Hyperledger Fabric Security Monitoring based on Hyperledger Explorer

Benedikt Putz & Fabian Böhm

Chair of Information Systems, University of Regensburg, Germany

go.ur.de/ifs

Agenda

- Motivation
- Background on Hyperledger Fabric and Explorer
- Security Monitoring Architecture
- Processing Pipeline and Live Demo
- Q&A

Why you should care about Hyperledger Fabric security

Fabric is trusted to provide security for critical applications

- Tracks >50% of global container shipping
- Executes large trade finance transactions
- … many other production use cases

Despite blockchain's built-in crypto, security should NEVER be taken for granted

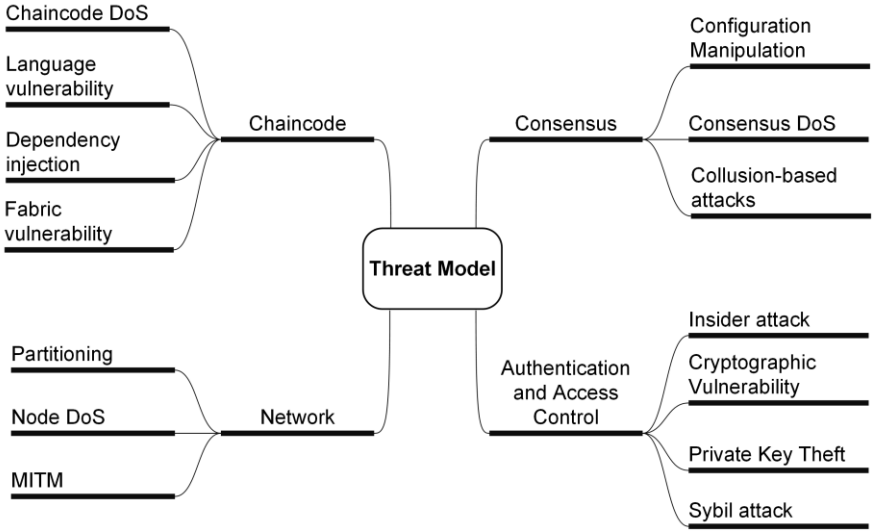
Like any software, Fabric is vulnerable to bugs, exploits, and DoS

Independent operation and configuration may increase the attack surface

Numerous attack vectors for Fabric are known

Precondition: many attacks require access to the network or elevated/admin privileges
Assumption: cryptographic primitives are safe

- Denial of Service: request or tx flooding, malicious transactions
- Chaincode bugs (read-after-write, non-determinism)
- Unsafe default configuration (State DB)
- Current lack of BFT consensus algorithm
- Server compromise/credential theft



Potential consequences: Downtime, inaccurate world state, loss of confidential data

Security analysts need support to detect attacks

Several independent peers with a limited view of the network

Monitoring of both host and network data necessary

Lack of fully automated systems for live attack detection

Each node with various data sources from components

Large volume and velocity of observable data

Crucial domain knowledge of human experts to identify attacks

Agenda

- Motivation
- **Background on Hyperledger Fabric and Explorer**
- Security Monitoring Architecture
- Processing Pipeline and Live Demo
- Q&A

