A Blockchain-Based Retribution Mechanism for Collaborative Intrusion Detection

Wenjun Fan[1,4], Shubham Kumar[2], Sang-Yoon Chang[3,4], and Younghee Park[2,4](✉)

[1] School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou 215123, Jiangsu, People's Republic of China
wenjun.fan@xjtlu.edu.cn

[2] Computer Engineering Department, San José State University, San José, CA 95192, USA
{shubham.kumar,younghee.park}@sjsu.edu

[3] Department of Computer Science, College of Engineering and Applied Science, University of Colorado Colorado Springs, Colorado Springs, CO 80918, USA
schang2@uccs.edu

[4] Silicon Valley Cybersecurity Institute (SVCSI), San José, CR 95192, USA

Abstract. Collaborative intrusion detection approach uses the shared detection signature between the collaborative participants to facilitate coordinated defense. In the context of collaborative intrusion detection system (CIDS), however, there is no research focusing on the efficiency of the shared detection signature. The inefficient detection signature costs not only the IDS resource but also the process of the peer-to-peer (P2P) network. In this paper, we therefore propose a blockchain-based retribution mechanism, which aims to incentivize the participants to contribute to verifying the efficiency of the detection signature in terms of certain distributed consensus. We implement a prototype using Ethereum blockchain, which instantiates a token-based retribution mechanism and a smart contract-enabled voting-based distributed consensus. We conduct a number of experiments built on the prototype, and the experimental results demonstrate the effectiveness of the proposed approach.

Keywords: Blockchain · Collaborative intrusion detection · Retribution · Detection signature · Verification · Token

1 Introduction

A collaborative intrusion detection system (CIDS) [17] can share security information (e.g., the intrusion signature) across multiple domains to gain a collaborative intelligence for intrusion detection. A CIDS can have a global view for large networks or IT ecosystems in contrast to the standalone IDS that only monitors the intrusion events occurring at one place. In the context of CIDS, there are two architectures [18]: centralized and distributed. A centralized CIDS often applies a centralized server to collect and share the security information to the distributed IDS, and such a centralized server is often deployed in cloud [5,7]. However, the centralized server often suffers the single point of failure, and the cloud is also honest-but-curious. On the contrary, a distributed CIDS relies on a peer-to-peer (P2P) network to propagate the security information. However, the distributed CIDS has to cope with other security problems, e.g., it needs to ensure the integrity of the data transmission and to build the trust among the participants. For addressing those issues, several blockchain-based CIDSes have been proposed [6,11,13,16].

The blockchain-based CIDSes often use permissioned blockchain, which means any participant must be registered to the authority which could be a centralized CA [11,16] or a distributed PKI [6]. Also, one blockchain-based CIDS often involves distributed consensus protocol to verify and agree the propagated viable transmissions. Most consensuses of the CIDSes focus on resisting against the insider attacks, e.g., a number of participants collude together (or are controlled by the attacker) to issue/verify the malicious transmissions across the whole P2P network. To address this, the blockchain P2P network can use practical Byzantine Fault Tolerance (pBFT)-based voting to achieve the n-compromise resistance [6]. However, this research area lacks an approach to verify the efficiency of the shared detection signature.

A *detection signature* is a malicious data pattern or attack rule that is compared with current behavior to decide if is that of an intruder. The unverified but widely adopted detection signature not only costs the CIDS resource but also impacts the coordinated defense, because some participants may yield false alerts due to the incorrect/inefficient detection signature.

In this paper, we propose a blockchain-based retribution mechanism to verify the propagated detection signature for CIDS. The following properties of the blockchain typically motivate our approach: i) *decentralization* includes distinct autonomous participants which is consistent with the nature of the CIDS; ii) *digital currency* provides the financial nature to incentivize the participants to contribute to donating and verifying detection signatures; iii) *consensus* enables the participants of the P2P network to reach agreement based on a distributed manner; iv) *smart contract* provides the programmability to create specific consensus for processing certain payload of transaction; v) *permissioned blockchain* controls the participation of the blockchain network.

The retribution mechanism aims to reward the participants who donate efficient detection signatures and punish the participants who share inefficient detection signatures. Thus, the retribution mechanism introduces an incentive to the participants to take part in the coordinated defense by sharing efficient detection signatures. The contributions of this paper are summarized as follows:

– A blockchain-based retribution mechanism is proposed, which is used for incentivizing the participants to contribute to the collaborative intrusion detection system.

- An Ethereum blockchain-based prototype is implemented for validating the approach, whereby the Ethereum token is applied as the incentive.
- The corresponding experiments are conducted to show the effectiveness of the proposed approach.

