HyperSec: Visual Analytics for Blockchain Security Monitoring

Benedikt Putz, Fabian Böhm, Günther Pernul

HyperSec: Visual Analytics for blockchain security monitoring

Benedikt Putz[0000−0002−4265−1106], Fabian Böhm[0000−0002−0023−6051], and Günther Pernul[0000−0003−1338−9003]

University of Regensburg, Regensburg, Germany
{firstname.lastname}@ur.de

Abstract. Today, permissioned blockchains are being adopted by large organizations for business critical operations. Consequently, they are subject to attacks by malicious actors. Researchers have discovered and enumerated a number of attacks that could threaten availability, integrity and confidentiality of blockchain data. However, currently it remains difficult to detect these attacks. We argue that security experts need appropriate visualizations to assist them in detecting attacks on blockchain networks. To achieve this, we develop HyperSec, a visual analytics monitoring tool that provides relevant information at a glance to detect ongoing attacks on Hyperledger Fabric. For evaluation, we connect the HyperSec prototype to a Hyperledger Fabric test network. The results show that common attacks on Fabric can be detected by a security expert using HyperSec's visualizations.

Keywords: distributed ledger · permissioned blockchain · information security · visual analytics · security monitoring

1 Introduction

New use cases of distributed ledger technology (DLT) are proposed on a daily basis by academia and practice, leading to an increasing number of projects and solutions. Beyond that, blockchain applications are increasingly being used in real large-scale supply chain environments, such as the TradeLens [10] and DLFreight [20] platforms. At first glance, DLT seems to increase an application's security or even solve existing applications' security issues. However, the task of securing the DLT itself is often neglected in practice due to its complexity and the number of serious challenges connected to it.

The complexity of blockchain technology makes it particularly challenging to identify malicious activities [4]. In any blockchain network, there are several independent peers operated by independent organizations, where each organization only has a limited view of the network. Each node also has various data

The first two authors have contributed equally to this manuscript.

sources from its components, making it difficult to obtain an overview of the network's state [17]. Since blockchain is a networked database, it also requires monitoring both the host and the network, which results in a large volume and velocity of observable data.

Fully automated systems for live attack detection on blockchains do not yet exist. Even if respective technologies for blockchain security monitoring were available, human experts remain indispensable as their domain knowledge is crucial to identify and analyze intricate attack patterns [2]. Therefore, we need a way to make the heterogeneous data at hand available for domain experts. Visualizations offer a well-known path to achieve this goal. A visual representation can help a domain expert make sense of the displayed information and efficiently draw conclusions [12]. These observations lead to our work's research question:

RQ. *What are appropriate visualizations to assist security experts in detecting DLT threats?*

In this work, we make a two-fold contribution to this research question. We first characterize the domain problem: monitoring permissioned DLTs for attacks. This domain problem and derived general design requirements serve as the foundation for our visualization approach. The second part of our contribution is the task-centered design and prototypical implementation of *HyperSec*, a visual representation of security-relevant DLT information to support security experts' monitoring tasks.

The remainder of this work is structured as follows. Section 2 gives a brief overview of related academic work in the field of security visualizations in the blockchain domain. In Section 3, we flesh out the domain problem faced by security experts monitoring permissioned blockchain environments for immediate threats. Section 4 then introduces our visualization design and its prototypical implementation using open source technologies. Afterwards, we evaluate our visualization design by simulating attacks in Section 5. We discuss how an expert may proceed after an attack has been detected in Section 6. Finally, Section 7 concludes our work with a summary and possible future research directions.

2 Related Work

Recently, Tovanich et al. [23] carried out a systematic review to structure existing work on the visualization of blockchain data. Their research and previously conducted studies [19] identify several visualization approaches with a focus on criminal and malicious activity [8,13]. These surveys highlight that visualization tools for blockchains are on the rise. However, most of these existing visualization approaches for criminal or malicious activities in blockchains focus on historical analysis, i.e. detecting the events only after they have occured [23].

To effectively prevent attacks upfront, blockchain networks have to be actively monitored by blockchain security experts. Several studies discuss external and internal threats that could impair a blockchain network's functionality [9,18].

Zheng et al. propose a framework for monitoring the Ethereum blockchain's performance [24]. They introduce some respective metrics while using both node logs and Remote Procedure Calls (RPC) to gather data. Threat indicators to detect malicious activities in a blockchain network have recently been introduced by Putz et al. [17]. Based on this limited body of work from academia, blockchain metrics and threat indicators need to be made available to security experts for effective monitoring. Existing monitoring solutions like the dashboard by Bogner [3] focus only on transaction activity but do not consider other security-relevant data and metrics.