Fabric Architecture is modular including many data sources

Everything runs in Docker containers

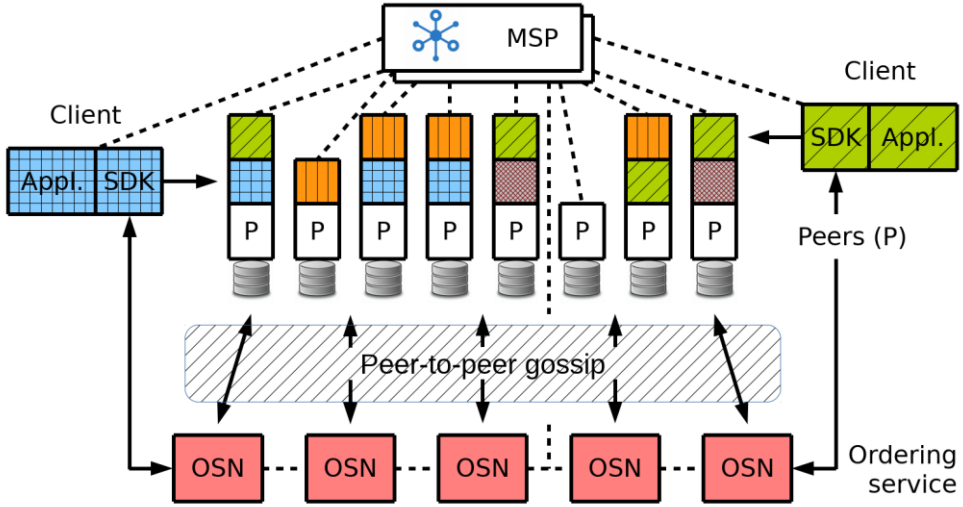
Configurable logging levels

Blockchain data via peers' SDK API *

* Requires channel subscription for most of the interesting data

Numeric metrics via Prometheus (or statsd**)

