Detecting Blockchain Security Threats

Benedikt Putz, Günther Pernul
Chair of Information Systems
University of Regensburg
Regensburg, Germany
benedikt.putz@ur.de, guenther.pernul@ur.de

***Abstract*—In many organizations, permissioned blockchain networks are currently transitioning from a proof-of-concept stage to production use. A crucial part of this transition is ensuring awareness of potential threats to network operations. Due to the plethora of software components involved in distributed ledgers, threats may be difficult or impossible to detect without a structured monitoring approach. To this end, we conduct a survey of attacks on permissioned blockchains and develop a set of threat indicators. To gather these indicators, a data processing pipeline is proposed to aggregate log information from relevant blockchain components, enriched with data from external sources. To evaluate the feasibility of monitoring current blockchain frameworks, we determine relevant data sources in Hyperledger Fabric. Our results show that the required data is mostly available, but also highlight significant improvement potential with regard to threat intelligence, chaincode scanners and built-in metrics.**

***Index Terms*—distributed ledger, permissioned blockchain, information security, security monitoring, insider threat**

I. Introduction

Enterprise applications based on Distributed Ledger Technology (DLT) are no longer just proof-of-concept. They are now used in production environments to track shipping containers across the supply chain [1], settle trade finance deals [2] and to handle the exchange of valuable aerospace parts on a decentralized marketplace [3]. These use cases present attractive targets for internal and external attackers to exploit. While blockchains attempt to thwart attackers by replicating the database and code execution, there is still no shortage of attacks, as discovered by researchers and practitioners [4]. Attacks may target the consensus algorithm, flaws in smart contract programming languages, or flaws in the blockchain framework itself. Besides vulnerabilities, there are also operational security concerns such as private key or host system compromise. Security professionals looking to protect against these attacks need to have a clear idea what threats they are facing, how to detect ongoing attacks and how to protect against them. Commonly, Security Information and Event Management (SIEM) systems are used for such tasks, but currently no such SIEM system dedicated to permissioned blockchains exists. It is challenging for off-the-shelf SIEM systems to provide an integrated overview of a blockchain network's state, since there is a large number of components within a single blockchain node [5], whose behavior also depends on nodes outside the own organization's confines.

To illustrate the contribution of this work we use Jaquith's model of IT security controls [6]. While there is plenty of literature detailing threats [7] and exposures [4] of blockchains, approaches for countermeasures are still scarce. In the model shown in Figure 1, countermeasures consist of deterrent, detective, preventative and corrective controls. While there are some *deterrent* controls built into blockchains (such as the use of hashes and signatures for integrity preservation), *detective* controls to discover threats are still scarce.

To this end, we present a comprehensive study of attacks on permissioned blockchains. Based on our findings, we develop a set of threat indicators for automated attack detection based on log data (detective control). We focus specifically on the permissioned blockchain framework Hyperledger Fabric, which is used by almost half of the world's biggest companies evaluating DLT (23 out of 50 companies with more than $1 billion valuation) [8].

In summary, we contribute to research by

- providing a **comprehensive overview of possible attacks** on permissioned blockchains
- developing a **set of blockchain threat indicators** for attacks on permissioned blockchains
- investigating the **feasibility of monitoring attacks on Hyperledger Fabric** and identifying areas for future research and development

The remainder of this paper is structured as follows. We first provide an overview of related work in Section II, before defining the threat model for monitoring in Section III. In Section IV we present a study of possible attacks on permissioned blockchain networks, and define threat indicators for each attack. A suitable data processing architecture to gather

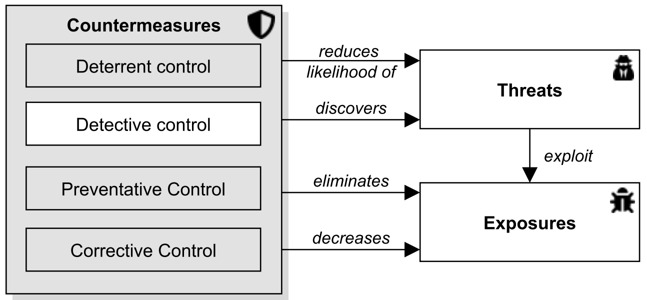

Fig. 1. Logical model of IT Security Controls [6], with the missing DLT detective control highlighted

these indicators from blockchain data sources is defined in Section V. We evaluate the developed monitoring approach by investigating the feasibility of indicator collection with Hyperledger Fabric in Section VI. Finally, we discuss SIEM and organizational integration of blockchain monitoring in Section VII.

II. Related Work

Currently, the majority of proposed monitoring tools and methods focus on isolated detection of anomalies within specific components of a DLT node. BAD [9] is designed to detect anomalous Bitcoin transactions to prevent transactions with malicious payloads from spreading in the network. LedgerGuard [10] monitors a Hyperledger Fabric peer's ledger and restores corrupted blocks from connected peers if needed. Garcia et al. propose Lazarus [11], a solution for diversity management of consensus nodes. Lazarus monitors the software stack of each blockchain node for newly found or zero-day exploits and quarantines affected replicas. A number of tools exist for formal verification and vulnerability detection of Ethereum smart contracts, including both offline scanners [12], [13] and online detection frameworks [14].