An approach pointing in this direction is the Hyperledger Explorer [21], the Hyperledger project's tool for monitoring Hyperledger blockchains. The Explorer connects to a local blockchain node and extracts data about blocks, transactions, peers, and more into a local PostgreSQL database. Additionally, a web application is available for inspecting blockchain data, including some basic visualizations of transaction data. However, these visualizations are not tailored to provide the necessary insights or indicators to detect threats. In addition, there is a Hyperledger Labs project integrating Fabric with ElasticSearch and Kibana, resulting in a Kibana dashboard able to display some transaction data [1]. Unfortunately, their visualizations are not very well suited to detecting blockchain threats in Hyperledger Fabric. In our experiments we found that the necessary integration and aggregation of additional data sources and custom visualizations are difficult to achieve in standard products like the Elastic stack.

Analyzing related work highlights an evident lack of dedicated and security-specific visualization approaches enabling security experts to monitor blockchain networks in real-time, while detecting common indicators of compromise or ongoing attacks on the network. Our work contributes a valuable solution approach to this issue.

3 Blockchain Security Monitoring

This Section addresses the first part of our contribution. Within our main contribution, we follow the user-centered and problem-driven Nested Blocks and Guidelines model (NBGM) for visualization designs [14,16]. This allows us to identify and address security experts' core problems and lay a foundation for a visualization design fitting their needs.

The first step of the NBGM is the definition of a domain problem. We characterize the problem at hand based on two primary sources of information. First, we consider reports from blockchain security professionals [11]. Second, we analyze literature on blockchain attacks to identify concerns for operators of a blockchain node [7,9,18]. We begin by outlining the overall blockchain security monitoring process in Section 3.1. The domain problem is then specified according to Miksch and Aigner's design triangle through more in-depth descriptions of specific users (Section 3.2), their tasks (Section 3.3), and data elements (Section 3.4) [15]. We address the second step of the NBGM (*Data/Operation Abstrac-*

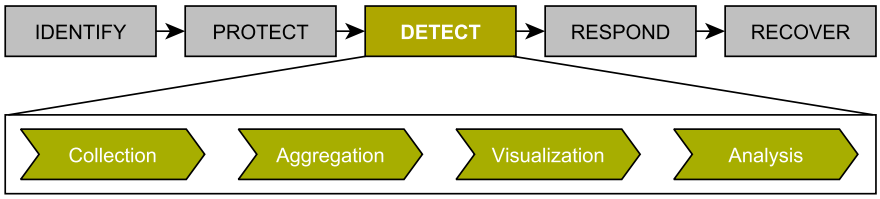

Fig. 1. Blockchain Security Monitoring process based on the NIST Cybersecurity Framework [5].

tion) in Section 3.5 by deriving general design requirements for a visualization approach to support blockchain security monitoring.

3.1 Blockchain Security Monitoring Process

Before we dive into users, tasks, and available data, we first need to understand the overall process underlying blockchain security monitoring. This subsection introduces our conceptual process based on the NIST Cybersecurity Framework for protecting critical infrastructures [5]. As shown in Figure 1, the framework has five main functions: *Identify*, *Protect*, *Detect*, *Respond* and *Recover*. We apply these functions to a permissioned blockchain network. The *Identify* function serves to identify relevant assets and risks. This problem has been already addressed in prior work [17]. *Protect* involves a variety of protection measures applied to the system: identity management and access control, data security, secure configuration, and backups/log files, among others. These protection measures are usually part of the blockchain framework itself, with additional measures being applied at deployment time (such as secure configuration and appropriate backup procedures) [22]. The *Detect* function currently lacks appropriate visualization and analysis tools. It's the focus of this work and further developed in the following subsection. During the *Respond* phase, threats detected using our visualization approach are met with a response plan and appropriate mitigation actions. Finally, the *Recover* function provides appropriate tools to restore functionality after an attack has occurred. *Respond* and *Recover* are not specifically part of this work as attacks need to be identified before effective *Respond* and *Recover* can take place. Corresponding tools might be integrated into future work to permit swift threat response.

The *Detect* function can be subdivided into four smaller process steps. Relevant data needs to be collected (*Collection*) and aggregated to provide appropriate metrics if necessary (*Aggregation*). Data and metrics can then be visualized (*Visualization*) allowing domain experts to identify possible threats (*Analysis*). Please note that all steps beside *Analysis* can be performed automatically.

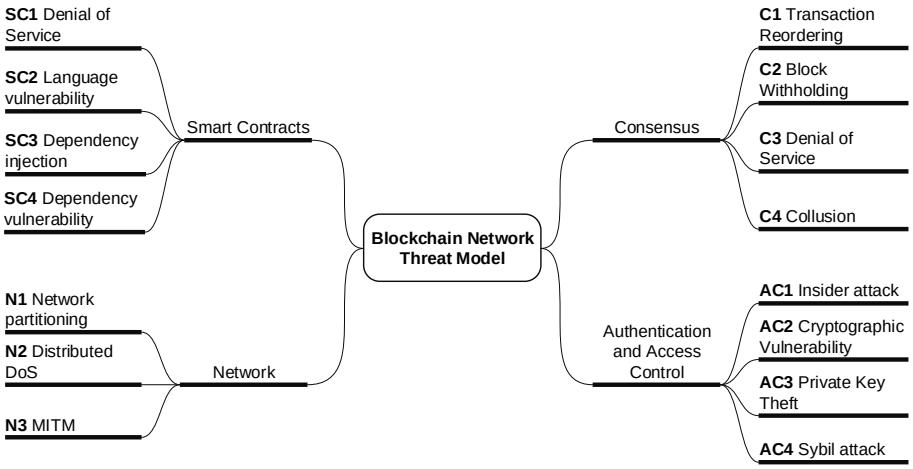

Fig. 2. Blockchain Networks Threat Model in attack tree notation.