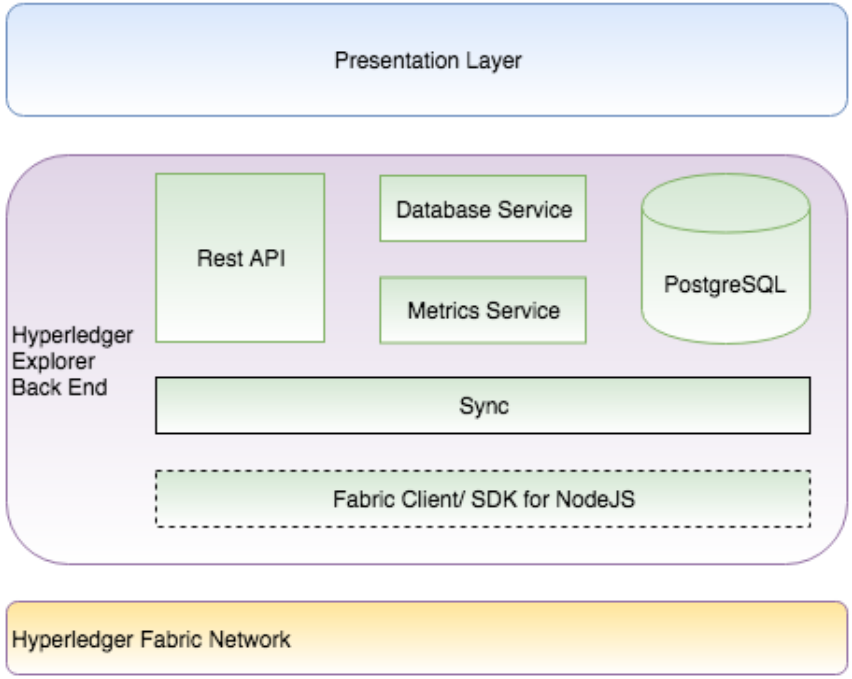
** disabled by default



Hyperledger Explorer prepares blockchain data for analysis

Sync service runs channel subscriptions

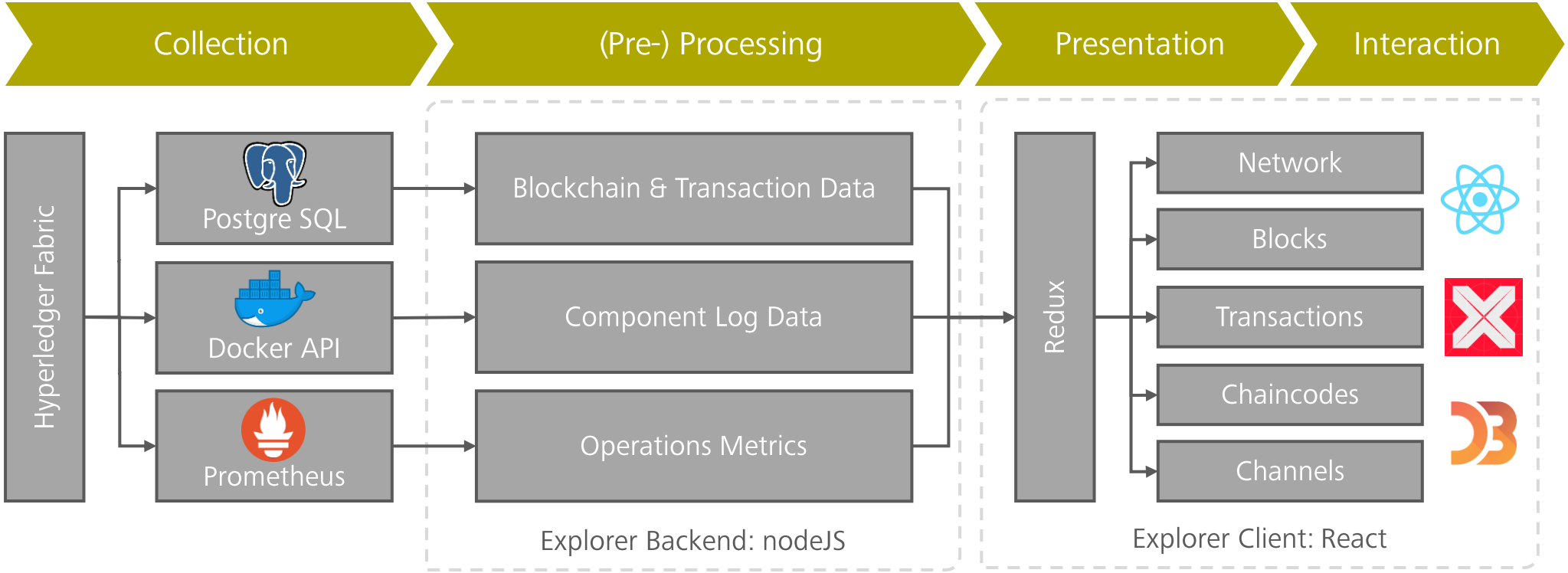
Relevant block and transaction data is persisted in PostgreSQL-DB's relational schema



Agenda

- Motivation
- Background on Hyperledger Fabric and Explorer
- **Security Monitoring Architecture**
- Processing Pipeline and Live Demo
- Q&A

HyperSec follows a standard data processing pipeline



github.com/sigma67/hypersec

5k LoC changed from Explorer's original code base

Backend

Persistence
- use more data from transactions (size, identity)

Platform
- add config tx notifications

Sync
- use more data from transactions (size, identity, config tx)

REST
- *dbroutes*: update tx routes with chaincode filter
- *metricroutes*: Prometheus reverse proxy routes
- *logroutes*: Docker reverse proxy routes
- *externalroutes*: Hyperledger JIRA reverse proxy for current issues

app	**1187**
persistence	30
fabric	30
CRUDService.ts	24
postgreSQL	6
platform	**854**
fabric	**854**
config.json	4
connection-profile	12
test-network.json	12
FabricClient.ts	10
Proxy.ts	714
sync	**111**
FabricEvent.ts	13
SyncPlatform.ts	25
SyncService.ts	73
utils	**3**
FabricConst.ts	3
rest	**303**
dbroutes.ts	22
externalroutes.js	43
logroutes.js	72
metricroutes.js	163
requestutils.ts	3

Client (Frontend)

Charts
- *Transaction**: interactive charts for tx data and related metrics
- *NotificationsPanel*: config notification

Lists
- *Issues*: display Hyperledger JIRA issues
- *Transactions*: transformed into function component

View
- *NetworkView*: interactive network chart
- *TransactionsView*: included the interactive charts for tx data and related metrics

State
- connectors for new backend API routes

client	**4031**
src	**4031**
components	**3838**
Charts	**1135**
Map.js	84
TransactionBrush.js	164
TransactionCount.js	305
TransactionSize.js	263
TransactionTime.js	305
TransactionUser.js	14
Header	**17**
HeaderView.js	17
Lists	**987**
Blocks.js	93
Chaincodes.js	72
Issues.js	117
Peers.js	97
Transactions.js	608
Main.js	**46**
Panels	**38**
NotificationsPanel.js	38
Styled	**16**
Table.js	16
Theme	**51**
Theme.js	51
types	**14**
index.js	14
View	**1534**
BlockView.js	71
ChaincodeScanModal.js	52
DashboardView.js	9
LandingPage.js	3
LogView.js	125
NetworkView.js	651
TransactionsView.js	597
TransactionView.js	4
useResizeObserver.js	22
state	193
redux	**193**
charts	100
tables	93

