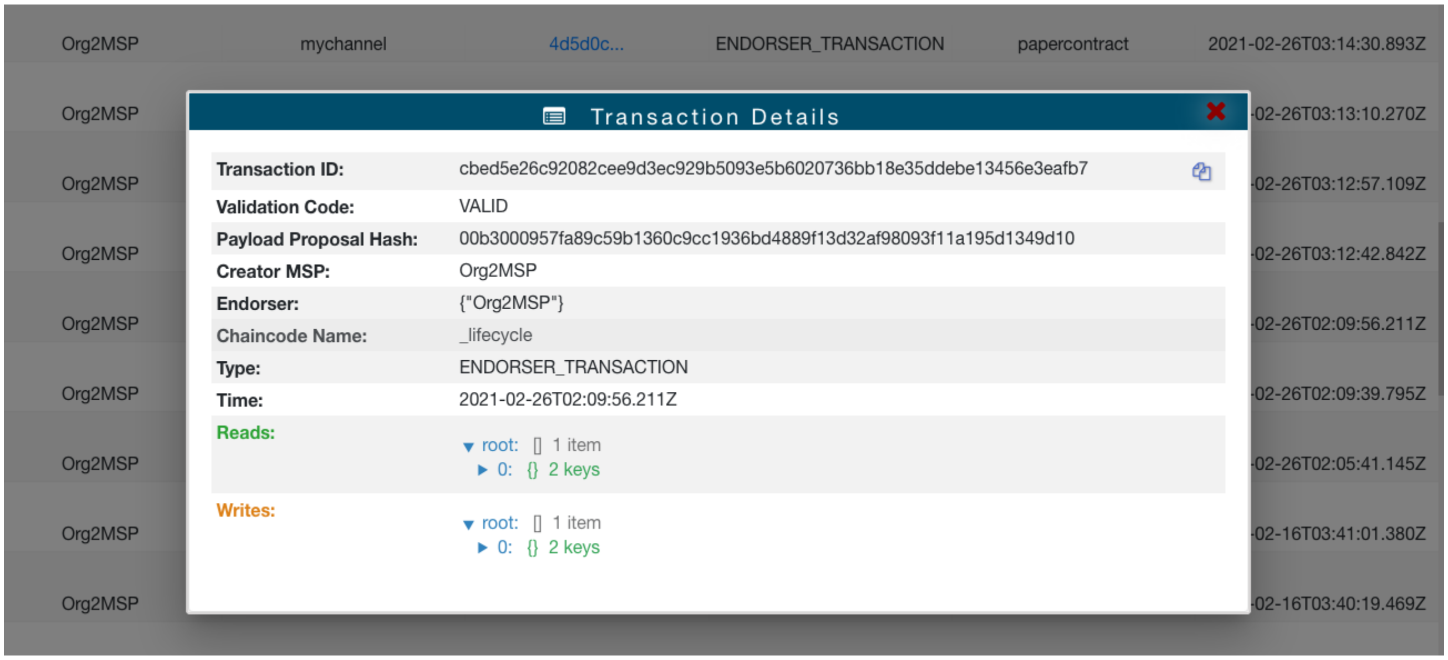


Transaction Details

Transaction ID:	cbed5e26c92082cee9d3ec929b5093e5b6020736bb18e35ddebe13456e3eafb7
Validation Code:	ENDORSEMENT_POLICY_FAILURE
Payload Proposal Hash:	00b3000957fa89c59b1360c9cc1936bd4889f13d32af98093f11a195d1349d10
Creator MSP:	Org2MSP
Endorser:	{"Org2MSP"}
Chaincode Name:	_lifecycle
Type:	ENDORSER_TRANSACTION
Time:	2021-02-26T02:09:56.211Z

```
Reads:
  ▼ root: [] 1 item
    ▼ 0: {} 2 keys
        chaincode: "_lifecycle"
      ▶ set: [] 5 items
Writes:
  ▼ root: [] 1 item
    ▶ 0: {} 2 keys
```

Same transaction from "malicious" Peer's ledger



Transaction Details

Transaction ID:	cbed5e26c92082cee9d3ec929b5093e5b6020736bb18e35ddebe13456e3eafb7
Validation Code:	VALID
Payload Proposal Hash:	00b3000957fa89c59b1360c9cc1936bd4889f13d32af98093f11a195d1349d10
Creator MSP:	Org2MSP
Endorser:	{"Org2MSP"}
Chaincode Name:	_lifecycle
Type:	ENDORSER_TRANSACTION
Time:	2021-02-26T02:09:56.211Z

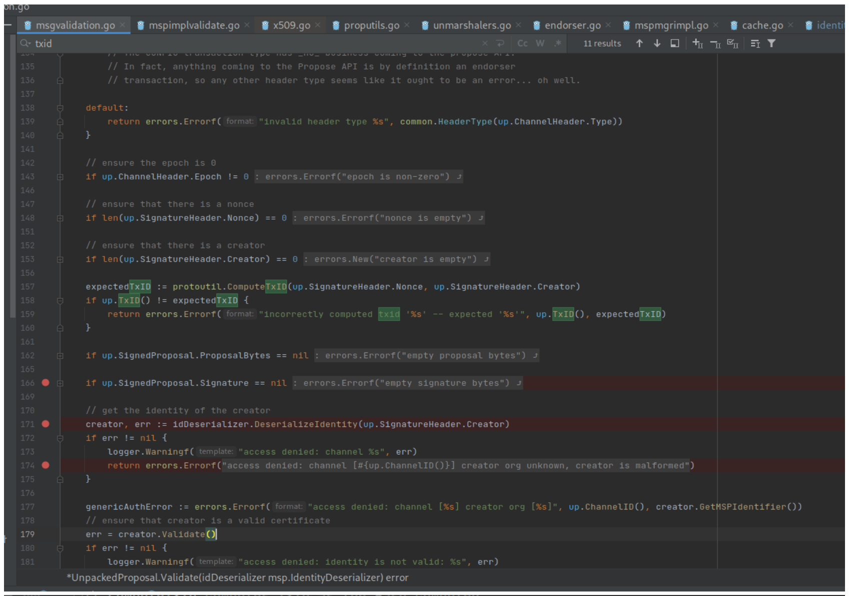
```
Reads:
  ▼ root: [] 1 item
    ▶ 0: {} 2 keys
Writes:
  ▼ root: [] 1 item
    ▶ 0: {} 2 keys
```

Transaction Replay

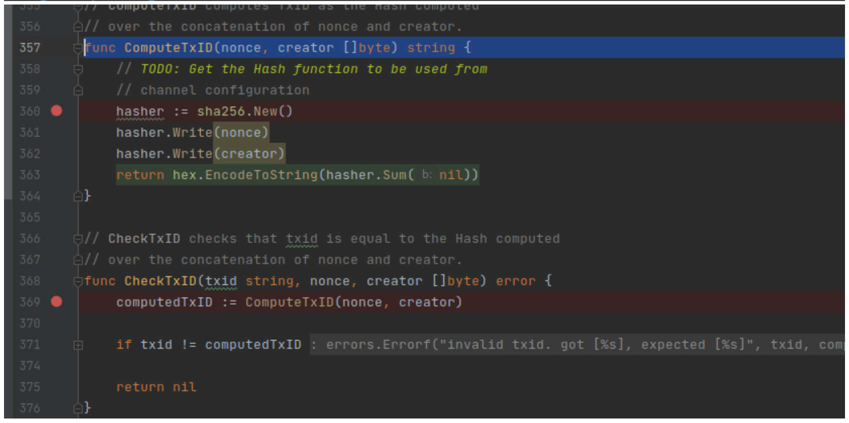
Tevora reviewed the Peer endorsement lifecycle architecture, observed transaction processing with Transaction ID validation, and tested with repeated messages to validate the expected behavior of a Fabric deployment. As designed, duplicate transactions cannot be currently replayed in the tested Fabric 2.3 test-network.

Analyzing TXID (Transaction ID) generation and validation

Tevora found that TXIDs are made using a client-generated nonce, which is concatenated with the creator (essentially public key for the authenticated client) and hashed. Both the nonce and creator info is sent in the authenticated payload contents to Peers, allowing peers to confirm the correct transaction ID.