The rest of the paper is organized as follows: Sect. 2 reviews the related work; Sect. 3 presents our models including the system network model and the threat model that our approach builds on; Sect. 4 proposes the design of the blockchain-based retribution mechanism; Sect. 5 presents the Ethereum blockchain-based prototype implementation; Sect. 6 shows the experimental results; Sect. 7 concludes the paper.

2 Related Work

In this section, we review the related work including the existing retribution mechanisms in the context of blockchain P2P network and the detection signature verification methods in the IDS research area.

2.1 Retribution Mechanism

The well-known retribution mechanism in the context of Blockchain P2P network is IKP [12], which provides an instant Karma mechanism to the distributed PKI based on permissionless cryptocurrency blockchain. The major contribution of IKP is that it proposes a resilient mechanism to provide incentives to the CAs to perform correctly and to the detectors to report unauthorized certificates. IKP is used to detect the unauthorized certificates for a domain due to CA misbehavior in terms of certificate policies that specify automatic responses. However, IKP still involves the mining process which is consistent with Nakamoto consensus provides natural financial incentives for the participants in the permissionless blockchain P2P network, in contrast to that, our approach is used for the permissioned blockchain without mining process.

Apart from monetary incentive based retribution mechanism, some other approaches consider using reputation to reward or punish the participants. Thus, we also consider the Proof-of-Reputation (PoR) as related work where the reputation serves as incentive to push the nodes to participate in the distributed consensus [8,14,19]. PoR is used to replace the Proof-of-Work (PoW) consensus essentially to reduce the heavy computation power requirement and increase the scalability of the blockchain P2P network, while PoR itself should resist against the attacker whose objective is to get high reputation. The PoR-based consensus often allows the node having the highest reputation to sign the block, or uses the reputation to weight the vote of the node. That is different from our approach, as we encourage every node to share the new detection signature, and also the reputation is not used to weight the node's vote.

2.2 Detection Signature Verification

There are a number of solutions and research about verifying the detection signature in order to reduce the false positive in the context of signature-based NIDS, e.g., to track and analyze the protocol status code of the response to see if it is valid or unexpected [10]; to use the context knowledge, i.e., the protected network and system configuration to verify the alters [4]; to correlate IDS alarm with network vulnerability [15] using vulnerability scanning tool like Nessus [1]. Another case in point is the challenge-based approach, which evaluates a node's detection correctness by sending challenges and receiving the corresponding feedback [11]. In particular, ATLANTIDES [3] is a notable approach that uses an automatic anomaly-based analysis of the system output, which provides useful context information regarding the network services. In other words, it takes an approach by correlating the anomalies detected on the output with the alerts raised by the NIDS monitoring the input traffic, and built on that, it can discard a number of the latter as being false positive alerts. More specifically, a communication's incoming traffic triggers the signature-based NIDS's alarm, and then ATLANTIDES uses anomaly-based IDS to monitor the outgoing traffic of that communication. If the anomaly-based IDS against outgoing traffic alarms, that is a true positive detection signature, otherwise, that is a false positive one.

All the above approaches focus on verifying the truthfulness of the detection signature, while out work stresses on the efficiency of the detection signature.

3 Models

This section presents the models that our approach is based on, which includes the system network model and the threat model respectively.

3.1 System Network Model

In our approach, the P2P network is decentralized with collaborative detection communication to gain a coordinated defense. We define the (virtual) network boundary that comprises standalone intrusion detection engine nodes across geographical domains. Thus, the system network boundary can be considered as one autonomous system (AS) overseeing multiple domains, or as a single-domain with multiple networking inbound points, e.g., firewalls or security-enabled controllers in SDN network [6]. With this definition, we assume that there is no outsider attacks threatening the coordination communication between the participants, e.g., the Denial-of-Service (DoS) attack making the node unable to participate the distributed consensus [9]. Therefore, we focus on the insider attacks which compromise the participants to share bogus information and manipulate the consensus. Those participants will comprise a permissioned blockchain P2P network, and once such a network is created, the set of peers is static within the defined network boundary.

3.2 Threat Model

The threat model stresses on the insider attacks rather than the outsider attack corresponding to the system network model mentioned above. Hence, the DoS attack from outside against the availability of the participant nodes is out of the scope. Also, we do not consider the insider-and-outsider collusion case, whereby an insider adversary does not share newly discovered detection signature to other participants while inform the zero-day vulnerability to the outsider adversary so that the outsider adversary can exploit the vulnerability.

The objective of an insider adversary is to take malicious actions to disrupt the coordinated defense. More specifically, the insider adversary aims to deliver inefficient detection signatures across the P2P network to enforce other participants to download the inefficient detection signature to update their detection rule set ineffectively. Except for the single node's bad actions, the insider attack often involves the network-wide compromise to manipulate the consensus. That means the adversary has compromised enough participants and can mislead the entire system to make a wrong decision.