There are also several tools focusing on performance and availability monitoring. Hyperledger Caliper [15] focuses on performance benchmarking in an isolated testing environment. Hyperledger Explorer [16] displays basic information about running nodes, allows users to inspect blockchain state and interact with deployed chaincode. However, its main purpose is browsing activity on the underlying blockchain network, not monitoring the network for security threats.

Compared to these approaches, the present work provides a holistic perspective on blockchain security monitoring, instead of only focusing on isolated aspects. Based on a comprehensive survey of attacks on permissioned distributed ledgers, we determine suitable threat indicators and corresponding data sources to compute them. Going beyond plain status monitoring offered by tools like Hyperledger Explorer, we focus on extracting security-relevant threat indicators from Hyperledger Fabric.

III. Threat Model

Before going into detail on threat indicators, a threat model of possible attackers and attacks needs to be established. In this Section we provide a brief overview of DLT actors and DLT-specific threats (split into vulnerabilities and malicious intent), followed by a detailed enumeration of attacks in Section IV.

A. Actors

As a first step, we model the actors in a blockchain network. Figure 2 shows a data flow context diagram containing relevant actors. Each actor may cause a threat, either by acting as a malicious insider or as an external attacker. *Transactors* are regular users, which have read access to the blockchain and can submit transactions. *Peer*, *Orderer* and *Certificate Authority (CA) Admins* administrate the corresponding blockchain components and have special privileges. *External Users* have no privileges and are locked out of the system by access

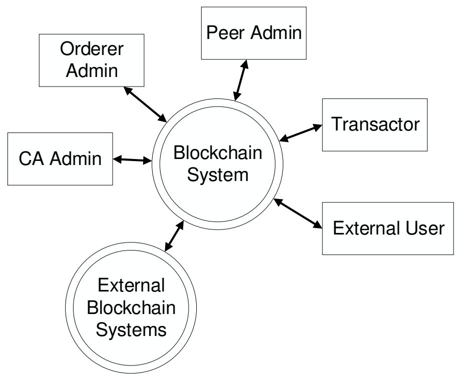

Fig. 2. Level 0 data flow diagram of blockchain actors

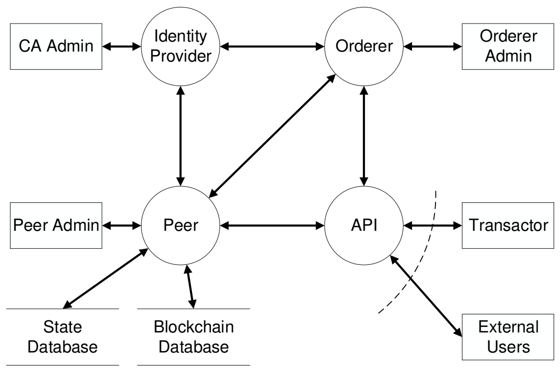

Fig. 3. Level 1 data flow diagram of blockchain actors

control, barring vulnerabilities. Figure 3 shows a Level 1 data flow diagram detailing how these actors interact with the processing nodes within the blockchain system (modeled after Hyperledger Fabric). Each *Admin* controls the corresponding blockchain component. Outside access is provided to *Transactors* and *External Users* via an API.

B. Vulnerabilities

Vulnerabilities increase the exposure to threats by providing attackers with ways to compromise the protection goals confidentiality, integrity and availability. Like any software, distributed ledger frameworks are prone to software bugs, which may result in vulnerabilities. A prime example are **vulnerable protocols**, caused by implementation bugs in cryptographic, networking or storage components or dependencies. For example, Hyperledger Fabric uses gRPC for exchanging blocks, which has been subject to a number of high and critical severity Common Vulnerability Enumerations (CVEs)[1]. Another type of exposure are **vulnerable contracts**, where the intricacies of smart contract development can lead to exploitable behavior [13].

Distributed ledger frameworks also offer a large number of configuration options. Since these options are rarely doc-

[1]see https://nvd.nist.gov/

umented in a single place, negligence or oversight may lead to **misconfiguration** by administrators. Bad configuration and a lack of consideration for security on deployment of blockchain nodes then increases exposure to attacks.

C. Malicious intent

Internal threats. Distributed Ledgers are subject to a variety of threats by insiders and external insiders [17], who may attempt to exploit the blockchain for personal or organizational gain. Besides inadvertent misconfigurations, insiders may intentionally manipulate the configuration of blockchain peers and threaten network security. Through initiation of updates for smart contracts they may introduce vulnerabilities or backdoors. For these reasons, administrators are potential single points of failure for an organization's blockchain node and their actions should be monitored by an independent information security team.

External attackers. External adversaries may attempt to gain blockchain network access in order to read and possibly manipulate ledger data. To this end, attackers may exploit the aforementioned vulnerabilities. A successful attack could result in subversion of a blockchain peer or identity provider, which is a prerequisite for many attacks only possible from inside the network. Denial of service attacks are an additional attack vector, which may stall consensus if sufficient nodes are affected.