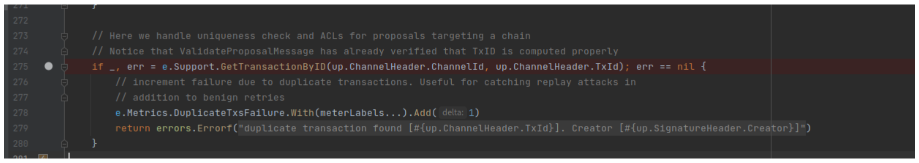


```
msgvalidation.go  mspimplvalidate.go  x509.go  proputils.go  unmarshalers.go  endorser.go  mspmgrimpl.go  cache.go
txid                                                        11 results
        // In fact, anything coming to the Propose API is by definition an endorser
        // transaction, so any other header type seems like it ought to be an error... oh well.

    default:
        return errors.Errorf( format: "invalid header type %s", common.HeaderType(up.ChannelHeader.Type))
    }

    // ensure the epoch is 0
    if up.ChannelHeader.Epoch != 0 : errors.Errorf("epoch is non-zero") ↵

    // ensure that there is a nonce
    if len(up.SignatureHeader.Nonce) == 0 : errors.Errorf("nonce is empty") ↵

    // ensure that there is a creator
    if len(up.SignatureHeader.Creator) == 0 : errors.New("creator is empty") ↵

    expectedTxID := protoutil.ComputeTxID(up.SignatureHeader.Nonce, up.SignatureHeader.Creator)
    if up.TxID() != expectedTxID {
        return errors.Errorf( format: "incorrectly computed txid '%s' -- expected '%s'", up.TxID(), expectedTxID)
    }

    if up.SignedProposal.ProposalBytes == nil : errors.Errorf("empty proposal bytes") ↵

    if up.SignedProposal.Signature == nil : errors.Errorf("empty signature bytes") ↵

    // get the identity of the creator
    creator, err := idDeserializer.DeserializeIdentity(up.SignatureHeader.Creator)
    if err != nil {
        logger.Warningf( template: "access denied: channel %s", err)
        return errors.Errorf("access denied: channel [#{up.ChannelID()}] creator org unknown, creator is malformed")
    }

    genericAuthError := errors.Errorf( format: "access denied: channel [%s] creator org [%s]", up.ChannelID(), creator.GetMSPIdentifier())
    // ensure that creator is a valid certificate
    err = creator.Validate()
    if err != nil {
        logger.Warningf( template: "access denied: identity is not valid: %s", err)
*UnpackedProposal.Validate(idDeserializer msp.IdentityDeserializer) error
```



```
// over the concatenation of nonce and creator.
func ComputeTxID(nonce, creator []byte) string {
    // TODO: Get the Hash function to be used from
    // channel configuration
    hasher := sha256.New()
    hasher.Write(nonce)
    hasher.Write(creator)
    return hex.EncodeToString(hasher.Sum( b: nil))
}

// CheckTxID checks that txid is equal to the Hash computed
// over the concatenation of nonce and creator.
func CheckTxID(txid string, nonce, creator []byte) error {
    computedTxID := ComputeTxID(nonce, creator)

    if txid != computedTxID : errors.Errorf("invalid txid. got [%s], expected [%s]", txid, com

    return nil
}
```

Validated duplicate check in core/endorser/endorser.go



```
    // Here we handle uniqueness check and ACLs for proposals targeting a chain
    // Notice that ValidateProposalMessage has already verified that TxID is computed properly
    if _, err = e.Support.GetTransactionByID(up.ChannelHeader.ChannelId, up.ChannelHeader.TxId); err == nil {
        // increment failure due to duplicate transactions. Useful for catching replay attacks in
        // addition to benign retries
        e.Metrics.DuplicateTxsFailure.With(meterLabels...).Add( delta: 1)
        return errors.Errorf("duplicate transaction found [#{up.ChannelHeader.TxId}]. Creator [#{up.SignatureHeader.Creator}]")
    }
```

Technical Summary

Summary of Findings

Total Penetration Test Findings 3

Application Findings	**Status**	**HydraRisk**	
APP-01296 Go Profiling Endpoint Exposed	Discovered	**13**	**Medium**
APP-01302 Dependency Confusion	Discovered	**12**	**Medium**
APP-01310 Fabric CA Bootstrapped with Admin Password in Cleartext on Commandline	Discovered	**6**	**Low**

Technical Findings

APP-01296 Go Profiling Endpoint Exposed

Description

While reviewing Fabric's source files, Tevora discovered imports of "net/http/pprof". When this item is imported, it opens up a debug server on port 6060, which an attacker can use to enumerate:

- Function names and file paths
- Profiling data
- Denial of Service risk via profiling load

Tevora observed this port being spun up inside the sample test-net docker containers (although not forwarded outside the docker net) as well as when running peers or orderers locally.

A hyperledger fabric administrator may inadvertently leave this port exposed in production if they are not familiar with the risks of this interface.

Status	Discovered	**CVSS Base Score**	7.3	**High**	**HydraRisk**	13	**Medium**
					Consequence	4	
					Probability	3	
					Velocity	2	
					Criticality	3	
					Responsiveness	1	

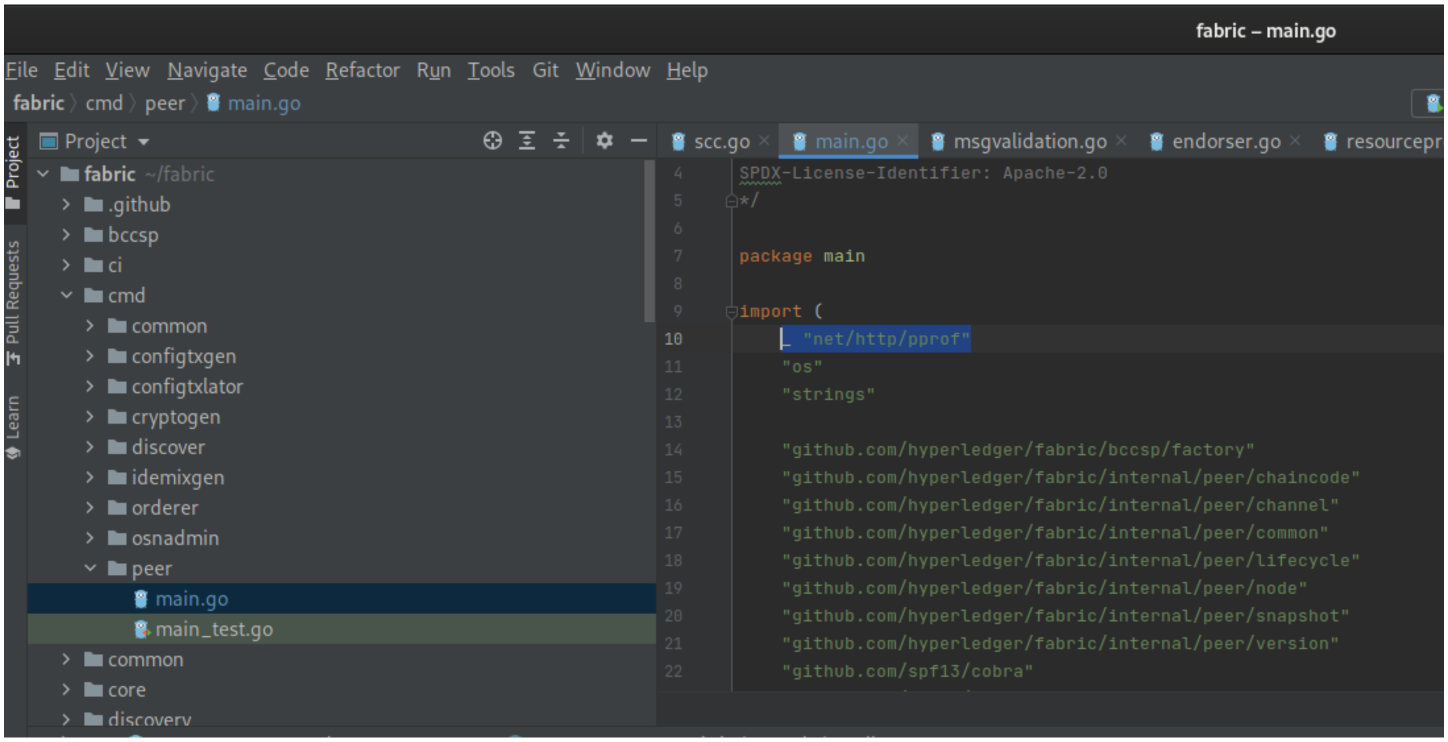
Affected Systems

The following systems have been identified and are affected:

Repo	**File**
https://github.com/hyperledger/fabric-ca	fabric-ca/lib/server.go
https://github.com/hyperledger/fabric	fabric/cmd/peer/main.go
https://github.com/hyperledger/fabric	fabric/orderer/common/server/main.go

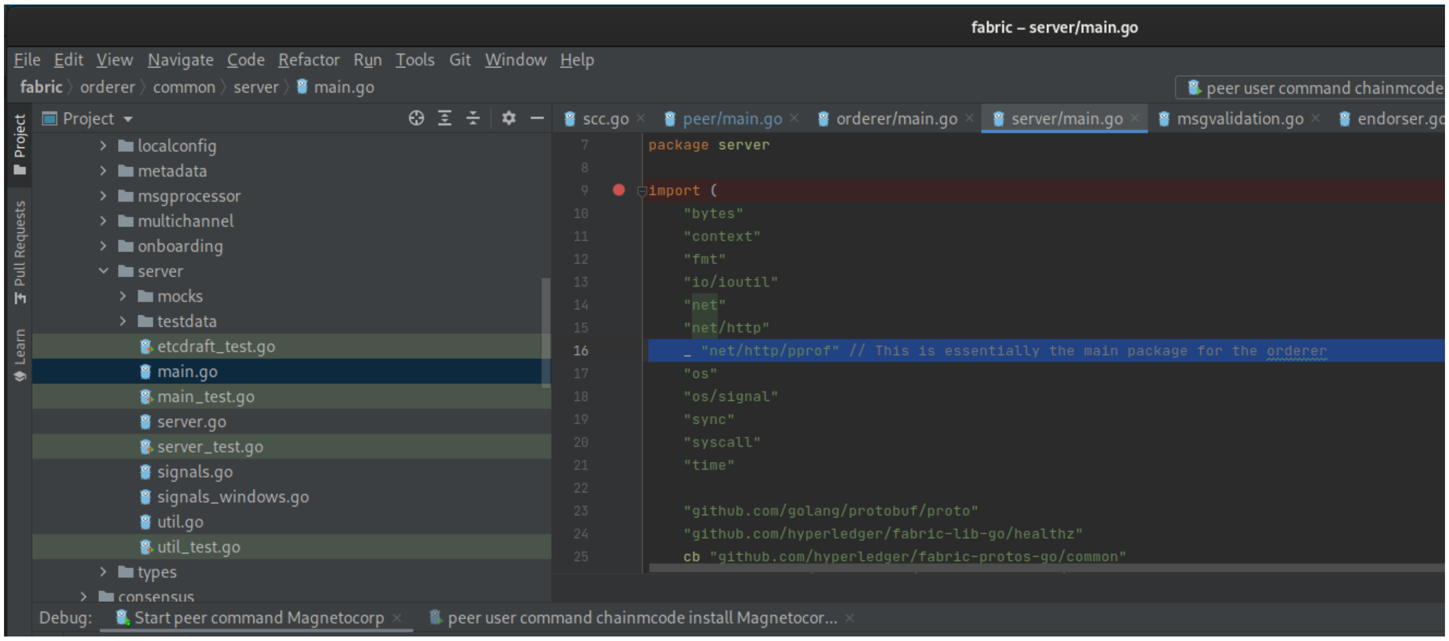
Details

Import statement enabling pprof debug interface on peer main.go



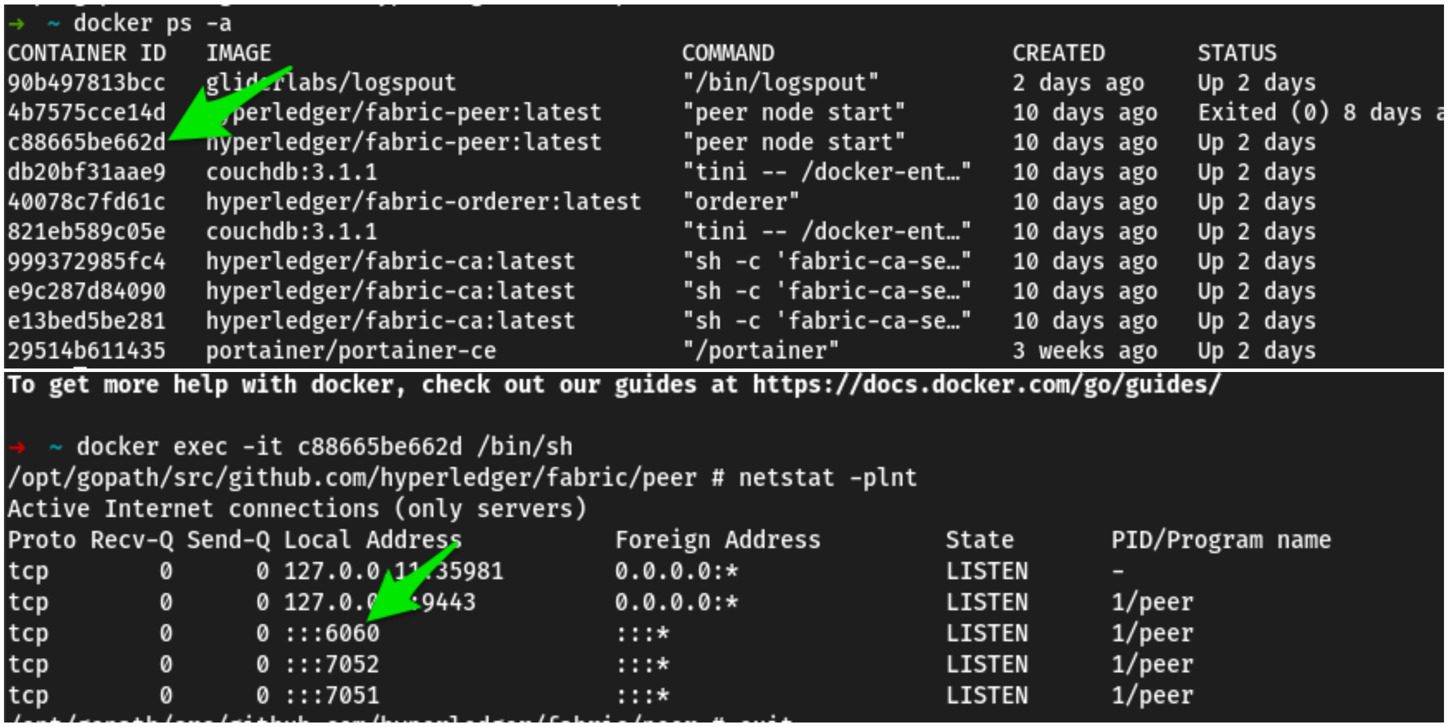
```go
SPDX-License-Identifier: Apache-2.0
*/

package main

import (
	_ "net/http/pprof"
	"os"
	"strings"

	"github.com/hyperledger/fabric/bccsp/factory"
	"github.com/hyperledger/fabric/internal/peer/chaincode"
	"github.com/hyperledger/fabric/internal/peer/channel"
	"github.com/hyperledger/fabric/internal/peer/common"
	"github.com/hyperledger/fabric/internal/peer/lifecycle"
	"github.com/hyperledger/fabric/internal/peer/node"
	"github.com/hyperledger/fabric/internal/peer/snapshot"
	"github.com/hyperledger/fabric/internal/peer/version"
	"github.com/spf13/cobra"
```

Import statement enabling pprof debug interface on orderer server main.go



```go
package server

import (
	"bytes"
	"context"
	"fmt"
	"io/ioutil"
	"net"
	"net/http"
	_ "net/http/pprof" // This is essentially the main package for the orderer
	"os"
	"os/signal"
	"sync"
	"syscall"
	"time"

	"github.com/golang/protobuf/proto"
	"github.com/hyperledger/fabric-lib-go/healthz"
	cb "github.com/hyperledger/fabric-protos-go/common"
```