The following items are the potential attack vectors that an insider adversary can leverage, which our approach is designed to defend against:

1. **Flooding**: A peer keeps sending tremendous malicious/inefficient detection signatures to other peers to waste their computation resource.
2. **Malicious injection**: A peer submits a malicious/inefficient detection signature to the other coordinated peers, which increases the false alert ratio.
3. **Collusive verification**: Multiple peers collude together to manipulate the distributed consensus to affect the verification result. The objectives of this vector can be i) to share and verify a malicious/inefficient detection signature (*collusive injection*); ii) or to defame a valid/efficient detection signature (*collusive defamation*).

4 Design

This section proposes the system architecture overview, the efficiency calculation method, the distributed verification consensus, and the retribution mechanism as follows.

4.1 System Architecture Overview

An overview of the system architecture is presented by Fig. 1. In our approach, the blockchain is permissioned, and a registered peer can play two alternative roles, *donor* or *verifier*. Also, according to our system network model mentioned in Sect. 3.1, the donor and verifiers can monitor different domains/networks.

When the peer is a donor, it submits detection signature to the P2P network; as a verifier, the peer needs to verify the received detection signature to report the verification result to indicate the efficiency of the detection signature. With

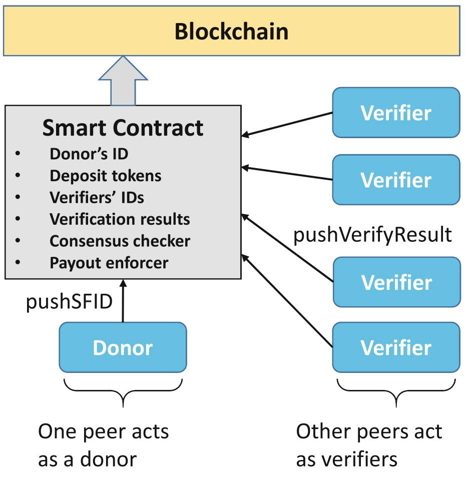

Fig. 1. An overview of the architecture design: it includes n peers; when 1 peer plays as a donor, the rest of the peers play as verifiers.

that, for each detection signature transmitted, there must be a donor and the rest peers become verifiers in nature. We define the efficiency of a detection signature as the presence of the detection signature across the verifiers. If a donor circulates a detection signature, the other peers as verifiers need to examine the presence of the detection signature separately. That means each verifier checks its local rule-sets to see if the examined detection signature exists or not. Thus, if a detection signature presents more times across the peers, it has the higher efficiency. We propose a retribution mechanism (see Sect. 4.4) to incentivize the peers to contribute to verifying the efficiency of the detection signature.

In Fig. 1, assuming there are five peers in the blockchain P2P network, when one of them acts as a donor to push a detection signature file ID (SFID), other peers will play as the verifiers to download and verify the new detection signature separately. It is worth noticing that in practice, people do not directly submit the detection signature file to the blockchain, since storing big size file (like the detection signature file) in blockchain ledger is not economical. Thus, we employ the InterPlanetary File System (IPFS) [2] to store the raw detection signature file, whereby a SFID will be given by IPFS. Thus, the donor just submits the SFID representing the corresponding detection signature to the blockchain, and the verifiers can use the SFID to request to the IPFS for downloading the detection signature for verification.

The distributed verification carried out by multiple verifilers is asynchronous, since every domain has its own context. When a verifier gets a verification result, it needs to report the result to the smart contract. Meanwhile, the consensus checker of the smart contract keeps track of the reports from the verifiers. Once the results achieve a consensus threshold, the smart contract can make the consensus decision and execute the payout enforcer to perform the retribution.

4.2 Efficiency Calculation Method

The verification method is that the verifier examines the presences of the downloaded detection signature in its own rule set (assuming one presence denotes that the verifier convinces itself that the detection signature is valid by certain checking approach like [3,4,15]). Thus, we define the peer/participant as p, the rule set of p_i as $\mathcal{R}_i$, where $i \in \{1, 2, ..., 3n + 1\}$ (when $3n + 1$ is the total number of the peers participating in the P2P network to support the n-compromise resistance), the detection signature as s, the count of the presence of s in $\mathcal{R}_i$ as $x_i \in \mathcal{Z}$. Thus, if $s \notin \mathcal{R}_i$, $x_i = 0$; if $s \in \mathcal{R}_i$, $x_i \geqslant 1$. Also, we define the total number of the rules in $\mathcal{R}_i$ as y_i, and the efficiency of s calculated by p_i as f_{s_i}. Thus, f_{s_i} can be calculated by the following Eq. (1):

$$f_{s_i} = \frac{x_i}{y_i} \times 100\% \tag{1}$$

All the verifiers need to report the efficiency of s from their own perspectives respectively to the smart contract. If $f_{s_i} \geqslant 50\%$, the smart contract considers that s is efficient for p_i; whereas if $f_{s_i} < 50\%$, the smart contract considers that s is inefficient for p_i. The smart contract counts the number of p who reports the efficiency of s which is greater than or equal to 50%, meanwhile, counts the number of p who reports the efficiency of s which is less than 50%. These two counted numbers will be used to make an unanimous decision based on a distributed consensus (see Sect. 4.3) as an overall verification result to s.