3.2 Users

The intended users of visualization designs within the *Detect* function in the blockchain monitoring process are domain experts. These experts are responsible for analyzing blockchain data to identify malicious events within this function [11]. More specifically, we define the domain experts as security professionals knowledgeable in the cybersecurity domain. Therefore, we expect them to have the expertise to decide whether specific events or event series indicate an imminent threat to the blockchain. Within a permissioned blockchain, these security experts are responsible for monitoring the distributed network through the view of the local organization's blockchain node. Other organization's nodes could also be monitored, but data availability is likely limited due to access restrictions within the blockchain network.

3.3 Tasks

Visualizations or any other tool supporting the *Detect* function of the blockchain security monitoring process should be based on the tasks that the respective users need to carry out. Following the user characterization above, we derive the crucial tasks of the domain expert's work.

To illustrate the monitoring task's complexity, we show an overview of possible attacks in an attack tree notation in Figure 2. The listed attacks are based on prior work [17,18] and related literature surveys [7,9]. For each leaf on the tree, there are various ways to successfully deploy the attack, which we did not include for conciseness. The attack tree focuses on the blockchain network and nodes. Therefore, it does not include application-specific attacks such as web

Table 1. Security expert tasks and related attacks (cf. Figure 2).

Task	Description	Related attacks
T1	Identify vulnerable smart contracts	SC1, SC2, SC3
T2	Identify blockchain framework vulnerabilities	SC4, AC2
T3	Inspect log files of running services on demand	SC4, N1, N3, C3, C4
T4	Review networking activity	N1, N2, N3
T5	Compare transaction metrics over time	N2, C2
T6	Explore block and transaction history	SC1, SC2, C3, AC1
T7	Review configuration changes	C1, AC1
T8	Detect identity abuse	AC1, AC3, AC4

application vulnerabilities. Each of the shown attacks is indicated by different combinations of threat indicators [17]. Security experts need to identify threats based on these indicators as part of the *Analysis* process step. Visualizing the indicators provides the necessary overview to identify vulnerable components for in-depth analysis. Therefore, domain experts' overarching task is the *analysis of blockchain data to identify possible threats*, which is to be supported by visualizations. To allow domain experts to execute this work adequately, we have identified more specific tasks based on the attacks and corresponding threat indicators from prior work [17]. These tasks are shown in Table 1.

Each task comprises several sub-tasks that help accomplish the main task. To identify vulnerable smart contracts (*T1*), the expert may manually inspect smart contract code or scan smart contracts for vulnerabilities and inspect scan results. Identifying framework vulnerabilities (*T2*) can be accomplished by reading release notes for the framework and its dependencies. Since many anomalies can have multiple causes (i.e., low transaction throughput), log file inspection (*T3*) helps to identify the root cause of anomalies. To review networking activity (*T4*), the main indicators are the count of active connections to other peers, the activity level of those connections, and last seen times of offline peers. Transaction metrics (*T5*) include throughput, latency and unprocessed transactions. Block and transaction history monitoring (*T6*) implies watching the chain of blocks for inconsistencies such as changed blocks or missing transactions. Reviewing configuration changes (*T7*) includes both active and proposed changes to be able to intervene in case of manipulation attempts. Identity abuse (*T8*) is possible during all phases of an identity's lifecycle, so an expert must monitor issuance, usage in transactions, and revocation.

3.4 Data elements

Blockchain Frameworks such as Ethereum and Hyperledger Fabric offer a number of data sources for monitoring. The most obvious data sources are blocks and associated transaction data [23]. These can be used to derive active users, smart contracts, and the general level of activity on the network (i.e., transac-

tion throughput). Numerical data on network activity is also provided through metrics, which can be used to raise alerts for anomalous behavior. On a more technical level, each component of the blockchain node also provides log files. These files give detailed information about smart contract execution, consensus protocol violations, and other node internals. They can be helpful to determine the root cause of an anomaly.