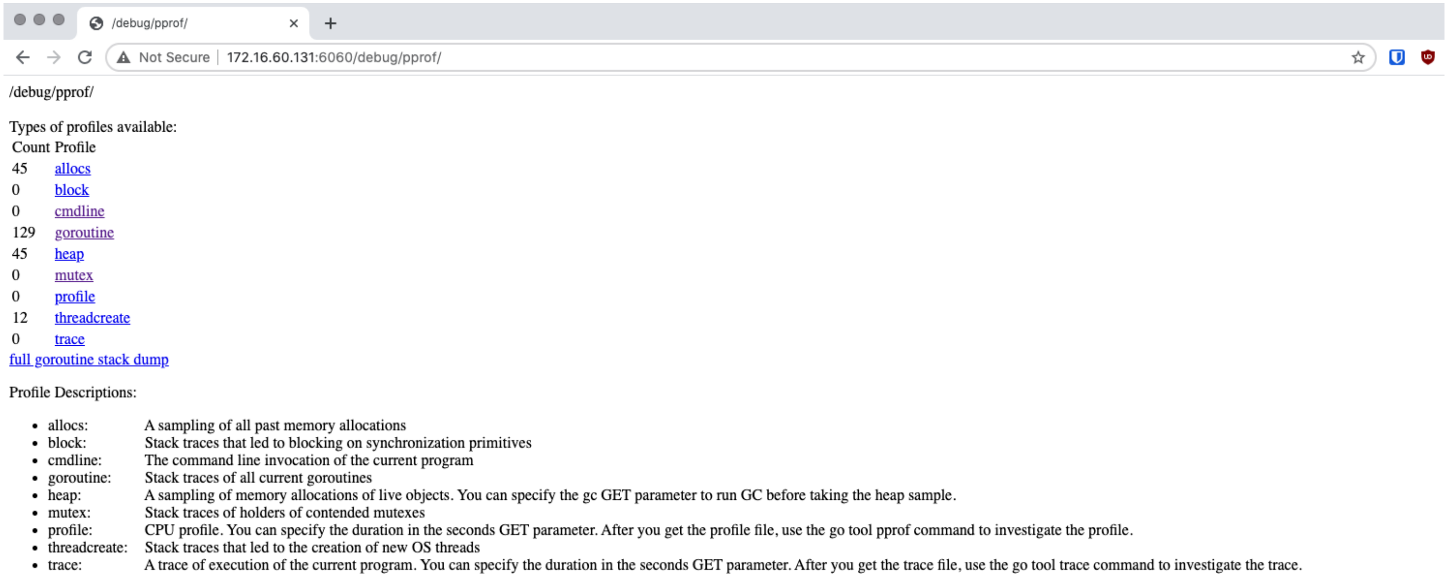
Viewing pprof listener on Org1 sample peer container



```
→ ~ docker ps -a
CONTAINER ID   IMAGE                               COMMAND                  CREATED        STATUS
90b497813bcc   gliderlabs/logspout                 "/bin/logspout"          2 days ago     Up 2 days
4b7575cce14d   hyperledger/fabric-peer:latest      "peer node start"        10 days ago    Exited (0) 8 days a
c88665be662d   hyperledger/fabric-peer:latest      "peer node start"        10 days ago    Up 2 days
db20bf31aae9   couchdb:3.1.1                       "tini -- /docker-ent…"   10 days ago    Up 2 days
40078c7fd61c   hyperledger/fabric-orderer:latest   "orderer"                10 days ago    Up 2 days
821eb589c05e   couchdb:3.1.1                       "tini -- /docker-ent…"   10 days ago    Up 2 days
999372985fc4   hyperledger/fabric-ca:latest        "sh -c 'fabric-ca-se…"   10 days ago    Up 2 days
e9c287d84090   hyperledger/fabric-ca:latest        "sh -c 'fabric-ca-se…"   10 days ago    Up 2 days
e13bed5be281   hyperledger/fabric-ca:latest        "sh -c 'fabric-ca-se…"   10 days ago    Up 2 days
29514b611435   portainer/portainer-ce              "/portainer"             3 weeks ago    Up 2 days
```

```
To get more help with docker, check out our guides at https://docs.docker.com/go/guides/

→ ~ docker exec -it c88665be662d /bin/sh
/opt/gopath/src/github.com/hyperledger/fabric/peer # netstat -plnt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.11:35981        0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:9443          0.0.0.0:*               LISTEN      1/peer
tcp        0      0 :::6060                 :::*                    LISTEN      1/peer
tcp        0      0 :::7052                 :::*                    LISTEN      1/peer
tcp        0      0 :::7051                 :::*                    LISTEN      1/peer
```

Accessing pprof debug interface on local (non dockerized) peer from another system on the same lan



Not Secure | 172.16.60.131:6060/debug/pprof/

/debug/pprof/

Types of profiles available:

Count	Profile
45	allocs
0	block
0	cmdline
129	goroutine
45	heap
0	mutex
0	profile
12	threadcreate
0	trace

full goroutine stack dump

Profile Descriptions:

- allocs: A sampling of all past memory allocations
- block: Stack traces that led to blocking on synchronization primitives
- cmdline: The command line invocation of the current program
- goroutine: Stack traces of all current goroutines
- heap: A sampling of memory allocations of live objects. You can specify the gc GET parameter to run GC before taking the heap sample.
- mutex: Stack traces of holders of contended mutexes
- profile: CPU profile. You can specify the duration in the seconds GET parameter. After you get the profile file, use the go tool pprof command to investigate the profile.
- threadcreate: Stack traces that led to the creation of new OS threads
- trace: A trace of execution of the current program. You can specify the duration in the seconds GET parameter. After you get the trace file, use the go tool trace command to investigate the trace.

References

- Your pprof is showing
  - https://mmcloughlin.com/posts/your-pprof-is-showing

- pprof exposed on kubelet healthz port
    - https://github.com/kubernetes/kubernetes/issues/81023

Recommendations

In general, Tevora recommends default configurations be hardened, with debug interfaces or other potentially dangerous functionality being opt-in instead of on by default.

The pprof server import should be removed, and pprof loads to be done only when opted by the administrator, perhaps via an environment variable or config.yaml specification. Additionally, Tevora recommends the pprof server be opened on localhost only by default unless otherwise specified.

APP-01302 Dependency Confusion

Description

During testing, another party discovered and disclosed library imports potentially affected by dependency confusion in the Node chaincode SDKs. An attacker can register the package names in public repositories, which will take precedence over the intended local package files. This could allow code execution on a system which installs the affected SDKs. Another dependency issue was also detected by Tevora, however it is not exploitable due to NPM similar name rules preventing the public package from being registered.

Status	Discovered	**CVSS Base Score**	7.1	**High**	**HydraRisk**	12	**Medium**
					Consequence	4	
					Probability	2	
					Velocity	3	
					Criticality	2	
					Responsiveness	1	

Affected Systems

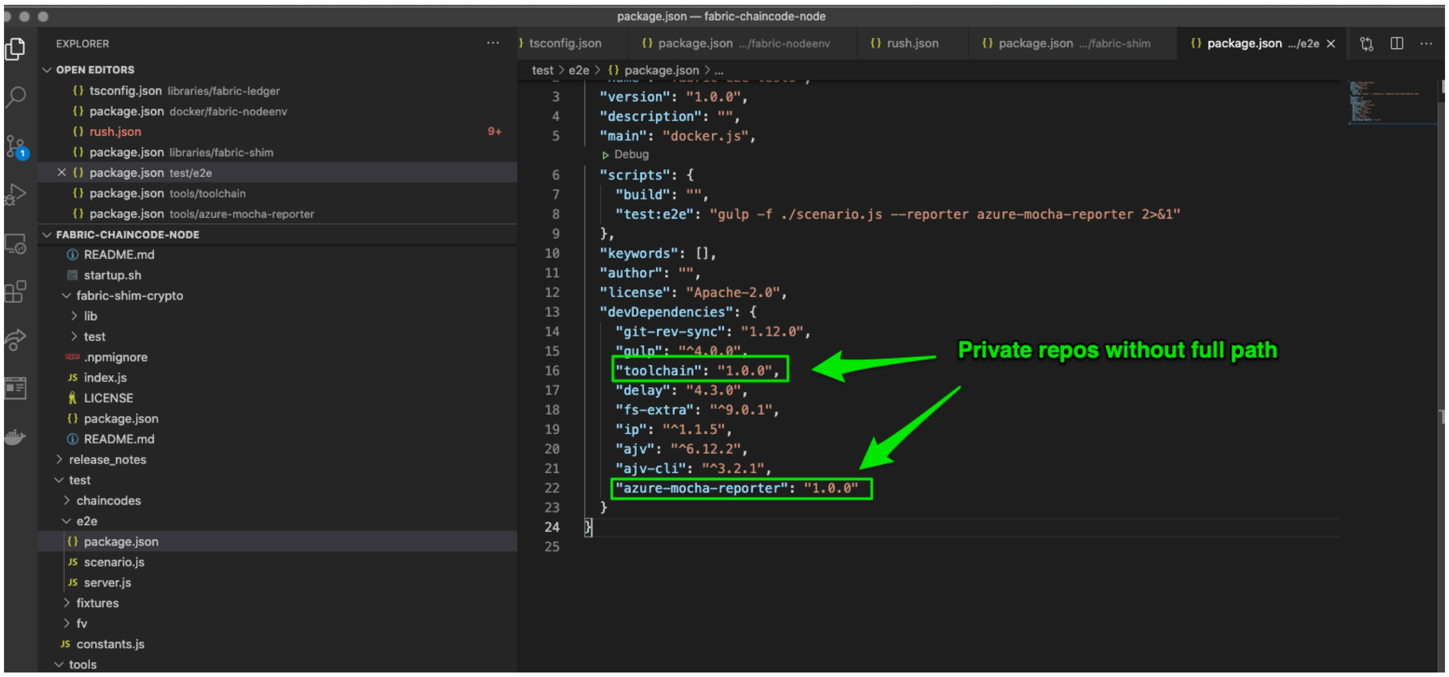
The following systems have been identified and are affected:

Repo	**Package Type**	**Vulnerable Dependency**	**File**
https://github.com/hyperledger/fabric-chaincode-node	NPM	azure-mocha-reporter	libraries/fabric-ledger/package.json libraries/fabric-shim/package.json libraries/fabric-shim-crypto/package.json test/e2epackage.json test/fv/package.json
https://github.com/hyperledger/fabric-chaincode-node	NPM	toolchain	docker/fabric-nodenv/package.json test/e2epackage.json

Details

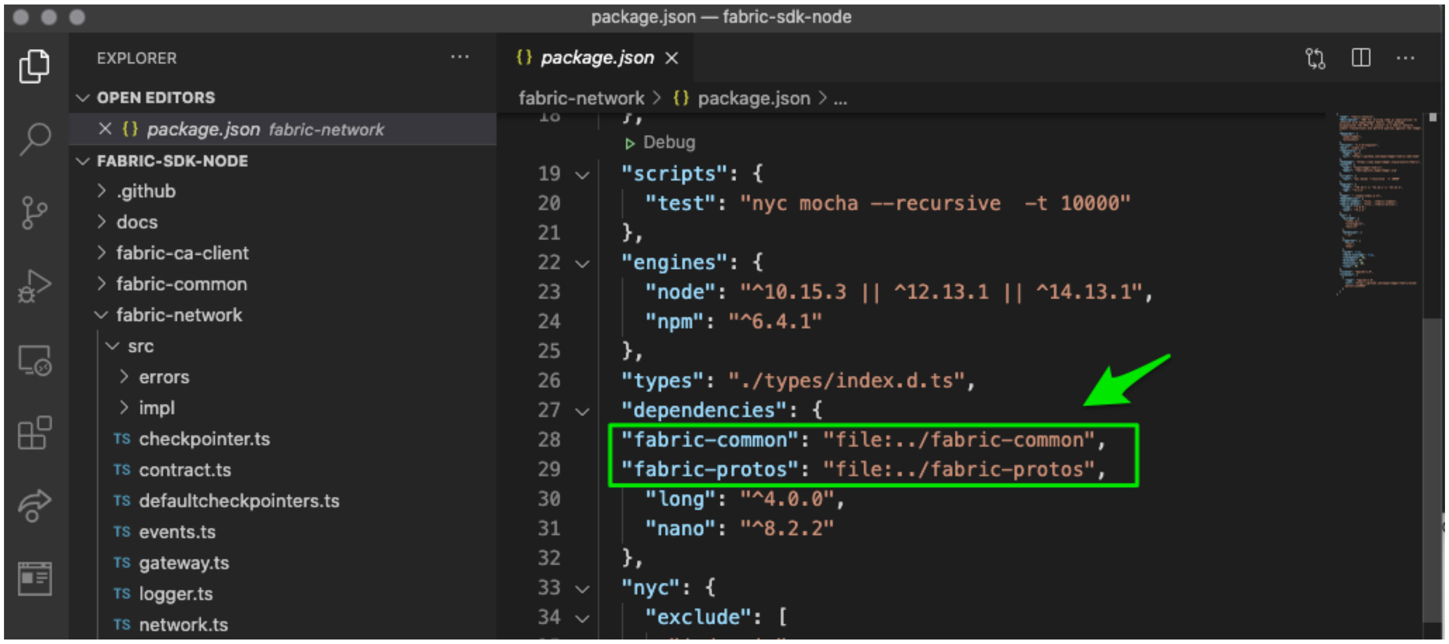
Ticket for ‘azure-mocha-reporter is vulnerable to package confusion’

https://jira.hyperledger.org/browse/FABCN-441‘toolchain’ and ‘azure-mocha-reporter’ local repositories imported without full path in hyperledger/fabric-chaincode-node/test/e2e/package.json


package.json — fabric-chaincode-node
EXPLORER
OPEN EDITORS
FABRIC-CHAINCODE-NODE
Private repos without full path

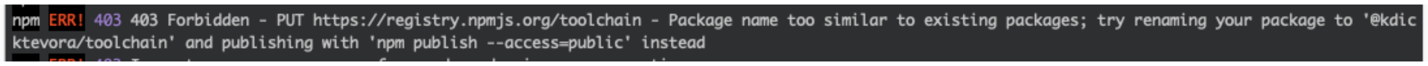
```
  "version": "1.0.0",
  "description": "",
  "main": "docker.js",
  "scripts": {
    "build": "",
    "test:e2e": "gulp -f ./scenario.js --reporter azure-mocha-reporter 2>&1"
  },
  "keywords": [],
  "author": "",
  "license": "Apache-2.0",
  "devDependencies": {
    "git-rev-sync": "1.12.0",
    "gulp": "^4.0.0",
    "toolchain": "1.0.0",
    "delay": "4.3.0",
    "fs-extra": "^9.0.1",
    "ip": "^1.1.5",
    "ajv": "^6.12.2",
    "ajv-cli": "^3.2.1",
    "azure-mocha-reporter": "1.0.0"
  }
}
```

Example correct explicit import of local dependencies in fabric-sdk-node/fabric-network



```
  "scripts": {
    "test": "nyc mocha --recursive  -t 10000"
  },
  "engines": {
    "node": "^10.15.3 || ^12.13.1 || ^14.13.1",
    "npm": "^6.4.1"
  },
  "types": "./types/index.d.ts",
  "dependencies": {
    "fabric-common": "file:../fabric-common",
    "fabric-protos": "file:../fabric-protos",
    "long": "^4.0.0",
    "nano": "^8.2.2"
  },
  "nyc": {
    "exclude": [
```

Attempting to register ‘toolchain’ repo and being blocked by npm similar name rules



```
npm ERR! 403 403 Forbidden - PUT https://registry.npmjs.org/toolchain - Package name too similar to existing packages; try renaming your package to '@kdic
ktevora/toolchain' and publishing with 'npm publish --access=public' instead
```

References

- Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies
  - https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610
- 3 Ways to Mitigate Risk When Using Private Package Feeds
  - https://azure.microsoft.com/mediahandler/files/resourcefiles/3-ways-to-mitigate-risk-using-private-package-feeds/3%20Ways%20to%20Mitigate%20Risk%20When%20Using%20Private%20Package%20Feeds%20-%20v1.0.pdf

Recommendations

Ensure local dependencies are referenced by explicit file path rather than name alone. Ensure that all dependencies are present and resolvable. Ensure dependencies in use are monitored, and patched, when vulnerabilities or supply chain attacks are discovered.

APP-01310 Fabric CA Bootstrapped with Admin Password in Cleartext on Command Line

Description

The Fabric CA Server must be bootstrapped with an admin password if not using LDAP. Currently, this is done via the -b command-line argument providing both a username and a password as input.

Tevora advises against CLI arguments with cleartext passwords as a default bootstrapping scheme, because this can lead to the password being stored in insecure locations such as git repositories or centralized audit logs. Tevora commonly observes and takes advantage of this behavior in other applications during infrastructure penetration tests.

This is not a major issue, as it typically requires multiple security failures in the deployed environment unrelated to Fabric to be exploitable in practice, however, because the Fabric CA is a tentpole of a Fabric organization's security, Tevora suggests implementing a more stringent bootstrapping procedure by default.