Agenda

- Motivation
- Background on Hyperledger Fabric and Explorer
- Security Monitoring Architecture
- **Processing Pipeline and Live Demo**
- Conclusion / Q&A

HyperSec collects additional security-relevant data

Transactions

Prometheus
- Endorser Proposal Duration
- Broadcast Enqueue Duration
- Broadcast Validate Duration

Block Data
- Size
- Submitter Identity

Peers

GRPC
- Stream messages sent (delivered blocks, broadcast)
- Gossip messages received
- Is connected to HL Explorer

Missing/Desirable

Orderer
- outstanding/unprocessed tx
- discarded blocks
- failed leader elections

Vulnerabilities
- threat intelligence by version
- chaincode scanners for other languages

Logs

Docker
- all local orderer containers
- all local peer containers
- certificate authority containers (todo)

Vulnerabilities

Hyperledger JIRA
- Latest issues based on Security Tag

Chaincode Scans
- revive-cc tool based on Go static analysis
- scan result inserted to DB

Data binning improves **HyperSec** frontend performance

User Input

start: 3 | **end**: 34 | **step**: 9

1 Query based on user input

(4-3) / 9 = 0 → **B1**

(11-3) / 9 = 0 → **B1**

Raw transactions (Explorer)

T1: {**timestamp**: *4*, **size**: *12*, **creator_msp_id**: *orgA*}

T2: {**timestamp**: *11*, **size**: *34*, **creator_msp_id**: *orgA*}

T3: {**timestamp**: *18*, **size**: *4*, **creator_msp_id**: *orgB*}

T4: {**timestamp**: *25*, **size**: *89*, **creator_msp_id**: *orgC*}

Raw metrics (Prometheus)

M1: {**timestamp**: *8*, **metricA**: *12*, **metricB**: *123*}

M2: {**timestamp**: *23*, **metricA**: *23*, **metricB**: *324*}

M3: {**timestamp**: *24*, **metricA**: *12*, **metricB**: *432*}

M4: {**timestamp**: *31*, **metricA**: *23*, **metricB**: *423*}

2 Prepare data buckets

Data buckets

B1: {**id**: *timestamp / step*, **timestamp**: *3*}

B1: {**id**: *timestamp / step*, **timestamp**: *3 + 9*}

B3: {**id**: *2*, **timestamp**: *21*, **transactions**: *[]*, **metrics**: *[]*}

B4: {**id**: *3*, **timestamp**: *30*, **transactions**: *[]*, **metrics**: *[]*}