IV. Attacks and Threat Indicators

To gain an overview of relevant attacks, we conducted a literature review of attacks on permissioned blockchain systems. We searched for the terms `("permissioned blockchain" OR "hyperledger fabric") AND ("vulnerability" OR "attack")`, using the ACM digital library (106 results), SpringerLink (361 results), ScienceDirect (218 results), IEEE Xplore (10 results) and the Wiley Online Library (55 results). We filtered these results for works dealing with attacks and vulnerabilities on permissioned blockchains, which left us with 10 papers. The low number of filtered results can be attributed to the fact that many papers mention or cite existing vulnerabilities and attacks, but do not contribute new vulnerabilities. We conducted additional in-depth research on each of the found attacks by searching for the attack's name, which yielded additional literature [4], [7], [18].

We extract all attacks applicable to permissioned blockchains from the surveyed papers. For conciseness, some attacks are grouped under a common term (i.e. Contract Vulnerabilities). Based on the identified attacks and affected blockchain components, we develop threat indicators that allow a security expert to recognize ongoing attacks. While we focus on applicability of the attacks on Hyperledger Fabric, many of the attacks are applicable to other permissioned blockchain frameworks as well. The categorized overview of attacks is shown in Table I. Generally, threat indicators may be *proactive* or *reactive*. Proactive approaches attempt to detect the vulnerability before exploitation, while reactive approaches detect the act of exploitation and attempt to limit the damage.

For the remainder of this chapter, we go into more detail on each attack and how it can be detected with suitable indicators. Based on the threat model, we first focus on *vulnerabilities* followed by attacks of *malicious intent*. These indicators are then elaborated in Section VI with regard to Hyperledger Fabric.

A. Vulnerabilities

Contract Vulnerability. A contract vulnerability refers to a security bug in a smart contract that must be fixed through a contract upgrade. Since an organization may not have control over all contracts that it is sending transactions to, it is important to also monitor contracts owned by other organizations on the network. In general, contract vulnerabilities are difficult to detect, since abuse transaction patterns vary depending on the vulnerability. For example, the *Re-entrancy attack* on Solidity smart contracts [19] may be detected by excessive resource consumption from a single transaction. Other vulnerabilities may require inspection of contract state changes, such as the *delegatecall injection* [13]. Yamashita et al. provide an overview of Hyperledger Fabric chaincode risks related to non-determinism, phantom state database reads and unchecked inputs [24]. Unchecked inputs may for example result in JSON injection vulnerabilities [25]. By scanning each contract deploy/upgrade transaction, new vulnerabilities can be derived from in the scan logs. The number of *scanned potential vulnerabilities* in deployed contracts is a useful indicator for monitoring and reducing the attack surface of smart contracts.

Framework Vulnerability. The code of the blockchain framework may be subject to vulnerabilities. This category includes vulnerabilities such as *insufficient smart contract virtualization* [20], [26] and *injection of malicious code* due to improper input checking [26]. Correspondingly, *framework releases* should be monitored for such vulnerabilities to upgrade to new versions as soon as possible.

Dependency Vulnerability. Blockchain frameworks also rely on a number of *direct* and *transitive* dependencies. A major category of dependencies are database systems used for storage of blockchain state. Most blockchain frameworks use self-sufficient DBMS such as LevelDB, CouchDB or Postgres [17]. As dependencies, their versions are often updated infrequently and insufficient default configurations are used. For example, the CouchDB instance preconfigured with Hyperledger Fabric was found to be susceptible to direct unauthenticated manipulation by via the built-in web interface, voiding integrity assumptions of the framework [20]. This vulnerable default configuration is also present in the latest test-network provided with Fabric 2.1. While this should be prevented by securely configuring the dependency initially, monitoring *configuration changes* and *database container logs* for foreign IP access would also detect exploitation.

Cryptographic Vulnerability. Vulnerabilities in cryptographic protocols are a serious threat, since blockchain frameworks rely on hashes and digital signatures for integrity,

TABLE I
OVERVIEW OF ATTACKS AND CORRESPONDING THREAT INDICATORS (BOTH VULNERABILITIES AND MALICIOUS INTENT).

Attack Category	**Attack Examples**	**Threat Indicators**	**Type**
Contract Vulnerability	Reentrancy [19], delegatecall [13], Dependency injection [17]	scanned potential vulnerabilities threat intelligence on vulnerabilities	Proactive Proactive
Framework Vulnerability	Unrestricted Chaincode Containers [20]	framework releases	Proactive
Dependency Vulnerability	CouchDB web interface [20]	threat intelligence on vulnerabilities dependency container logs	Proactive Reactive
Cryptographic Vulnerability	Quantum Computing Threat [17], Hash Collision Resistance Attack [17]	threat intelligence on vulnerabilities	Proactive
Denial of Service	Dust transactions [4], Storage pollution [20]	transaction throughput transaction latency incoming network messages oustanding transactions	Reactive
Network Partitioning	BGP hijacking [21], DNS attacks [4], [22], Eclipse attack [4], [22], Attack of the Clones [21]	connected peers	Reactive
Malicious Consensus Behavior	Consensus Delay [4], Alternative History [7], [20], Block Withholding [4], Transaction Reordering [20]	discarded blocks latest block hashes leader election frequency outstanding transactions (age) client application outgoing transactions	Reactive
Consensus Configuration Manipulation	Batch Time attack [20], Block Size attack [20]	configuration changes configuration value bounds	Proactive
Identity Provider Compromise	CA Attack [20], [22], Sybil attacks [18], [20], [23], Boycott attack [20], Blacklisting attack [20]	certificate requests (successful/denied) certificate revocations transactor identities	Reactive