Status	Discovered	CVSS Base Score	3.9	Low	HydraRisk	6	Low
					Consequence	2	
					Probability	1	
					Velocity	1	
					Criticality	1	
					Responsiveness	1	

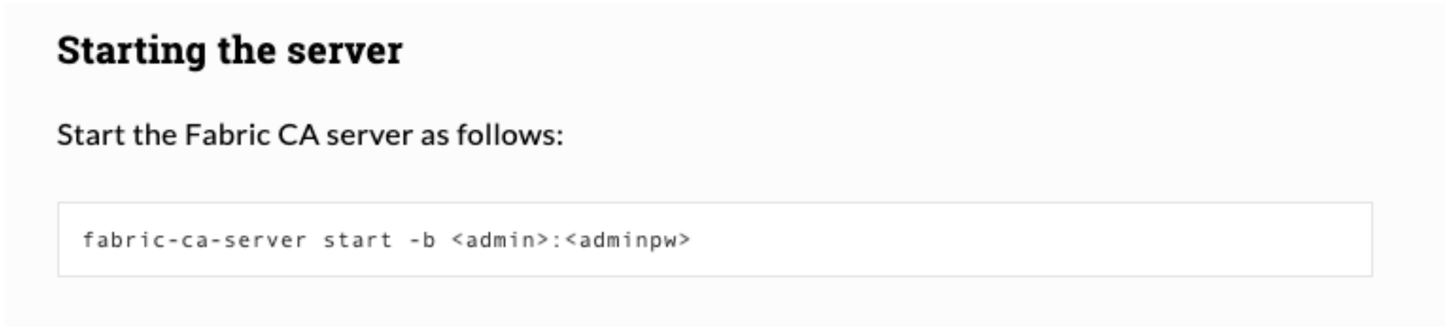
Affected Systems

The following systems have been identified and are affected:

Repo
https://github.com/hyperledger/fabric-ca

Details

Plaintext password default bootstrap instructions on https://hyperledger-fabric-ca.readthedocs.io/en/release-1.4/users-guide.html#initializing-the-server



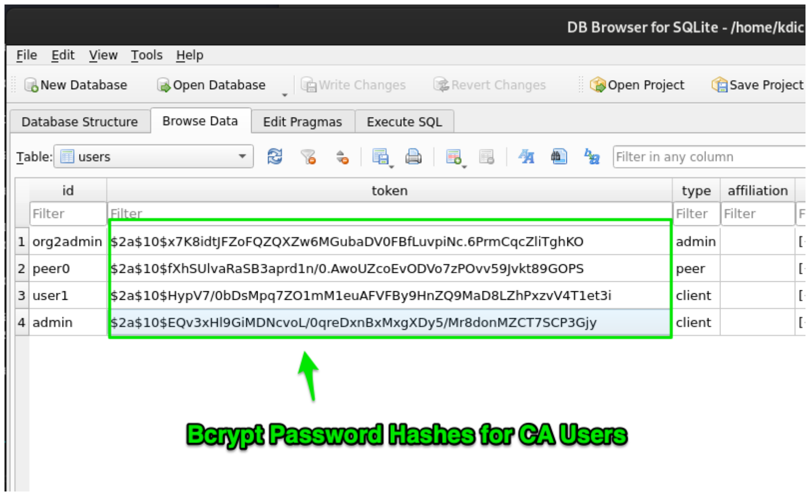
Starting the server

Start the Fabric CA server as follows:

```
fabric-ca-server start -b <admin>:<adminpw>
```

Passwords stored as Bcrypt in Fabric CA database

Bcrypt hashes are a great choice, and could potentially be specified instead of the cleartext in the command line parameter for additional security.



	id	token	type	affiliation	F
	Filter	Filter	Filter	Filter	
1	org2admin	$2a$10$x7K8idtJFZoFQZQXZw6MGubaDV0FBfLuvpiNc.6PrmCqcZliTghKO	admin		[
2	peer0	$2a$10$fXhSUlvaRaSB3aprd1n/0.AwoUZcoEvODVo7zPOvv59Jvkt89GOPS	peer		[
3	user1	$2a$10$HypV7/0bDsMpq7ZO1mM1euAFVFBy9HnZQ9MaD8LZhPxzvV4T1et3i	client		[
4	admin	$2a$10$EQv3xHl9GiMDNcvoL/0qreDxnBxMxgXDy5/Mr8donMZCT7SCP3Gjy	client		[

Recommendations

Tevora recommends, by default, generating a strong random password for the CA admin account and presenting it to the user (or orchestration software) on standard out.

For pushing custom passwords directly, Tevora recommends allowing users to specify a strong hash (such as the Bcrypt observed in the actual SQLite database) to push the password, rather than a plaintext password specification.

Overall, this would only be a marginal security improvement, so Hyperledger should weigh it against any difficulties it may cause users in implementation.

Appendix A: About Tevora

Tevora is a leading management consulting firm specializing in enterprise risk, compliance, information security solutions, and threat research. We offer a comprehensive portfolio of information security solutions and services to clients in virtually all industries and serve institutional and government clients.

Tevora's leaders are professionals with years of experience and records of accomplishments in technology as well as business. This dual background means that we understand the importance of growth and profitability and our solutions are designed to enhance both.

As a consulting firm that can fully implement whatever it recommends, Tevora works with all the industry's top vendors, yet is beholden to none. Our work and dedication have established us as a reliable partner CTOs CIOs, and CISOs can depend on to help protect against threats, both internal and external. With Tevora as a partner, business leaders can devote their energies to enhancing the overall value of information technology to their enterprise.

Tevora is a Qualified Security Assessor (QSA) and Payment Application Qualified Security Assessor (PA-QSA) in good standing with the PCI Security Standards Council. Tevora is also a DVBE (Disabled Veteran Business Enterprise) certified by the California General Services Department (Cert REF# 32786). For more information, please visit www.tevora.com.

Report Content

This report has been compiled for the exclusive use of Linux Foundation. Care has been taken to ensure that all report content and recommendations are of the highest quality and are based on sound analysis, research, and experience. Please direct any questions or concerns about the content of this report to Clayton Riness at criness@tevora.com.

Clayton Riness

Clayton Riness, Managing Director

Appendix B: Scoring of Findings

Penetration Test findings are qualified using the CVSS Version 3,1 Base Score and the Tevora proprietary HydraRisk model.

CVSSv3.1 Scoring

The CVSS version 3.1 vulnerability scoring system produces a base vulnerability score based on an Impact, and Exploitability metrics. This score is recorded for all applicable findings and is intended to provide an objective, industry-standard view of the vulnerabilities that have been found and potentially exploited.

Scoring guidelines:

- The CVSS version 3 Temporal and Environmental score metrics are not used in this report. Those factors are captured in the HydraRisk scoring model.
- In cases when multiple vulnerabilities with differing CVSS scores are summarized into a single finding, the highest contributing CVSS score is used for that finding.
- Some findings may not be given a CVSS since there is no known vulnerability but where an issue was found with the in-scope environment which differs from industry best practices or which may be used in combination with other findings to exploit a system.

HydraRisk Scoring

Enterprise risk management is an enterprise approach to addressing the culture, processes and structures that are directed towards effective management of potential opportunities and adverse effects as they relate to risk. Taking control of informed risks allows for risks to be identified, analyzed, evaluated, treated, and monitored.

Tevora's proprietary HydraRisk Model is founded on extensive experience in enterprise risk management which has been adapted for the scoring penetration testing results. The HydraRisk score is the sum of the score for all five factors defined as follows.

Consequence

The information security impact a threat and/or exploit has on the organization.

1	Trivial: Non-vital information disclosure: email addresses, WHOIS info, etc.
2	Reasonable: Disclosure of non-public but non-vital information
3	Significant: Non-privileged system access
4	Intolerable: Privileged system access through exploit, pivoting, or escalation
5	Major: Exfiltration of data: PCI, PII, intellectual property, etc.

Probability

The likelihood of the vulnerability/threat to be exploited.

1	Low: No known exploit, requires skilled attacker creating a new 0-day
2	Unlikely: Exploit only possible using specialized tools
3	Moderate: Exploit is possible using common attacks or attack chaining
4	High: Easy to exploit by low skilled penetration tester using common tools
5	Critical: Easy to exploit with simple tools that are readily available

Velocity

Assessment of how quickly a vulnerability could be exploited.

1	Protracted: Requires brute forcing crypto, application fuzzing, etc. over extended period
2	Slow: Requires extensive rainbow tables or other reference libraries to exploit
3	Moderate: Requires readily available reference libraries or casual observation to exploit
4	Quick: Requires casual observation to discover exploit
5	Immediate: Vulnerability can be discovered and exploited readily

Criticality

The depth and breadth of the impact including the types of systems compromised or affected by exploiting this vulnerability.