3.5 Design Requirements

To wrap up this first part of our contribution, we derive the following general requirements for visualizations aiming to support the *Detect* function of the blockchain security monitoring process. The requirements are based on the above user, task, and data characterizations. Although we follow these requirements in the remainder of this work to design our prototype, they can serve as a general collection for respective visualization designs. We summarize the requirements under several main views that a Visual Analytics system supporting the domain experts' tasks should comprise:

R1 - General Security Information: A view should allow users to overview a series of general, security-relevant information from the monitored blockchain. Attention should be drawn to any changes on the blockchain's overall configuration (*T7*). Whenever new smart contracts are deployed to the blockchain, they should be checked (automatically or manually) for vulnerabilities. The results of these checks need to be made available for the analysts (*T1*). Additionally, newly discovered vulnerabilities within the applied blockchain framework should be shown to users within this general view (*T2*).

R2 - Network View: Another view should provide access to any data and metrics related to the peers and their network activities. This includes displaying available information about the peers themselves and the respective identities that interact with the blockchain on behalf of the peers (*T8*). This view should also provide visual access to any network-related metrics that assess the overall network's health (*T4*).

R3 - Transaction View: Domain experts need to access a view displaying information about the blocks and transactions being handled by the blockchain. This includes detailed information on the blocks and transactions themselves (*T6*) as well as a time-based view on transaction-related metrics allowing to identify any changes in typical structure and processing of transactions (*T5*).

R4 - Interactivity & Details: Any of the previously mentioned views (R1 – R3) needs to be fully interactive to provide the best possible support for domain experts' tasks and enable exploratory analysis. Whenever suspicious actions or threat indicators are identified, experts also need access to further details and underlying log files (*T3*).

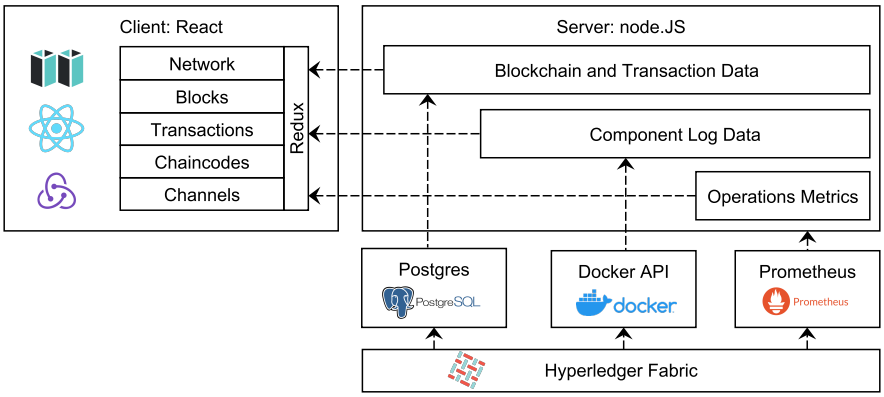

Fig. 3. Prototype architecture and data flows.

4 HyperSec: Hyperledger Security Monitoring using Visual Analytics

We now introduce our prototype **HyperSec** (**Hyper**ledger **Sec**urity Explorer), a modified version of the open-source project Hyperledger Explorer based on the design requirements introduced in Section 3.5. The prototype is open-source and available online, along with a demo deployment[2]. Our modifications address the two remaining layers of the NBGM by designing our solution based on the domain problem and implementing it within a prototype.

4.1 Architecture and Technology

We choose Hyperledger Explorer as a starting point since it already provides a working synchronization architecture based on Hyperledger Fabric's block event subscription. We extend the existing architecture to allow for more comprehensive accessibility of relevant data and effective security monitoring. This results in the architecture displayed in Figure 3. We keep the basic structure (data sources, server, and client) of the original architecture for interoperability and transparency reasons. However, in our previous study [17] we found that security-relevant information for Hyperledger Fabric must be retrieved from several data sources: the Hyperledger Fabric SDK, operations metrics, and the application logs available via Docker. Block data is already stored in Hyperledger Explorer's PostgreSQL database. We integrate additional metrics and log sources through server-side proxies to the respective Prometheus and Docker APIs. The React client accesses these through the API exposed by the Hyperledger Explorer server.

[2] `https://github.com/sigma67/hypersec`

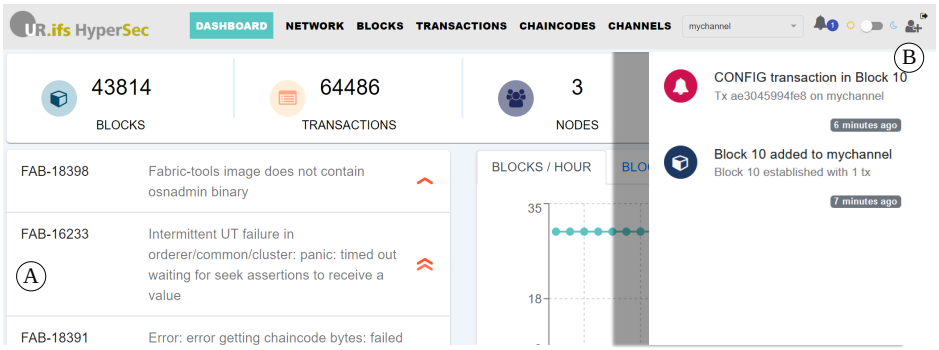

Fig. 4. *Dashboard* view: Security issues, alerts and general overview.