4.3 Distributed Verification Consensus

The distributed verification consensus presented here is used to ensure the distributed verification of the efficiency of the propagated detection signature, which is different from the distributed consensus used for mining the block in permissionless cryptocurrency. Hence, we use the practical Byzantine Fault Tolerance (pBFT)-based voting to sustain the distributed verification consensus, which provides an "n-compromise resistance" against the up to n participants compromise where we have $3n + 1$ participants in total. In other words, our distributed verification consensus is based on a majority decision, whereby even 1/3 of the participants are compromised to perform collusive verification to make an inefficient s become efficient, the whole P2P network can still work and resist against such an insider attack.

It is worth noting that Ethereum itself does not use pBFT as the consensus algorithm for processing the transactions/blocks, while we use pBFT with smart contract just for voting (and verifying) for the detection signature file proposed by any peer. With using the pBFT-based voting for the distributed verification consensus, the smart contract only makes the unanimous decision saying that s is efficient when more than 2/3 of the total participants reporting that s is efficient. Otherwise, s is judged as inefficient. Thus, our system is strict to decide an efficient s (i.e., must have more than 2/3 participants agree) while is loose to decide an inefficient s (i.e., only needs at least 1/3 participants agree).

Table 1. Operations of the retribution scheme vs. the threats.

Operation	Threat
Deposit (donor)	Flooding
Penalty (donor)	Malicious injection
Reward (donor)	Malicious injection
Reward (verifiers)	Collusive verification

In addition, once the consensus is done, there must be a majority group and a minority group, unless the number of participants agreeing on the efficiency of s equals the number of participants agreeing on the inefficiency of s. We define the total number of verifiers which report result as n_v, the number of the majority verifiers as n_{major} and the number of the minority verifiers as n_{minor}. Thus, $n_v = n_{major} + n_{minor}$. In the event that s is efficient, the majority group must includes the participants supporting the efficiency of s, and the minority group must consist of the participants reporting the inefficiency of s. In contrast, if s is verified as an inefficient one, the majority group and the minority group could include either kind of participants respectively, i.e., the majority group could be comprised of the participants supporting the inefficiency of s while the minority group could consist of the participants supporting the efficiency of s, or vice versa.

4.4 Retribution Mechanism

The retribution mechanism represents the reward and punishment methods incentivizing the participants to contribute to donating and verifying the detection signature. To this end, we define three sorts of token-based operations: *deposit*, *penalty* and *reward*. Table 1 summarizes these operations with the threats (mentioned in Sect. 3.2) resisted against. These operations are detailed as follows.

Deposit. The deposit operation indicates that one donor must deposit a number of tokens accompany with the submission of s. We define the number of the deposited tokens as d. Such an operation can effectively prevent the submission flooding launched by the malicious donor, as now any donor has to deposit a certain number of tokens for submission, meaning the submission is not free.

Penalty. The penalty operation towards the donor means that if the submitted s turns out to be inefficient based on the consensus result reported by the verifiers, then d will be deducted completely. Such as operation can effectively mitigate the malicious injection including inefficient s, since if the submitted s is inefficient, the deposit will be deducted as a penalty to the donor.

Reward (donor). The reward operation towards the donor denotes that if the submitted s is verified as an efficient one, the donor will get the deposit back and also get a number of extra tokens as reward for its contribution. We define the number of the reward tokens for the donor as r_d. In our case, we specify $r_d = d$, which means the donor will get $2d$ back as reward, and that actually awards the donor the number of tokens for one extra submission attempt. Such an operation can incentivize the donor to submit efficient s in order to get the deposit and the extra reward back.

Reward (verifier). Regarding the reward for the verifiers, we define the verifier as v_i where $i \in \{1, 2, ...3n\}$ (as there is one out of the $3n+1$ participants already playing as a donor), the number of the reward tokens for all the verifiers as r_v, which equals the number of the deposited tokens from the donor, i.e., $r_v = d$. If the verifier belongs to the majority group, it will get a maximized reward, defined as r_{v-max}, and if the verifier belongs to the minority group, it will get a minimized reward, defined as r_{v-min}. Note that $r_v = r_{v-max} + r_{v-min}$. In addition, if $n_{major} = n_{minor}$, r_v will be divided by n_v, which then is allocated to every v_i equally, though this kind of event occurs occasionally. With that, we define the reward per majority verifier as $r_{per-major}$, the reward per minority verifier as $r_{per-minor}$, and the reward per verifier as r_{per-v} (when $n_{major} = n_{minor}$). Such an operation can incentivize the verifiers to report their verification results, and also can resist against the collusive verification when the malicious verifiers intend to make an inefficient s become an efficient s, unless the attacker controls more than 2/3 verifiers of the whole P2P network[1].