authentication and non-repudiation. If SHA256 were to be affected by a collision-resistance attack similar to the one discovered for SHA-1 (CVE 2005-4900), most major blockchain frameworks would be affected [17]. Detecting such an attack is only possible by *monitoring threat intelligence*, i.e. new CVEs in the NIST database.

B. Malicious Intent

Denial of Service (DoS). While blockchain systems are less threatened by DoS than centralized servers due to built-in replication and fault-tolerance, targeted attacks still represent a threat. To achieve DoS, an outsider may attempt to flood specific or multiple blockchain peers with TCP syn packets. In particular, DoS attacks targeting the consensus leader can significantly reduce or stall consensus entirely [17]. Internal attackers with access to the network can simply send a large number of transactions (or transactions with a large size), which must all be processed by endorsing peers [4]. Even if they are invalid they are included in the blockchain by all peers, polluting the storage [20]. Consequently, indicators for DoS attacks are low *transaction throughput* and high *transaction latency* [27]. Since these metrics are also affected by other factors (such as network usage by regular Transactors), monitoring *incoming network messages* and *incoming transactions* provides a more comprehensive picture.

Network Partitioning. Internal attackers may attempt to partition a blockchain network by manipulating network routing [4]. The goal of such an attack is manipulation of the consensus protocol. This can be accomplished through network-level attacks such as *BGP hijacking* [21] and *DNS attacks* [4], [22]. This attack can then be followed up with consensus manipulation attacks. Examples of attacks based on network partitioning include the *Attack of the Clones* on the Proof-of-Authority consensus protocols [21][2] and *Eclipse attacks* [4]. In permissioned networks, monitoring the number of *connected peers* can be used to detect network partitioning attempts.

Malicious Consensus Behavior. Orderer Admins can launch a variety of attacks by behaving maliciously during consensus. A *Consensus Delay* attack can be initiated by peers propagating invalid blocks [4]. The *Intentional Fork* attack in Hyperledger Fabric describes a similar scenario where the ordering service sends out conflicting versions of blocks to peers [20]. Both attacks result in consensus delay since the blockchain peers waste computing power on verifying invalid blocks. They can be detected by monitoring the number of *discarded blocks* that were received through peer-to-peer communication.

If several Orderer Admins collude, they may attempt to rewrite the blockchain in an *Alternative History* attack [7], [20]. This requires $> 50\%$ of nodes to collude (crash-fault tolerance), or $> 2f$ nodes for byzantine-fault tolerant consensus where $3f + 1$ nodes tolerate f malicious nodes. To detect any state forks or attempts at rewriting history, the *latest block hashes* of all peers must be monitored and compared.

Byzantine attacks become possible when a non byzantine-fault tolerant (BFT) consensus algorithm is used. Hyperledger Fabric 2.1 only offers Kafka and Raft implementations, which

[2]applicable to permissioned networks based on Ethereum

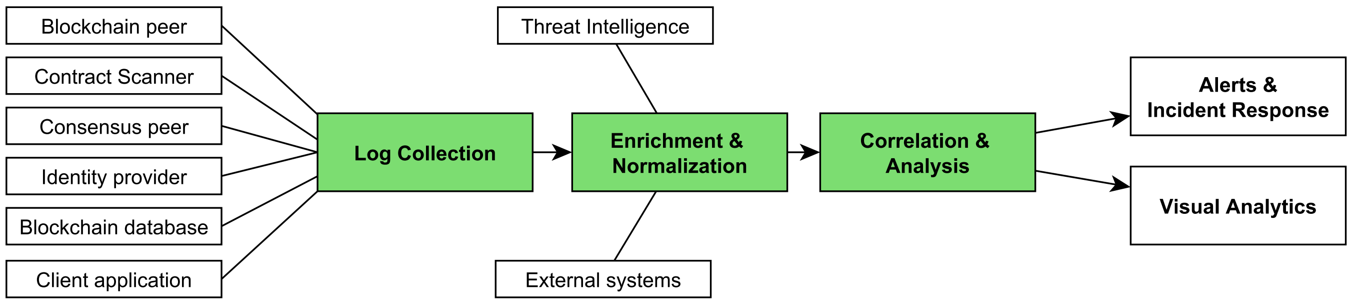

Fig. 4. Proposed Blockchain Security Monitoring pipeline.

are merely crash-fault tolerant. A malicious Raft node may prevent consensus indefinitely by constantly starting new leader elections, or cause correctness violations if elected as leader [28]. Leader election misbehavior can be detected by monitoring *leader election frequency*.