We implement the views defined in Section 3.5 by adapting existing views from the Hyperledger Explorer project. This allows us to retain the frontend structure while introducing new monitoring capabilities. Therefore, domain experts do not need to work with a completely new interface but rather get additional relevant information on the respective views. The updated views host a series of interactive visualizations based on the *visx*[3] visualization primitives for React. They all follow a similar structure: relevant data is retrieved from the client's *Redux* state handling, transformed for use in the visual display, mapped into visual primitives, and finally rendered [6].

4.2 Visual representations and interactions

We now go into more detail on our HyperSec prototype's visual representations addressing the requirements *R1*-*R3* and their interactivity (*R4*). As mentioned before, we integrate the visualizations into existing Hyperledger Explorer views to retain the familiar structure for domain experts. This Section is structured accordingly to the naming of the original Hyperledger Explorer views.

Views *Dashboard* and *Chaincodes*: To fulfill the Design Requirement *R1*, we adjust two views of the Hyperledger Explorer. First off, directly on the Explorer's landing page, called "Dashboard", we show a list of known Hyperledger Fabric issues of High/Highest importance from the Hyperledger JIRA[4] ordered by last updated (Figure 4**A**). Any list item can be expanded to reveal additional information about the issue. Although there is no issue category directly reflecting security issues, this information is highly relevant for *T2 – Identify blockchain framework vulnerabilities*. Additionally, there is no other source for the respective information. In the side menu (4**B**), an alert appears whenever the configuration of the monitored Hyperledger Fabric blockchain is changed (*T7*).

To allow domain experts to detect vulnerable chaincodes, we include available security scans in the respective "Chaincodes" view. Whenever a smart contract

[3] `https://airbnb.io/visx/`
[4] `https://jira.hyperledger.org`

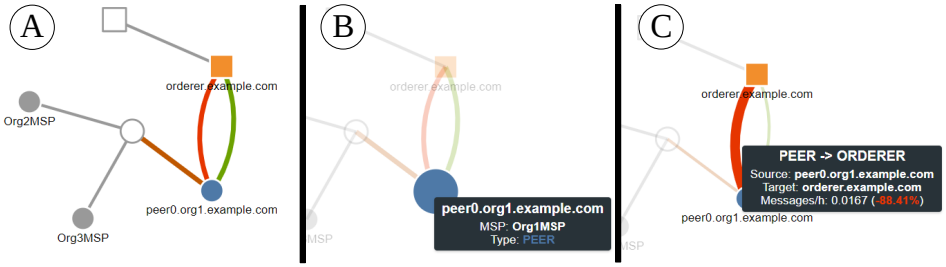

Fig. 5. *Network* view: Interactive visualization of network traffic between peers and orderers.

went through a security scan, analysts can directly check this scan's results in the HyperSec prototype (*T1*). We use the open-source static analysis tool revive-cc[5] to detect security vulnerabilities and store the scan result in the Hyperledger Explorer PostgreSQL database. To ensure the scans are up to date, we set up automated jobs to regularly generate security reports of deployed chaincodes.

View *Network*: The *Network* view targets design requirement *R2* intended for tasks *T4* and *T8*. The original Hyperledger Explorer shows a tabular list with basic information about the peers connected to the monitored Hyperledger Fabric network. In our HyperSec prototype, we extend this table with a force-directed node-link diagram to effectively visualize networking activity (Figure 5**A**). The nodes' different shapes indicate different peer types within the network: Circles are used to display peers while rectangles represent orderers. Links between the glyphs are used to display known networking activities.

However, the unavailability of core information restricts this view's expressivity. While rich information about the peers can be easily retrieved from the Hyperledger Explorer, no data about the peers' network connections is provided. Therefore, HyperSec retrieves networking information directly from Hyperledger Fabric through the Prometheus API. By doing so, experts get at least some information about the peers' networking activity within the own Membership Service Provider (MSP). However, as the Hyperledger Fabric network is decentralized, it is not possible to get any information about other MSPs' networking activities. Because of this restriction, we introduce two empty nodes in the node-link diagram (uncolored nodes in Figure 5**A**), which mark the border of the monitoring visibility regarding networking activities. Nodes within the owned MSP are colored; those within other MSPs are greyed out.