With the above mentioned reward operations for the verifiers, we can have several payout cases. For example, we specify that $d = r_d = r_v = 100$, and there $\exists v_i$ reporting the verification result. Therefore, we can have the following reward payout cases.

Case 1. When there is no v_i in the minority group, i.e., $n_{minor} = 0$, we specify $r_{v-min} = 0$ so that $r_{v-max} = 100$. Hence, each v in the majority group will get $r_{per-major} = \frac{100}{n_{major}}$ tokens as reward.

Case 2. When there is v_i in the minority group while $n_{minor} \neq n_{major}$, we specify $r_{v-min} = 1$ so that $r_{v-max} = 99$. Thus, each v in the majority group will get $r_{per-major} = \frac{99}{n_{major}}$ tokens as reward, and each v in the minority group will get $r_{per-minor} = \frac{1}{n_{minor}}$ tokens as reward.

Case 3. When $n_{minor} = n_{major}$, each v that reports verification result will get $r_{per-v} = \frac{100}{n_v}$ tokens equally as reward.

Algorithm (1) presents the verifier's reward payout algorithm, which includes the above three cases.

One example of the approach regarding verifying a detection signature as efficiency is illustrated by Fig. 2. In this instance, the P2P network has 5 participants. One p as a donor submits an SFID called "DS-file1" with 100 tokens as deposit to the smart contract. The other four participants play as v to verify the efficiency of the detection signature. Among these verifiers, three of them report

[1] By contrast, to make an efficient s become inefficient, the collusive verification only needs more than 1/3 verifiers of the whole P2P network. In our system, we consider the "inefficiency becomes efficiency" more dangerous than the other way round.

Algorithm 1. Verifier's Reward Payout Algorithm

```
Require: 100 tokens deposited
Ensure: $\exists v_i$ reporting verification result
 1: if $n_{major} == n_{minor}$ then
 2:     $r_{per-v} = \frac{100}{n_v}$
 3: else
 4:     if $n_{minor} == 0$ then
 5:         $r_{per-major} = \frac{100}{n_{major}}$
 6:     else
 7:         $r_{per-major} = \frac{99}{n_{major}}$
 8:         $r_{per-minor} = \frac{1}{n_{minor}}$
 9:     end if
10: end if
```

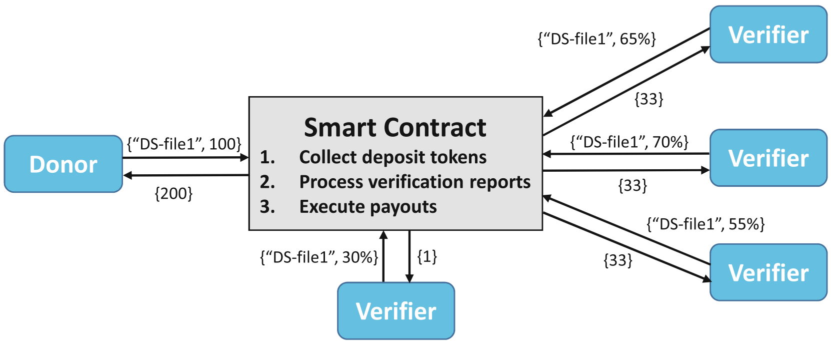

Fig. 2. Sample interactions between the participants in our approach when a detection signature is judged as efficiency.

the efficiency of DS-file1 greater than 50% (i.e., 65%, 70% and 55% respectively), while one of them reports that this detection signature is inefficient since the efficiency is only 30%. Upon that, the smart contract makes the consensus decision in terms of the majority results and judges DS-file1 as an efficient detection signature. Thereafter, the smart contract returns the deposit 100 tokens and an extra reward with another 100 tokens, i.e., 200 tokens altogether, to the donor as it contributes an efficient detection signature to the whole system. Also, the smart contract rewards the verifiers who report the verification results according to the reward payout algorithm (see Algorithm (1)). Therefore, the smart contract splits the 99 tokens proportionally in terms of the number of the majority verifiers, i.e., 33 tokens per majority verifier, while still gives 1 token to the minority verifier who claims that the detection signature is inefficient.

Because the design principal of the retribution mechanism is to encourage every participant to work for verification, even though the result makes a verifier belong to the minority group, the verifier still gets a minimized reward, but surely the result making the verifier belong to the majority group will lead to a maximized profit for the verifier.

5 Implementation

This section presents the prototype including the system implementation and the token values setup for validating the proposed approach.