A malicious leader (also possible in BFT consensus [4]) is more difficult to detect, since it may cause different types of correctness violations. For example, during a *Block Withholding* [4] or Sabotage [20] attack the consensus leader or ordering service witholds blocks containing unwanted transactions, or transactions from specific participants. This attack can be detected by monitoring the *age of outstanding transactions* in the transaction pool. Transactions with a large age indicate that the orderer cluster is not reaching consensus on them.

Another example of a correctness violation is a *Transaction Reordering* attack, where the leader of the ordering service reorders transactions to favor specific organizations [20]. Orderer Admins might abuse this to gain an advantage in smart contracts where timing is critical. If an organization relies on such timing-critical contracts, it should track *client application outgoing transactions*. When the transaction is eventually included in a block, reordering can be detected by comparing timestamps.

Consensus Configuration Manipulation. If an attacker controls a majority of consensus nodes, it becomes possible to manipulate the consensus process by changing configuration values. For the Hyperledger Fabric Ordering service, the *Batch Time attack* and *Block Size attack* are known [20]. Both can delay transactions indefinitely by increasing the time until transactions are included in a block. To mitigate this threat, *configuration changes* should be monitored to detect unsafe configuration values outside of specified bounds before they are approved by Peer Admins.

Identity Provider Compromise. In Hyperledger Fabric, identity providers are referred to as Membership Service Providers (MSPs) and the default implementation is called Fabric-CA. A Fabric-CA MSP may be compromised through private key theft, also referred to as a *CA Attack* [22]. The actual theft of private keys cannot be detected from the blockchain framework's perspective, but malicious actions using these keys can be discovered. A frequently-cited example are *sybil attacks*, where a single attacker forges multiple identities [18], [23], i.e. with certificates from a compromised MSP. These identities may be used to circumvent contract endorsement policies, and thus manipulate contract execution [20]. Sybil attacks can be detected by closely monitoring *newly issued certificates*. Since this may not be possible for certificates issued by external MSPs, *Transactor identities* should also be monitored.

The *Boycott attack* refers to a scenario where two organizations are under the same MSP and one of them is denied new certificates [20]. Consequently, the CA should be monitored for *denied certificate requests*. The *Blacklisting Attack* is based on revoked certificates and may result in peers or Transactors losing network access [20], [23]. Since certificate revocations should normally occur rarely, the *number of revoked certificates* is of interest.

V. Data Collection and Processing

To assemble the threat indicators developed above, a pipeline for data collection and processing is needed. The goal of data processing is to provide an aggregated view of the threat indicators, enabling an expert to detect security threats.

We derive the data processing pipeline from the SIEM pattern [29]. Initially, data is collected from internal blockchain data sources (*log collection*) and enriched with external data. Subsequently it must be normalized to a common data format (*enrichment & normalization*). The following steps are outside the scope of this paper, but briefly discussed in Section VII. Correlating multiple indicators (*Correlation & Analysis*), visualizing them (*Visual Analytics*) and triggering alerts (*Alerts & Incident Response*) provide valuable aid to security experts.

Log Collection. Logs are collected from various sources producing different types of information: numeric metrics (i.e. transactions per second), application log events and blockchain state events. Depending on the framework they are retrieved via *push* or *pull* mechanisms. A summary of data sources for Hyperledger Fabric is shown in Table II. The Operations Service provides numeric metrics about current operation, which can be consumed by a Prometheus (pull) or a StatsD instance (push). Peer channel-based event services allow an agent to subscribe to channel-specific block data (push), which also includes application-level chaincode events. Log data is provided by the Docker containers of the Hyperledger Fabric components, which provide logs of configurable log level detail (such as `INFO` and `DEBUG`). This includes auxiliary services

TABLE II
HYPERLEDGER FABRIC 2.1 DATA SOURCES AND METRICS [30].

Data	**Type**	**Source**	**Collection**
Numeric Metrics	Numeric	Prometheus (Peer, Orderer, MSP)	Pull
Numeric Metrics	Numeric	StatsD (Peer, Orderer, MSP)	Push
Channel Events	Application data	SDK (Peer)	Push
Logs	Behavioral	Docker (Peer, Chaincode, Orderer, MSP, CouchDB)	Pull
Blockchain state/history	Application data	SDK (Peer)	Pull

such as Fabric-CA (identity provider) and CouchDB (state database provider). Finally, the full spectrum of blockchain data is available via the Peer SDK, but data must be queried on demand (pull). In addition to these built-in data sources, chaincode vulnerability scanners provide log files for ingestion.

Enrichment & Normalization. Events occurring within the organization are enriched with contextual data from external systems. If accessible, data from connected blockchain peers' APIs should be collected. Such external peer data can help determine whether an incident is isolated or network-wide.