Links connecting the nodes in the graph represent known network connections. Again, outside the own MSP's borders, experts do not get much information. Therefore, we connect any foreign peer and orderer to the respective artificial node. The coloring of the links follows a continuous scale from -1 to 1. This scale measures the current deviation of the link's message traffic from the average, by comparing traffic in the last hour with traffic in the previous seven

[5] `https://github.com/sivachokkapu/revive-cc`

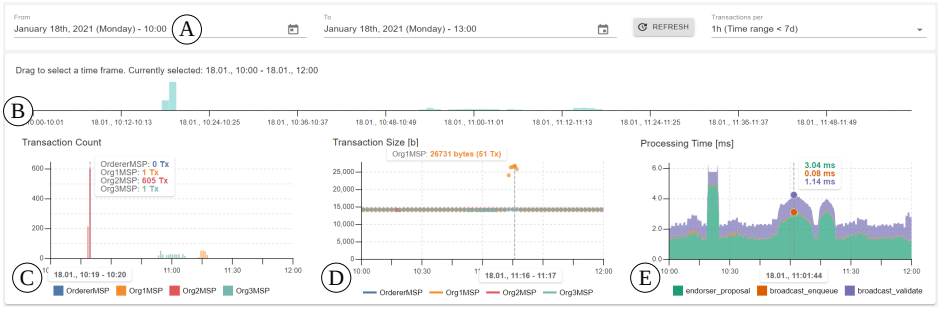

Fig. 6. *Transactions* view: Interactive visualizations for transaction count, size and processing time.

days. If this deviation is low, the link is colored in a green tone. A red link, in contrast, marks a high variation of message numbers.

The node-link diagram is fully interactive. Nodes are draggable to ensure that security analysts can adjust the layout to their own needs if necessary. Hovering over nodes (Figure 5**B**) or links (Figure 5**C**) highlights the hovered object and shows additional status information about it.

View *Transactions*: This view (*R3*) satisfies tasks *T5* and *T6*. Some modifications to the original simple table view ensure that the transactor identity and transaction size are visible. The primary adjustment we made to this view is introducing four visualizations (Figure 6). To ensure a high performance of the visualizations even when dealing with several thousands of transactions, we implement an efficient data bucketing algorithm which allows easy and fast look up of relevant transaction data (see Algorithm 1).

We make small adjustments to the original timeframe selection (Figure 6**A**). The selection defines the time range for which information about transactions should be displayed. On the right side of the timeframe selection, we added a dropdown menu to select the aggregation granularity (1 minute, 1 hour, 12 hours, 24 hours) for the visualizations. This helps security experts if they need to compare and contextualize available information.

The wide bar chart (Figure 6**B**) always displays the entire selected date range. Each bar represents the number of transactions within a specific range of time specified through the aggregation granularity. This bar chart supports analysts in navigating the selected time range. A brushing interaction (horizontal dragging on the chart) selects an even smaller time range for detailed analysis. On interaction, the other visualizations (Figure 6**C**, **D**, and **E**) and the transactions table are dynamically updated with data from this narrowed time range.

A stacked bar chart (Figure 6**C**) visualizes the number of transactions per aggregation window. However, it does this only for the transactions selected through the brushing interaction on the visualization 6**B**. It shows the transaction count's composition based on which MSP contributed how many transactions. The scatterplot 6**D** shows the transaction size in bytes throughout the

Algorithm 1: Transaction data bucketing

Input: Time Window from timestamp t_s to t_e with $t_s, t_e \in T$, $t_s < t_e$, and $T \in \mathbb{R}$. Aggregation granularity $s_b \in \mathbb{R}$

Output: Map M^{tx} with transactions sorted into the respective time-based bucket

```
1  function generateTxBuckets(t_s, t_e, s_b):
2      L^tx ← getTransactionListForTimeWindow(t_s, t_e);
3      M^tx ← new Map();
4      t_b ← t_s;
5      while t_b < t_e do
6          i_b ← ⌊t_b / s_b⌋;
7          b ← new Bucket() ;    // Object for transactions and meta-data.
8          M^tx_{i_b} ← b;
9          t_b ← t_b + s_b;
10     end while
11     foreach tx ∈ L^tx;        // Find correct Bucket and update with tx.
12      do
13         i_tx ← ⌊t_tx / s_b⌋;
14         M^tx_{i_tx} ← M^tx_{i_tx} ∪ parse(tx)
15     end foreach
16     return M^tx;
17 end
```

time range for each MSP. Each circle on the scatterplot represents the average size of transactions submitted by a specific MSP. This aggregation is performed to scale the chart for large numbers of transactions. During attacks, thousands of transactions can be submitted within just minutes, thus freezing the chart if each transaction were drawn individually. The stacked area chart 6**E** finally shows the development of three different metrics, which we identify as helpful to get an idea for the processing time in seconds that a transaction needs from proposal to validation. As information for processing times are not available distinctively per transaction but continuously per time unit, we choose to display this metrics with a continuous visualization technique.

The visualizations 6**C**, **D**, and **E** are again fully interactive. Hovering individual bars or hovering along the continuous sizes and times displays additional information as tooltips. Different metrics can also be toggled using the legend icons below the visual representations.

5 Evaluation

For our evaluation, we focus on three common attacks that cover the majority of the tasks outlined in Table 1: **SC2**, **N2**, and **AC1**. We simulate these attacks a Hyperledger Fabric test network, which the HyperSec prototype is connected to.