1	Trivial: vulnerability affects unimportant systems: ancillary support systems
2	Reasonable: exploitation affects access to DMZ or other highly segmented hosts
3	Significant: exploitation affects access to loosely segmented hosts or client environment
4	Intolerable: exploitation affects substantial portions of the environment and data
5	Major: exploitation affects access to critical data, data integrity, and availability

Responsiveness

The time required to treat and prevent the exploit from occurring.

1	Excellent: vulnerability patch or reconfiguration for exploit is readily available
2	Good: vulnerability patch is in development or a workaround is available
3	Moderate: patching, reconfiguration, and/or infrastructure re-architecting is required
4	Fair: infrastructure modification and/or downtime required to remediate
5	Poor: major infrastructure modification and/or downtime required to remediate

Scoring Key

The following scoring key is used throughout this report, with CVSS scores ranging from 0-10 while HydraRisk scores range from 5-25.

Risk Rating	HydraRisk Score
Critical	21-25
High	16-20
Medium	11-15
Low	5-10

Risk Rating	CVSS Score
High	7.0-10.0
Medium	4.0-6.9
Low	0.0-3.9

All findings are categorized as follows:

Status	Description
Informational	No security risk present
Discovered	Security risk discovered and verified, but not successfully exploited
Exploited	Security risk successfully exploited with proof of concept attack

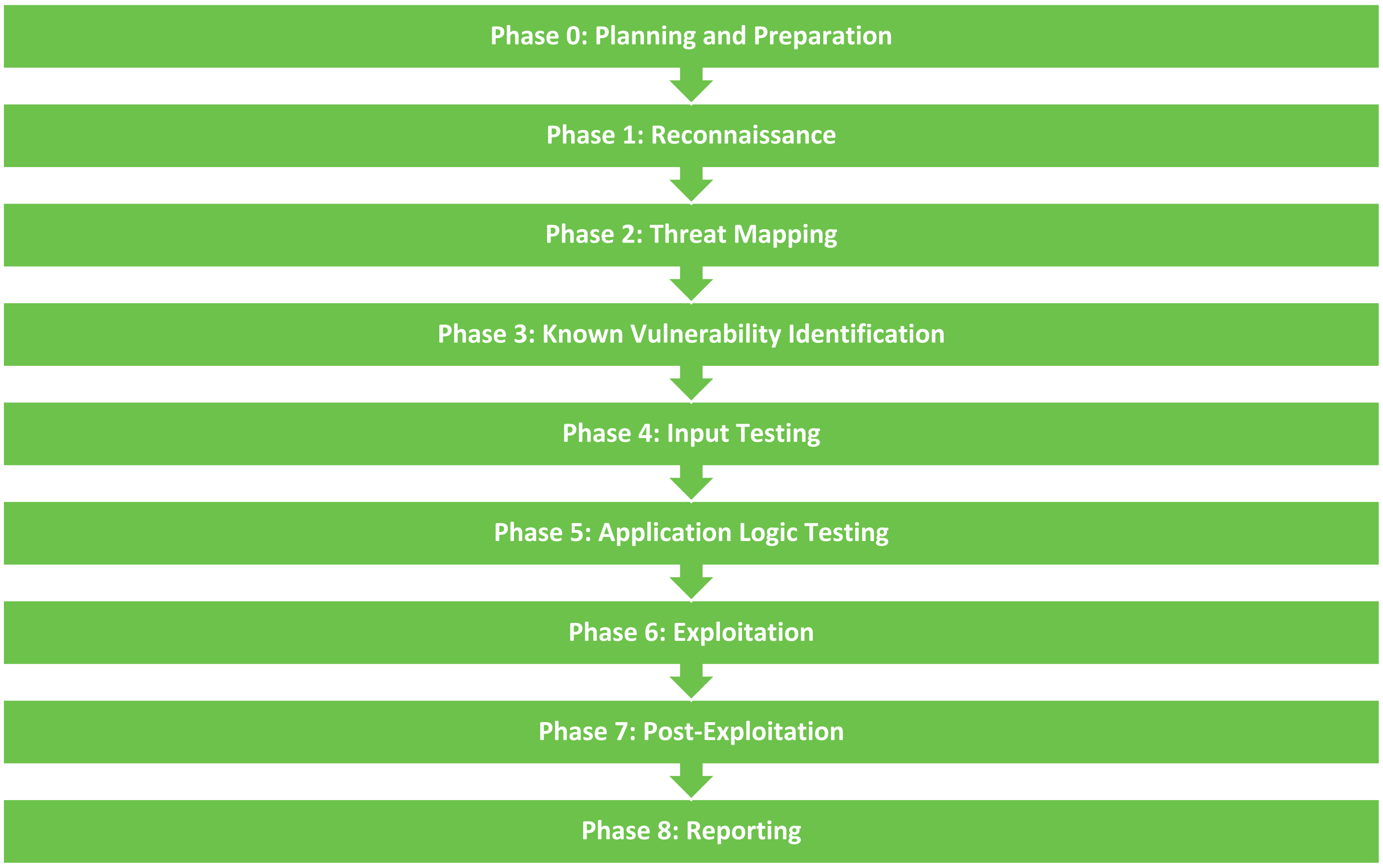
Penetration Testing Tools

Tevora employs many tools during penetration test to assist and complement manual testing including:

- Nessus Professional
- BurpSuite Pro
- ZAP (Zed Attack Proxy)
- SQLmap
- Acunetix
- NetSparker
- Custom Python scripts
- DirBuster

Appendix C: Penetration Testing Methodology

Tevora employs a standard methodology to ensure a repeatable level of quality in all assessments. Tevora's testing methodology is based on the Penetration Testing Execution Standard (PTES)[1], OWASP testing guide v4[2], and years of experience in network, web, and application penetration testing.


Phase 0: Planning and Preparation
Phase 1: Reconnaissance
Phase 2: Threat Mapping
Phase 3: Known Vulnerability Identification
Phase 4: Input Testing
Phase 5: Application Logic Testing
Phase 6: Exploitation
Phase 7: Post-Exploitation
Phase 8: Reporting

[1] http://www.pentest-standard.org/index.php/Main_Page
[2] https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf

Phase 0: Planning and Preparation

A successful penetration test begins with planning and preparation. During this phase, Tevora works with the Client to identify the scope and any prerequisites to project execution. Tevora performs the following pre-engagement activities to prepare for testing:

- **Scope Identification:** Tevora and the Client identify the in-scope targets to be tested.
- **Testing Window Identification:** The Client provides the range of acceptable testing windows and Tevora decides when the testing will be performed within that range.
- **Objective Identification:** Tevora and the Client discuss and agree on objectives for the test. These will be used to focus testing and ensure relevant results. Specifically, the expected security model of the target is discussed, and high impact compromises of the model are identified as objectives.
- **Gather Relevant Documentation:** Tevora works with the Client to acquire IT and business process documentation. Tevora can also take a zero-knowledge approach and attempt to acquire this information during the reconnaissance phase of the test.
- **Determine Level of Access:** Based on the objectives, Tevora and the Client determine if credentials are to be provided by the Client for testing. For the most thorough testing, Tevora will use low-level privileges.
- **Time Estimation:** Tevora determines the estimated time needed to cover the scope for the decided testing types.
- **Role Identification:** Tevora assigns a project lead, technical lead, and assistant technical lead to the test. Tevora's technical will have web services and web application specialists assigned to the project including at least one subject matter expert (SME) on the in-scope technologies.
- **Kickoff Meeting:** Tevora and the Client review the planned scope, discuss the project overview, and propose scheduling.
- **Testing Contact Identification:** Tevora and the Client identify their respective points of contact and determine testing status update intervals. Tevora provides an escalation list to the Client.
- **Incident Handling:** Tevora and the Client agree to a response plan for unexpected issues during testing.
- **Project Checklist:** Tevora ensures that every item for the project is checked prior to beginning the penetration test.

After preparation has been completed, the project checklist reviewed, and scheduling finalized, Tevora will begin the penetration test on the scheduled date.

Phase 1: Reconnaissance

The first phase of a penetration test is reconnaissance. This phase is conducted to gather information on the target and enumerate potential threat vectors. Tevora performs reconnaissance in a strategic manner that emulates the process of real-world adversaries. This process, called Open Source Intelligence Gathering (**OSINT**), is a multi-level approach that consists of several types of information gathering activities.