Depending on the blockchain setup, additional data sources may be needed. There are several optional blockchain features that involve external systems:

- **Permissionless Blockchain Anchoring**: The anchoring status needs to be monitored in case the anchoring chain is broken. Periodic API requests to a node on the anchoring target blockchain can provide the required data.
- **Oracles**: Oracles provide external data to smart contracts. If an oracle is manipulated or compromised the consequences can be severe for the relying contract. Thus, data provided by oracles should be monitored for anomalies.
- **Cross-chain interactions**: Cross-chain interactions such as hash-timelocked contracts are used to exchange assets between blockchains. The status of these asset swaps on blockchains other than the primary monitoring target should be tracked, in case there is an issue with contracts on either side of the swap.
- **Off-chain storage**: Blockchain applications often link data on the blockchain via hashes for timestamping and non-repudiation purposes. The availability of the linked data should be periodically checked.

VI. EVALUATION

To validate the detectability of the indicators outlined above in Table I, we conducted experiments with a Hyperledger Fabric deployment. We inspected log files and data sources hands-on to determine whether attacks can actually be detected with currently provided data sources. Hereafter, we elaborate for each attack which data source is suitable, and how these data sources could be improved to enhance detectability.

Scanned potential vulnerabilities. To our knowledge, only two vulnerability scanners exist for Hyperledger Fabric, both closed-source and only supporting Go chaincode (Chaincode Scanner[3] and an unnamed tool [24]). The coverage of these tools can be augmented by language-specific ones such as gosec[4]. However, there is a clear need for open-source chaincode scanners with support for all chaincode languages.

Threat intelligence on vulnerabilities. To be able to detect new vulnerabilities and upgrade network nodes as soon as possible, threat intelligence feeds need to be monitored constantly for vulnerabilities relevant to blockchain components. However, searching the NIST National Vulnerability database for "Hyperledger Fabric" yields 0 results. As of today, there are no blockchain-specific sources of threat intelligence information. Nevertheless, threat intelligence feeds can be monitored for relevant keywords (i.e. *gRPC*, *CouchDB*, *Golang*, *SHA256*) and manually filtered for applicable threats. Additional sources can be community collaboration tools for open-source frameworks (i.e. monitoring the Hyperledger JIRA[5] or mailing list for the keyword *security*).

Framework releases. Hyperledger Fabric provides a GitHub feed with the latest releases[6]. Each release has sections on known and resolved vulnerabilities, which can be used to determine if a timely upgrade is needed.

Dependency Container Logs. If CouchDB is used with Hyperledger Fabric, all requests are logged to the Docker container logs including the IP address, time and request URL. This information can be used to detect suspicious requests from foreign IP addresses and other types of attacks.

Transaction throughput. Throughput can be observed based on Fabric's ordering service metric `broadcast_processed_count`, which counts the number of processed transactions over time. Alternatively, this metric can be computed based on the transactions contained in block data, which is needed for other metrics (i.e. latest block hashes).

Transaction latency. Transaction latency can be computed as the delay between transaction timestamp and the block timestamp of the transaction's block. This metric is thus computed for each transaction upon block inclusion. By subscribing to Hyperledger Fabric's Channel Events, this metric can be computed for each transaction in each block signed by the ordering service.

Incoming network messages. The operations services provides numerous metrics for monitoring network communication. `gossip_comm_messages_received` tracks messages received via peer gossip, and the `grpc_server_*` metrics track gRPC communication with clients.

[3]https://chaincode.chainsecurity.com/

[4]https://github.com/securego/gosec

[5]https://jira.hyperledger.org

[6]https://github.com/hyperledger/fabric/releases

Outstanding transactions. To our knowledge, there is no metric that keeps tracks of outstanding unprocessed transactions. This would be beneficial to determine the cause of unprocessed transactions (high system load or deliberate exclusion). For transaction delay attacks, the operations service provides the histogram metric `blockcutter_block_fill_duration`, which monitors the time from transaction enqueuing at the orderer to inclusion in a block. By monitoring transactions fill duration over time, the average age of outstanding transactions can be determined.

Connected peers. Ordering service outgoing connections can be measured using the metric `cluster_comm_egress_tls_connection_count`, while peer connections are evident from `grpc_comm_conn_opened`. While isolated disconnections may also be caused by a benign crash fault, multiple can indicate a network partition.

Discarded blocks. The peer container logs provide information about received and validated blocks, which can be used to determine which blocks were successfully validated. Since this requires preprocessing, a built-in metric would be desirable.

Leader election frequency. For Hyperledger Fabric's Raft consensus, the metric `consensus_etcdraft_leader_changes` provides a counter for leader changes, and orderer log files indicate which node was elected. Tracking leader changes over time can provide an indication of potential attacks. It would be beneficial if the consensus algorithm also indicated failed leader elections, to be able to detect nodes continuously proposing elections.

Latest block hashes. Block hashes can be computed based on the block headers, which are obtained through channel block event subscription. A block hash in Hyperledger Fabric is the SHA256 hash of the block number, the block data hash and the previous block's hash. Tracking the correctness of block hashes over time provides certainty that no blockchain reorganization has occurred.

Client application outgoing transactions. Outgoing transactions signed by a Hyperledger Fabric SDK contain the transaction hash and timestamp. These can be sent to and tracked by a monitoring service (push).