SC2 refers to a language vulnerability, i.e. a software bug that exposes chaincode to malicious exploits. A security expert may become aware of such an exploit by identifying vulnerable smart contracts (*T1*) and by inspecting transaction history (*T6*). For example, consider a read-after-write vulnerability detected by the chaincode scanner *revive-cc*. The security expert can inspect an automatically generated chaincode scan in the *Chaincodes* view. Intuitively, the experts check for past exploitations using the *Transactions* view. Thereto, the transactions table can be filtered using the chaincode name and applicable time frame. The filtered transactions can be inspected individually to find unusual read/write sets.

N2 refers to a distributed denial of service attack. If a peer or orderer is targeted by a traffic-based denial of service attack, its connection to other peers will be impaired as well. The *Network* view (Figure 5**C**) shows high deviation in gossip communication traffic to the targeted peer during such an attack (*T4*). If the local peer is targeted, the metrics in the *Transaction* view (Figure 6**E**) show increased transaction processing latency due to high peer load (*T5*). For attackers that can send transaction to the network, transaction-based DoS is more effective. Figure 6**C** and 6**D** show two such attempts using high transaction volume (**C**) and large transaction size (**D**). 6**E** also shows spikes in processing latency during the time of attack (spikes 1 and 3 in that chart).

To investigate the source of the anomaly, experts can check the peer logs, which are available in the Network view (*T3*). They cross-reference any error messages with open issues in the Hyperledger JIRA, which are available in the Dashboard view (*T2*).

AC1 refers to an attack where an insider abuses valid credentials for malicious purposes. Consider an insider attempting to corrupt the blockchain network's configuration using a configuration transaction. Security experts are immediately notified about the configuration change in the notification sidebar (*T7*, see Figure 4). Details of the attempted configuration change are available in the transaction history table (*T6*), where the full read-write set of the transaction is available by selecting the respective transaction.

6 Discussion

The evaluation has shown that the visualizations can assist a security expert in detecting ongoing attacks. If an attack is detected, the next steps in the Cybersecurity Framework (see Figure 1) are *analysis*, *respond* and *recover* activities, which are discussed hereafter.

Analysis. Based on the present threat indicators the expert then proceeds with *analysis* of the root cause. The logs shown in HyperSec can be a starting point, but may only show symptomatic errors. In-depth analysis of application and network logs on the systems running blockchain components can yield further information. The expert must determine if it is a crash fault or a byzantine fault. At the same time, a communication channel should be available with other

organizations of the consortium to determine if it is a more widespread problem. Guidelines and checklists can help structure this process.

Respond. Once the cause is identified, the expert contacts operations teams to request mitigation actions. Local or network configuration changes can mitigate crash faults and network/consensus threats (see Figure 2). Compromised smart contracts may require an upgrade, or even a ledger rollback if the consequences were severe. Hyperledger Fabric supports ledger snapshots for this purpose [22]).

Recover. After mitigation of an attack, evidence collection is another subject of interest. System and Docker logs are the primary source of evidence, complemented by ledger transaction data stored in HyperSec's PostgreSQL database. However, the forensic analysis of attacks on Hyperledger Fabric is a topic in need of further research.

7 Conclusion

This work introduced the task-oriented design and prototypical implementation of HyperSec, a visual analytics security monitoring tool tailored for Hyperledger Fabric. Throughout the design of HyperSec, we followed the NBGM design methodology. The domain problem describes the activities of the blockchain security monitoring process to be supported by visualizations. Subsequently, we identified the involved users, their specific tasks, and the available data elements. These considerations culminated in design requirements that apply to any visualization system aiming to support blockchain security analysts. Our prototype HyperSec picks up on these design requirements. It extends the open-source architecture of Hyperledger Explorer with additional security-relevant data sources. The data is aggregated, processed and displayed in appropriate visualizations supporting blockchain security analysts to detect potential attacks.

Our prototype might not cover every possible subtask of the defined tasks of blockchain security analysts. This is in part due to limited availability of data provided by Hyperledger Fabric itself. We plan to update our prototype as additional data sources become available in the future, and are open to contributions from the community.

The security of the monitoring tool itself is also important, as it should not contribute additional attack vectors by leaking blockchain data. During our implementation we found some bugs and vulnerabilities within Hyperledger Explorer, which we subsequently fixed and contributed to the upstream project.