For implementing the system, the participant nodes are deployed on CloudLab with the virtual machine (VM) running Ubuntu 18-64-STD by default setting. On each node, we install Ethereum (Geth version 1.9.16-stable) and Solidity (version 0.5.16) for solidity files compilation. Thus, we use those nodes to generate a real premissioned blockchain P2P network. Truffle (version 5.1.34) is used to deploy the complied smart contracts on the locally created Ethereum blockchain network. We also use web3.js (version 1.2.1) and truffle-contract (version 4.0.31) to interact with the smart contract functions from our JavaScript files. Also, we employ IPFS to physically store the real detection signature files while only spread SFID on the blockchain.

The voting smart contract mainly consists of three functions, i.e., push SFID, consensus, and reward payout. The push SFID function is used to allow the donor to submit the detection signature to the blockchain P2P network, the consensus function aims to tally up the verification results sent by the verifiers, and the reward payout function is implemented in compliance with the retribution mechanism.

Furthermore, regarding the token values setup in a permissioned blockchain, the smart contract is initialized with a high amount of tokens to ensure that the system has enough funds for rewarding the participants who make contribution and in turn ensuring that the system can keep functioning for a long term. For instance, the smart contract is initialized with 10 million tokens, and each participant in the P2P network is credited with 1000 tokens initially by the smart contract. The donor node has to firstly deposit 100 tokens with the smart contract before it can write a SFID to the smart contract. Also, we specify the extra reward for the donor as 100 tokens and the total reward for the verifiers as 100 tokens as well.

In addition, the number of participant nodes (including donor and verifier) could vary from 10 to 100 in our experiment setting. More details are presented in the following experimental results.

6 Experiments

The section shows that several experiments are conducted built on the prototype implementation, and the corresponding experimental results are presented for evaluating our approach.

6.1 Computation Performance

First of all, we test the computation performance for carrying out different tasks based on the prototype. Table 2 shows the computation overhead including CPU usage, memory usage and execution time in terms of the corresponding tasks.

Table 2. Performance of different tasks on blockchain.

Task	CPU (%)	MEM (MB)	Exec. Time (ms)
Run geth	1.22	305.814	–
Deploy smart contracts	3.12	685.013	1683
Push SFID	4.72	68.36	144.63
Consensus	5.31	75.89	208.77
Reward payout	4.94	62.52	178.04

The blockchain network is initiated by running the Ethereum client (Geth) and connecting the nodes with each other for forming the P2P network. This Geth process continues running and takes 1.22% of the CPU and 305.814 MB of the memory. Once the Ethereum nodes are initialized and connected with each other, the smart contracts are then deployed to the blockchain network. This process takes 3.12% of the CPU and 685.013 MB of the memory as it covers many steps like the compilation of the smart contracts, initialization of smart contracts, and finally deployment to the blockchain network. This process lasts 1683 ms (around 1.6 s).

The push SFID task utilizes around 4.72% of the CPU, takes 68.36 MB of the memory, and spends around 144.64 ms for execution. Comparably, the reward payout task utilizes around 4.94% of the CPU, takes 62.52 MB of the memory, and spends around 178.04 ms for execution. The consensus task takes CPU utilization of 5.31%, which is greater than the push SFID task and the reward payout task, as the consensus task has higher complexity involved, i.e., it has to keep track of the verification results sent by all the verifiers and maintain the track of the participants belonging to the majority group and the minority group respectively. The memory usage of the consensus task is also higher than the push SFID task and the reward payout task, since the consensus task involves maintaining the variables and array for storing the incoming efficiency results from the verifiers, and also, it stores the information about the participant IDs of the majority group and the minority group. Consequently, the consensus task spends more time around 208.77 ms for execution.

6.2 Reward per Majority Verifier

We emulate the reward tokens for each majority verifier when there is no minority verifiers and the submitted detection signature is verified as efficient. Figure 3 shows the result. It is apparent that the more majority verifiers involved, the less reward each majority verifier can obtain. We can see that when the majority group includes only 10 verifiers, each one can gain 10 tokens as reward, however, when the majority group includes 100 verifiers, each one can only gain 1 token as reward. That result informs that the deposit should be reconciled with the scale of the P2P network, less deposit will decrease the verifiers' incentive to perform verification.

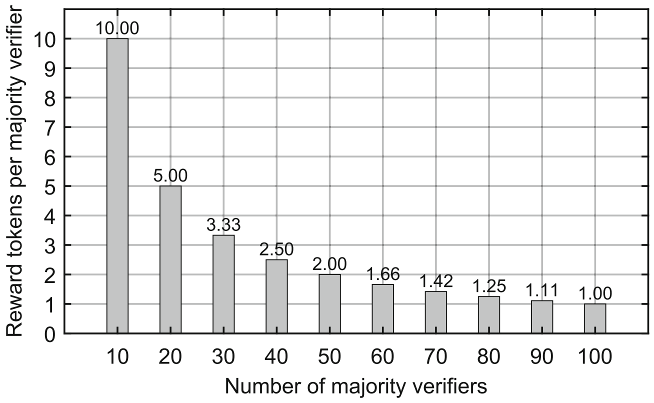

Fig. 3. Reward per majority verifier.