Configuration changes. Since timely notification about configuration updates is essential, this metric should also be derived through block subscription to the Channel Event Hub. Configuration transactions are identified by the type `CONFIG_UPDATE`. Additionally, the operations service provides the metric `consensus_etcdraft_config_proposals_received`, which keeps tracks of configuration transaction proposals.

Certificate requests and revocations. When using Fabric-CA server, certificate creation can be monitored by counting POST requests to `/enroll` (`[INFO]` log level). Successful requests have status code 2xx, while denied requests are evident from status codes 4xx and 5xx. For details such as the name of the registered identity, `[DEBUG]` log level is required. Revocations are monitored using log entries for requests to the `/revoke` endpoint.

Transactor identities. The node.JS SDK provides a way to subscribe to new blocks of a channel (`registerBlockEvent` method). The listener receives all transactions as part of the block, and each transaction contains the full sender certificate as part of the signature. Using these certificates, a monitoring system can keep track of existing identities.

VII. Discussion

After demonstrating practical feasibility of our concept, we now take a broader perspective and focus on integration with existing security monitoring facilities. This includes SIEM systems, but also Security Operations Centers (SOCs), which interpret data provided by monitoring systems.

A. SIEM integration

The monitoring architecture shown in Figure 4 can be implemented by connecting the data sources in Table II to SIEM software. In the next step, the available data needs to be normalized to a common format. Based on normalized data, correlation rules can be derived that pinpoint the source attack of the issue with certainty. If sufficient data is available to indicate an ongoing attack, security alerts can be triggered to prompt a detailed investigation. Thereto there is also significant potential to help experts interpret blockchain data and metrics by providing appropriate visualizations, for example as part of a monitoring dashboard.

B. Organizational aspects

Figure 5 shows how a Blockchain Security Monitor (shortened *BCSM*) interacts with human and software components. The security operations team monitors activities on the blockchain network and identifies the origin of potential threats. The *BCSM* only monitors blockchain components running on systems controlled by its operating organization. Data from nodes controlled by other organizations in the consortium may contribute to a complete overview of the threat situation, but cannot be relied upon.

To be able to detect manipulation by blockchain administrators, the security operations team must be separate from the blockchain operations team. It assumes a watchdog role that can help prevent insider attacks. *BCSM* is the technical

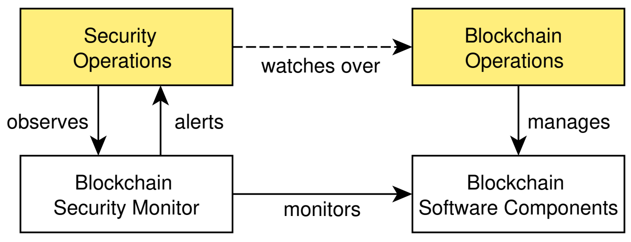

Fig. 5. Organizational integration of blockchain security monitoring.

component that supports this organizational function with informationa and alerts.

VIII. Conclusion

In this paper, we presented a review of attacks on permissioned blockchains. Based on the attacks, we developed indicators for an individual organization in a blockchain network to detect ongoing attacks, either proactively or reactively depending on the attack. For these metrics, we proposed a suitable data processing architecture inspired by SIEM systems. The feasibility of this architecture was demonstrated by closely examining the data sources that Hyperledger Fabric offers.

We also identified several areas of improvement for blockchain security. For example, adding metrics such as discarded blocks and failed leader elections to Hyperledger Fabric facilitates security monitoring. Additionally, open-source threat scanners for Hyperledger Fabric chaincode and blockchain-specific threat intelligence feeds can help patch vulnerabilities before they can be exploited. Finally, potential for malicious consensus behavior could be significantly reduced by introducing an ordering service based on BFT consensus for Hyperledger Fabric.

References

[1] T. Jensen, J. Hedman, and S. Henningsson, “How TradeLens delivers business value with blockchain technology,” *MIS Quarterly Executive*, 2019.

[2] IBM, “we.trade | IBM,” 2020. [Online]. Available: https://www.ibm.com/case-studies/wetrade-blockchain-fintech-trade-finance

[3] The Linux Foundation, “Honeywell Case Study – Hyperledger,” 2020. [Online]. Available: https://www.hyperledger.org/resources/publications/honeywell-case-study

[4] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and A. Mohaisen, “Exploring the Attack Surface of Blockchain: A Systematic Overview,” 2019.

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys ’18. New York, NY, USA: ACM, 2018, pp. 30:1–30:15.

[6] A. Jaquith, *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional, 2007.

[7] I. Homoliak, S. Venugopalan, Q. Hum, D. Reijsbergen, R. Schumi, and P. Szalachowski, “The Security Reference Architecture for Blockchains: Towards a Standardized Model for Studying Vulnerabilities, Threats, and Defenses,” 2019.