Populate buckets 3

Calculate ID of bucket: *(timestamp – start) /step*

HyperSec visualizes data with well established libraries

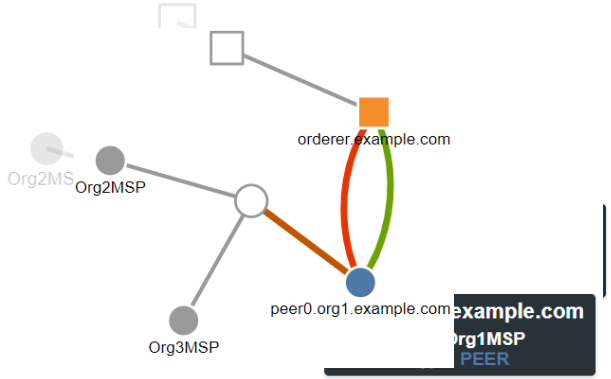
Main Frameworks

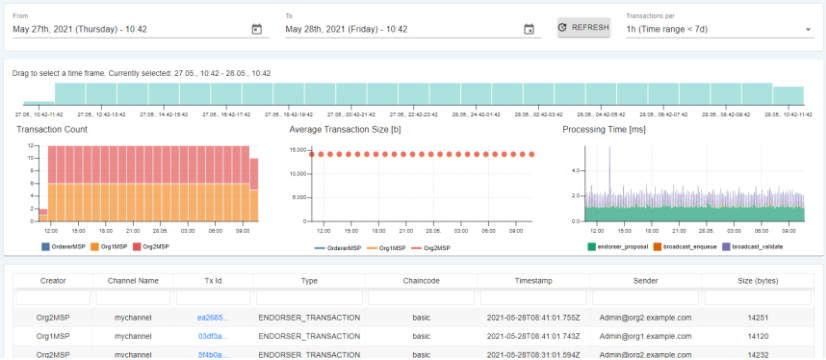
https://reactjs.org/

https://airbnb.io/visx

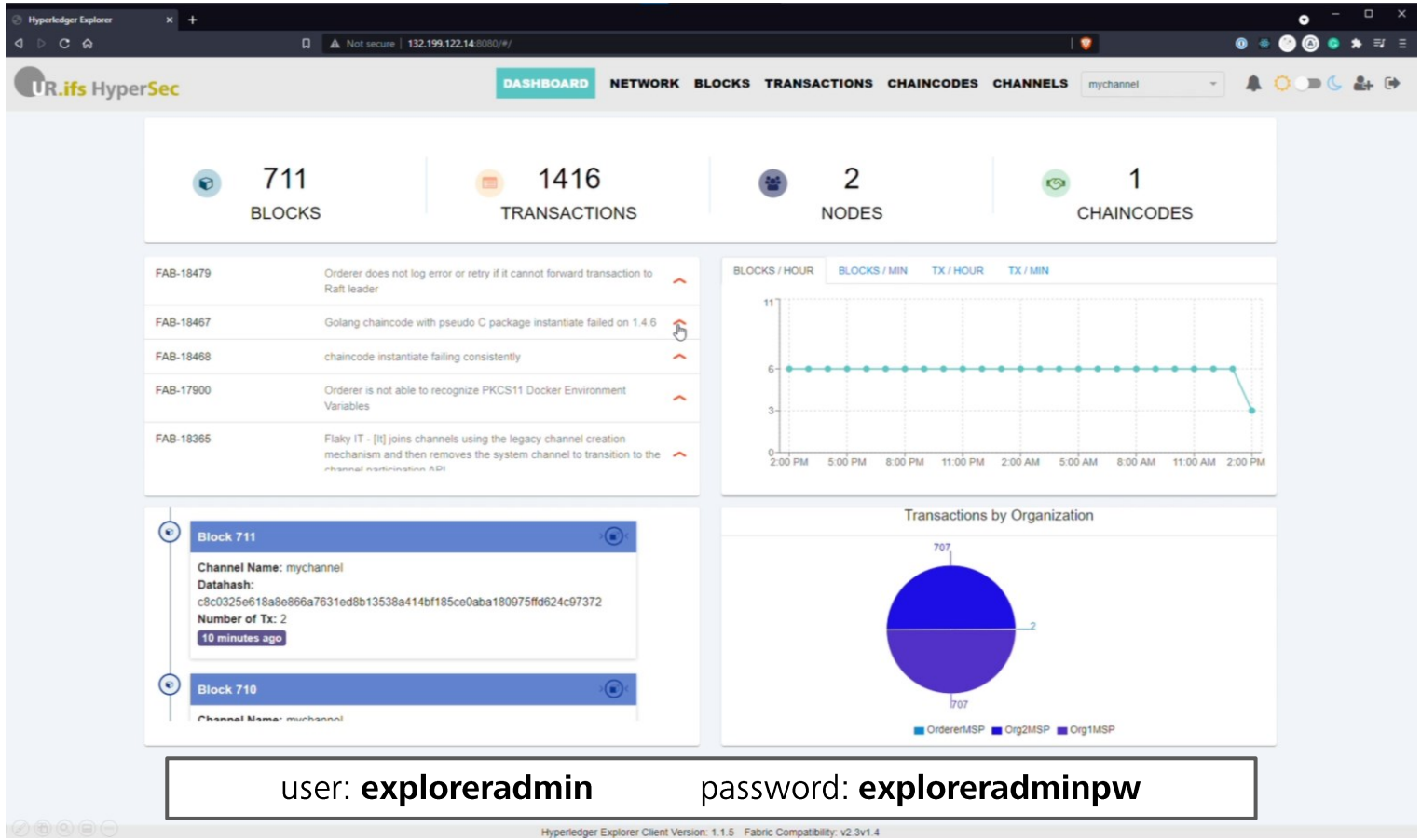
https://d3js.org/

New visualizations added to Hyperledger Explorer





Live Demo



user: **exploreradmin** password: **exploreradminpw**

Thank you for your attention!

Questions and feedback are very welcome!

Check out **HyperSec** at **github.com/sigma67/hypersec**