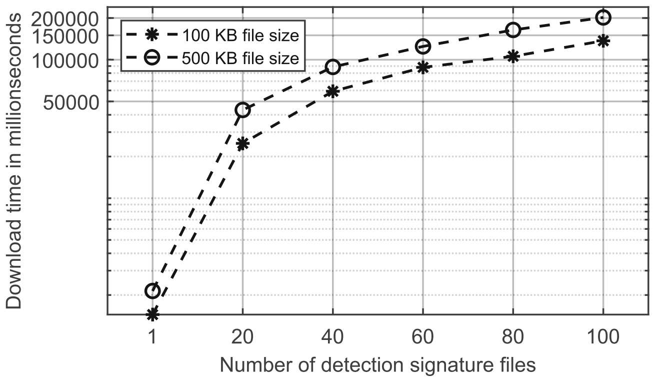

Fig. 4. Detection file download time cost.

6.3 Detection File Download Time Cost

Figure 4 shows the time cost for downloading the detection files from IPFS. We measure two sorts of files, i.e., one sort that the file size is 100 KB, and the other sort that the file size is 500 KB. Note that in our system, each file just contains one detection signature, and so the file sizes selected here are just in terms of the typical cases in practice. When the verifier only needs to download one detection signature file, it takes 1,445 ms for the 100 KB file size and 2,139 ms for the 500 KB file size on average respectively. It is obvious that the bigger file costs more time for downloading. For example, when 100 verifiers go to download one hundred 500 KB files, it takes 201,869 ms (∼202 s) altogether, by contrast, when they go to download one hundred 100 KB files, it only takes 136,909 ms (∼137 s) altogether. Further, according to the figure, we can see that the file-download time for the 500 KB file size increases much faster than the 100 KB file size, as the slope of the curve with the 500 KB file size is much steeper.

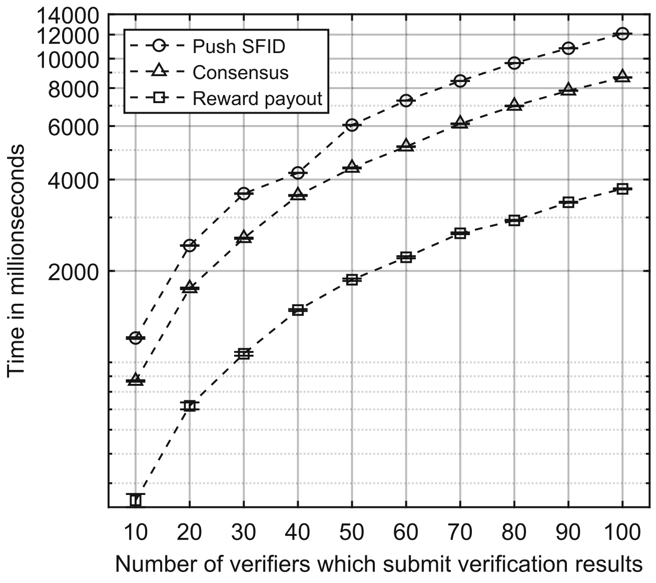

Fig. 5. Networking overheads: Push SFID vs. Consensus vs Reward payout.

Thus, the detection signature file size should keep as small as possible, otherwise it will cost more time for the whole verifiers to download it. Though the verifiers download the detection signature file from IPFS in parallel, the networking overhead is aggregated and increases dramatically.

6.4 Distributed Networking Overhead

Figure 5 presents the networking overheads in terms of the increasing number of verifiers. It includes three curves, i.e., the push SFID time cost, the consensus time cost and the reward payout time cost using average values with 95% confidence interval. We can see that when there are 10 verifiers, it spends 1202.63 ms for carrying out the push SFID task, 868.318 ms for performing the consensus task, and 351.052 ms for taking the reward payout task. Also, when there are 100 verifiers on the P2P network, it spends 1208.6 vs. 8672.6 ms vs. 3726.28 ms, respectively.

The push SFID task takes the largest networking overhead. This is because during the push SFID process, the smart contract first checks whether the donor node has enough tokens for deposit or not. After checking, it stores the SFID value in temporary storage and emits an event notifying all the other participants in the network that a new SFID has been pushed to the smart contract. This further triggers the consensus process.

The networking overhead of the consensus task is greater than the reward payout task, as the smart contract has to keep track of the efficiency percentage result sent by each participant. Also, in the consensus process, the smart contract keeps track of the number of participants in both the majority group and the minority group.