[8] M. del Castillo and M. Schifrin, “Blockchain 50,” 2020. [Online]. Available: https://www.forbes.com/sites/michaeldelcastillo/2020/02/19/blockchain-50

[9] M. Signorini, M. Pontecorvi, W. Kanoun, and R. D. Pietro, “BAD: Blockchain Anomaly Detection,” *CoRR*, vol. abs/1807.0, 2018. [Online]. Available: http://arxiv.org/abs/1807.03833

[10] Q. Zhang, P. Novotny, S. Baset, D. Dillenberger, A. Barger, and Y. Manevich, “LedgerGuard: Improving Blockchain Ledger Dependability,” pp. 1–8, 2018. [Online]. Available: http://arxiv.org/abs/1805.01081

[11] M. Garcia, A. Bessani, and N. Neves, “Lazarus: Automatic Management of Diversity in BFT Systems,” in *Proceedings of the 20th International Middleware Conference*, ser. Middleware ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 241–254. [Online]. Available: https://doi.org/10.1145/3361525.3361550

[12] M. D. Angelo and G. Salzer, “A Survey of Tools for Analyzing Ethereum Smart Contracts,” in *IEEE International Conference on Decentralized Applications and Infrastructures, DAPPCON 2019, Newark, CA, USA, April 4-9, 2019*. IEEE, 2019, pp. 69–78. [Online]. Available: https://doi.org/10.1109/DAPPCON.2019.00018

[13] H. Chen, M. Pendleton, L. Njilla, and S. Xu, “A Survey on Ethereum Systems Security: Vulnerabilities, Attacks and Defenses,” *CoRR*, vol. abs/1908.0, 2019. [Online]. Available: http://arxiv.org/abs/1908.04507

[14] T. Chen, R. Cao, T. Li, X. Luo, G. Gu, Y. Zhang, Z. Liao, H. Zhu, G. Chen, Z. He, Y. Tang, X. Lin, and X. Zhang, “SODA: A Generic Online Detection Framework for Smart Contracts,” in *27th Annual Network and Distributed System Security Symposium, NDSS 2020*. The Internet Society, 2020.

[15] The Linux Foundation, “Hyperledger Caliper,” 2020. [Online]. Available: https://www.hyperledger.org/use/caliper

[16] ——, “Hyperledger Explorer,” 2020. [Online]. Available: https://www.hyperledger.org/use/explorer

[17] B. Putz and G. Pernul, “Trust Factors and Insider Threats in Permissioned Distributed Ledgers,” *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, vol. XLII, pp. 25–50, 2019. [Online]. Available: http://link.springer.com/10.1007/978-3-662-60531-8_2

[18] D. Dasgupta, J. M. Shrein, and K. D. Gupta, “A survey of blockchain from security perspective,” *Journal of Banking and Financial Technology*, 2019.

[19] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts (SoK),” in *Lecture Notes in Computer Science*, 2017.

[20] A. Dabholkar and V. Saraswat, “Ripping the Fabric: Attacks and Mitigations on Hyperledger Fabric,” in *Applications and Techniques in Information Security*, V. S. Shankar Sriram, V. Subramaniyaswamy, N. Sasikaladevi, L. Zhang, L. Batten, and G. Li, Eds. Singapore: Springer Singapore, 2019, pp. 300–311.

[21] P. Ekparinya, V. Gramoli, and G. Jourjon, “The Attack of the Clones Against Proof-of-Authority,” in *27th Annual Network and Distributed System Security Symposium, NDSS 2020*. The Internet Society, 2020.

[22] A. Davenport, S. Shetty, and X. Liang, “Attack Surface Analysis of Permissioned Blockchain Platforms for Smart Cities,” in *2018 IEEE International Smart Cities Conference, ISC2 2018*, 2019.

[23] J. Holbrook, “Blockchain Security and Threat Landscape,” in *Architecting Enterprise Blockchain Solutions*. Wiley, 2020, pp. 323–347. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119557722.ch11

[24] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun, “Potential Risks of Hyperledger Fabric Smart Contracts,” in *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2019, pp. 1–10.

[25] S. Riedesel, P. Hakimian, K. Buyens, and T. Biehn, “Tineola: taking a bite out of enterprise blockchain,” 2018. [Online]. Available: https://github.com/tineola/tineola

[26] G. Shaw, “Hyperledger Fabric Security Audit,” Nettitude, Tech. Rep., 2017. [Online]. Available: https://wiki.hyperledger.org/display/fabric/Audits?preview=/2393550/2393584/technical_report_linux_foundation_fabric_august_2017_v1.1.pdf

[27] N. Andola, Raghav, M. Gogoi, S. Venkatesan, and S. Verma, “Vulnerabilities on Hyperledger Fabric,” *Pervasive and Mobile Computing*, 2019.

[28] C. Copeland and H. Zhong, “Tangaroa: a Byzantine Fault Tolerant Raft,” 2016.

[29] M. Vielberth and G. Pernul, “A Security Information and Event Management Pattern,” in *12th Latin American Conference on Pattern Languages of Programs (SLPLoP)*, nov 2018. [Online]. Available: https://epub.uni-regensburg.de/41139/

[30] The Linux Foundation, “Hyperledger Fabric 2.1 Documentation,” 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.1