References

1. Baset, S., Prehoda, B.: Hyperledger Labs Blockchain Analyzer (Mar 2021), `https://github.com/hyperledger-labs-archives/blockchain-analyzer`, original-date: 2019-05-30T11:18:45Z
2. Ben-Asher, N., Gonzalez, C.: Effects of cyber security knowledge on attack detection. Computers in Human Behavior **48**, 51–61 (2015). https://doi.org/10.1016/j.chb.2015.01.039
3. Bogner, A.: Seeing is understanding: anomaly detection in blockchains with visualized features. In: Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing. pp. 5–8. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3123024.3123157
4. Boshmaf, Y., Al Jawaheri, H., Al Sabah, M.: BlockTag: Design and Applications of a Tagging System for Blockchain Analysis. In: SEC 2019: ICT Systems Security and Privacy Protection. pp. 299–313. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-22312-0_21
5. Calder, A.: NIST Cybersecurity Framework (2018). https://doi.org/10.2307/j.ctv4cbhfx, publication Title: NIST Cybersecurity Framework
6. Chi, E.: A taxonomy of visualization techniques using the data state reference model. In: Proceedings of the IEEE Symposium on Information Visualization 2000. pp. 69–75. IEEE Computer Society (2000). https://doi.org/10.1109/INFVIS.2000.885092
7. Dabholkar, A., Saraswat, V.: Ripping the Fabric: Attacks and Mitigations on Hyperledger Fabric. In: Applications and Techniques in Information Security. pp. 300–311. Springer Singapore, Singapore (2019)
8. Di Battista, G., Di Donato, V., Patrignani, M., Pizzonia, M., Roselli, V., Tamassia, R.: Bitconeview: visualization of flows in the bitcoin transaction graph. In: 2015 IEEE Symposium on Visualization for Cyber Security (VizSec). pp. 1–8. IEEE (2015). https://doi.org/10.1109/VIZSEC.2015.7312773
9. Homoliak, I., Venugopalan, S., Reijsbergen, D., Hum, Q., Schumi, R., Szalachowski, P.: The Security Reference Architecture for Blockchains: Towards a Standardized Model for Studying Vulnerabilities, Threats, and Defenses. IEEE Communications Surveys and Tutorials (2020). https://doi.org/10.1109/COMST.2020.3033665
10. Jensen, T., Hedman, J., Henningsson, S.: How TradeLens delivers business value with blockchain technology. MIS Quarterly Executive (2019). https://doi.org/10.17705/2msqe.00018
11. Kacherginsky, P.: Attacking and Defending Blockchain Nodes. In: DEFCON 2020. p. 54 (2020)
12. Keim, D., Andrienko, G., Fekete, J.D., Görg, C., Kohlhammer, J., Melançon, G.: Visual analytics: Definition, process, and challenges. In: Lecture Notes in Computer Science (2008). https://doi.org/10.1007/978-3-540-70956-5_7, iSSN: 03029743
13. McGinn, D., Birch, D., Akroyd, D., Molina-Solana, M., Guo, Y., Knottenbelt, W.J.: Visualizing Dynamic Bitcoin Transaction Patterns. Big Data **4**(2), 109–119 (2016). https://doi.org/10.1089/big.2015.0056
14. Meyer, M., Sedlmair, M., Quinan, P.S., Munzner, T.: The nested blocks and guidelines model. Information Visualization **14**(3), 234–249 (2015). https://doi.org/10.1177/1473871613510429
15. Miksch, S., Aigner, W.: A matter of time: Applying a data–users–tasks design triangle to visual analytics of time-oriented data. Computers & Graphics **38**, 286–290 (2014). https://doi.org/10.1016/j.cag.2013.11.002

16. Munzner, T.: A nested model for visualization design and validation. IEEE Transactions on Visualization and Computer Graphics **15**(6), 921–928 (2009). https://doi.org/10.1109/TVCG.2009.111
17. Putz, B., Pernul, G.: Detecting Blockchain Security Threats. In: 2020 IEEE International Conference on Blockchain (Blockchain). pp. 313–320. IEEE (2020). https://doi.org/10.1109/Blockchain50366.2020.00046
18. Putz, B., Pernul, G.: Trust Factors and Insider Threats in Permissioned Distributed Ledgers. Transactions on Large-Scale Data- and Knowledge-Centered Systems **XLII**, 25–50 (2019). https://doi.org/10.1007/978-3-662-60531-8_2
19. Sundara, T., Gaputra, I., Aulia, S.: Study on Blockchain Visualization. International Journal on Informatics Visualization **1**(3), 76–82 (2017). https://doi.org/10.30630/joiv.1.3.23
20. The Linux Foundation: DLTLabs Case Study – Hyperledger (2020), `https://www.hyperledger.org/learn/publications/dltlabs-case-study`
21. The Linux Foundation: Hyperledger Explorer (2020), `https://www.hyperledger.org/use/explorer`
22. The Linux Foundation: Hyperledger Fabric 2.3 Documentation (2020), `https://hyperledger-fabric.readthedocs.io/en/release-2.3`
23. Tovanich, N., Heulot, N., Fekete, J., Isenberg, P.: Visualization of Blockchain Data: A Systematic Review. IEEE Transactions on Visualization and Computer Graphics p. 1 (2019). https://doi.org/10.1109/TVCG.2019.2963018
24. Zheng, P., Zheng, Z., Luo, X., Chen, X., Liu, X.: A detailed and real-time performance monitoring framework for blockchain systems. In: Proceedings - International Conference on Software Engineering (2018). https://doi.org/10.1145/3183519.3183546, iSSN: 02705257