OSINT is done in three phases: **Passive**, **Semi-Passive**, and **Active**:

- **Passive:** Tevora searches the internet for information that is posted by the Client or their employees. Tevora reviews third-party databases that could contain archived Client or employee information including Google, Shodan, and social networking platforms. Traffic is never sent to the Client during this phase, making the testing difficult to detect.
- **Semi-Passive:** Tevora gathers information on the target using requests disguised as normal internet traffic, including DNS requests, service probes, and analysis of document metadata. Traffic may be sent to the Client but will be difficult to detect.
- **Active:** Tevora uses ping sweeps, port scans, banner grabbing, vulnerability scans, and forced browsing to actively enumerate the Client's attack surface. This is a more aggressive phase of reconnaissance that generates significant amounts of abnormal traffic. Tevora gathers a significant amount of reliable information on the Client's systems during this phase. This phase is most likely to be detected by the Client.

Phase 2: Threat Mapping

Tevora analyzes the information gathered during the reconnaissance phase to map targets to potential threat vectors. This map is used to enumerate threats to the business and prioritize testing on high-impact targets.

The threat mapping phase closely follows the PTES Standard's threat modeling phase. During threat mapping, Tevora performs the following steps:

- **Gather relevant documentation:** Tevora works with the Client to acquire IT and business process documentation. Tevora can also take a zero-knowledge approach and attempt to acquire this information during the reconnaissance phase.
- **Identify and categorize primary and secondary assets:** Tevora identifies the assets on the in-scope targets and divides them into primary and secondary categories. These are assets that can be reached directly, and assets that can be reached from pivoting, respectively.
- **Identify and categorize threats and threat communities:** Tevora enumerates the potential threats to the in-scope targets and categorizes them by the groups of people (e.g., threat communities) that may execute those threats.
- **Map threat communities against primary and secondary assets:** Tevora maps the categorized threat list to the categorized asset list to determine relevant threats and their potential impact on the business.
- **Cross-reference threat map to test objectives:** Tevora reviews the threat map to identify the impact of potential threats in the context of testing objectives defined during the planning phase.

Tevora uses the output of this phase to enumerate potential threat vectors and prioritize testing on high-impact attack scenarios. This also enables alignment of threat exposure to testing objectives.

Phase 3: Known Vulnerability Identification

Tevora reviews information gathered during the threat mapping and reconnaissance phases to identify known vulnerabilities. Tevora reviews banners, network and HTTP response signatures, and running services. These are then cross-referenced against vulnerability databases such as Exploit-DB, Rapid7, and CVE.

Tevora takes a multi-assessment approach by analyzing information gathered from both passive and active vulnerability identification:

- **Passive:** Tevora reviews metadata from public documents and archived content in search engines for vulnerability signatures. Additionally, Tevora performs traffic monitoring on the internal network and analyzes network protocols for signatures of vulnerable network services.
- **Active:** Tevora uses vulnerability scanners for automated vulnerability enumeration and augments this with output from port scanners, HTTP responses, SNMP enumeration, NetBIOS enumeration, and more.

After identifying vulnerabilities, Tevora attempts to validate vulnerabilities and prioritize them for exploitation. Tevora researches all discovered vulnerabilities and performs manual testing to check for false positives. Vulnerabilities are cross-referenced against the threat map to identify their impact and potential risk to the business.

Phase 4: Input Testing

Tevora tests for input validation and injection issues on web application forms. Tevora fuzzes input fields using a combination of manual and automated techniques.

Tests performed include:

- LDAP Injection
- ORM Injection
- Directory Traversal / File Inclusion
- XML Injection
- SSI Injection
- XPath Injection
- IMAP/SMTP Injection
- Code Injection
- OS Commanding
- Buffer Overflow
- Incubated Input Vulnerability Testing
- HTTP Splitting/Smuggling
- SQL /NoSQL Injection

Any discovered input vulnerabilities are categorized, referenced against the threat map, and documented for use in the exploitation phase.

Phase 5: Application Logic Testing

Tevora uses information gathered during previous phases to analyze the web application's security model. Tevora reviews the application logic at all points in the platform to identify application logic weaknesses which may expose sensitive information or functionality.

Application logic tests make up the bulk of time spent on web application penetration tests. Application logic testing is a primarily manual process, with custom scripts and plugins used to automate testing for certain flaws, such as enumeration. Tevora focuses on high-impact functionality during this phase as well as complex multi-step processes, which are more likely to include dangerous bugs.

Testing performed includes:

- Data Flow Analysis
- Business Logic Testing
- Authentication, Authorization, Session management
- Sensitive Functionality Testing
- Local Storage
- Error Handling
- Client-Side Control Bypasses
- ID Enumeration

The impact of any data leakage, or unauthorized activities discovered during application logic testing are categorized and referenced against the threat map. If relevant, identified issues may be used during the platform exploitation phase.

Phase 6: Exploitation

During the exploitation phase, Tevora attempts to access the targets enumerated during the threat mapping phase. Tevora reviews discovered vulnerabilities and potentially insecure services to develop an exploitation plan. Tevora then executes this plan in a precision strike against the Client.

Tevora uses publicly available exploits and pursues development of custom and/or "zero-day" exploits for high impact targets or when known vulnerabilities are not discovered.

- **Known Vulnerabilities:** Tevora modifies public exploits to target the Client environment. Public exploits are only acquired from trusted sources such as Exploit-DB and are reviewed before modification and use. Commercial exploitation frameworks are also used during this phase.
- **Unknown Vulnerabilities:** If known vulnerabilities are not found, Tevora takes a zero-day approach. A replica environment is created and Tevora tests the discovered services for previously unknown security issues.
- **Application Layer Vulnerabilities:** If any custom applications are discovered during testing, Tevora will perform application-level assessments as permitted by the timeframe. These tests will be performed according to Tevora's application testing methodologies.

Tevora delivers payloads during the exploit to gain access to the targets in accordance with testing objectives. Payloads are designed to bypass security measures used by the Client. These will include encoded, packed, encrypted, and custom payloads designed to bypass anti-virus, IPS/IDS systems, and firewalls. These payloads are also used in the post-exploitation phase to pivot the attack to other targets.

Phase 7: Post-Exploitation

During this phase, Tevora evaluates the impact of the exploitation, tests the Client's internal defenses, and uses the initial exploits to escalate access to additional targets. The following activities are performed during this phase:

- **Establish Persistence:** Tevora establishes secure, persistent access so Tevora may notify the Client of the exploit and the Client can remediate without interrupting post-exploitation activities.
- **Initial Enumeration:** Compromised resources are enumerated for relevant information. User accounts and passwords are extracted for use in pivoting.
- **Pivoting:** Tevora repeats the reconnaissance, threat mapping, vulnerability identification, and exploitation phases on newly accessible targets. Tevora begins the new reconnaissance phase with network analysis and shifts to an internal penetration test methodology. Tevora uses information acquired during previous phases to escalate access to the Client's systems.
- **Target Profiling:** Tevora enumerates data and information on exploited targets.
- **Data Exfiltration:** Based on the purpose of the penetration test, Tevora targets and attempts to extract (or simulate an extraction of) information that is vital to the organization.
- **Cleanup:** When the penetration test is complete, Tevora cleans up all the tools and payloads that were placed in the target's environment.

Post-exploitation is an iterative testing process to continually escalate the attack simulation. Previous steps of the methodology are repeated to assess potential threats from the newly acquired foothold. Additional information about the target may be discovered during this phase such as source code, undocumented endpoints, and additional credentials, which all warrant further testing.

Phase 8: Reporting

Tevora compiles the findings during the penetration test and organizes them into a final report which is sent to the Client. The report documents each discovered vulnerability, remediation recommendations, and provides an analysis of risk to the business.

Topics covered by the report include:

- Executive Summary
  - People involved
  - Project objective
  - Project scope
- Findings Overview
  - Test results
  - Strategic recommendations
- Technical Summary
  - Scoring of findings
  - Findings summary based on HydraRisk model
  - Detailed summary of each finding
    - CVSS score
    - HydraRisk score
    - Finding description
    - External references
    - Recommended remediation
- Penetration Testing Methodology

The report provides both a detailed technical breakdown and a high-level executive summary, allowing for review by both technical and non-technical staff. The report can be tailored to a Client's needs, including being split into multiple documents. The report is the final deliverable for testing and may go through review and editing phases prior to acceptance. Once the report has been accepted, the project is considered closed unless otherwise stated.