In addition, the reward payout task takes the least networking overhead, since it utilizes the computation result calculated by the consensus function about the participant IDs which come under either the majority group or the minority group. The reward payout function then uses this information stored by the consensus process to distribute the tokens among the majority group and the minority group accordingly.

7 Conclusion

Collaborative intrusion detection system (CIDS) has a global view against the network attack, whereby the shared security information including the detection signature can help to build a coordinated defense. However, the existing CIDSes often neglect the efficiency of the shared security information, which actually not only consumes the storage resource and wastes the computational process, but also brings security problems as the inefficient detection signature will raise the false alert ratio. In this paper, we proposed a blockchain-based retribution mechanism to incentivize the participants to verify the efficiency of the shared detection signature for the CIDS. Thanks to the Ethereum blockchain financial nature and the smart contract programmability, we facilitated the incentivization by using Ethereum tokens and the distributed consensus with the smart contract pBFT-based voting. The prototype and the corresponding experiments demonstrated the effectiveness of the proposed approach.

Acknowledgment. This work was supported in part by XJTLU Research Development Funding RDF-21-02-012 and XJTLU Teaching Development Funding TDF21/22-R24-177.

References

1. Anderson, H.: Introduction to NESSUS. Retrieved from Symantec (2003)
2. Benet, J.: IPFS-content addressed, versioned, P2P file system (DRAFT 3). arXiv preprint arXiv:1407.3561 (2014)
3. Bolzoni, D., Crispo, B., Etalle, S.: ATLANTIDES: an architecture for alert verification in network intrusion detection systems. In: LISA, vol. 7, pp. 1–12 (2007)
4. Chaboya, D.J., Raines, R.A., Baldwin, R.O., Mullins, B.E.: Network intrusion detection: automated and manual methods prone to attack and evasion. IEEE Secur. Priv. **4**(6), 36–43 (2006)
5. Chadwick, D.W., et al.: A cloud-edge based data security architecture for sharing and analysing cyber threat information. Futur. Gener. Comput. Syst. **102**, 710–722 (2020)

6. Fan, W., Park, Y., Kumar, S., Ganta, P., Zhou, X., Chang, S.Y.: Blockchain-enabled collaborative intrusion detection in software defined networks. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 967–974 (2020)
7. Fan, W., et al.: Enabling privacy-preserving sharing of cyber threat information in the cloud. In: 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 74–80 (2019)
8. Gai, F., Wang, B., Deng, W., Peng, W.: Proof of reputation: a reputation-based consensus protocol for peer-to-peer network. In: Pei, J., Manolopoulos, Y., Sadiq, S., Li, J. (eds.) DASFAA 2018. LNCS, vol. 10828, pp. 666–681. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91458-9_41
9. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 51–68 (2017)
10. Zhou, J., Carlson, A.J., Bishop, M.: Verify results of network intrusion alerts using lightweight protocol analysis. In: 21st Annual Computer Security Applications Conference (ACSAC 2005), pp. 10–126 (2005)
11. Li, W., Wang, Yu., Li, J., Au, M.H.: Towards blockchained challenge-based collaborative intrusion detection. In: Zhou, J., et al. (eds.) ACNS 2019. LNCS, vol. 11605, pp. 122–139. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29729-9_7
12. Matsumoto, S., Reischuk, R.M.: IKP: turning a PKI around with decentralized automated incentives. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 410–426 (2017)
13. Putra, G.D., Dedeoglu, V., Pathak, A., Kanhere, S.S., Jurdak, R.: Decentralised trustworthy collaborative intrusion detection system for IoT. In: 2021 IEEE International Conference on Blockchain (Blockchain), pp. 306–313 (2021)
14. Qin, D., Wang, C., Jiang, Y.: RPchain: a blockchain-based academic social networking service for credible reputation building. In: Chen, S., Wang, H., Zhang, L.-J. (eds.) ICBC 2018. LNCS, vol. 10974, pp. 183–198. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94478-4_13
15. Subba, B., Biswas, S., Karmakar, S.: False alarm reduction in signature-based IDS: game theory approach. Secur. Commun. Netw. **9**(18), 4863–4881 (2016)
16. Tug, S., Meng, W., Wang, Y.: CBSigIDS: towards collaborative blockchained signature-based intrusion detection. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1228–1235 (2018)
17. Vasilomanolakis, E., Karuppayah, S., Mühlhäuser, M., Fischer, M.: Taxonomy and survey of collaborative intrusion detection. ACM Comput. Surv. **47**(4), 1–33 (2015)
18. Zhou, C.V., Leckie, C., Karunasekera, S.: A survey of coordinated attacks and collaborative intrusion detection. Comput. Secur. **29**(1), 124–140 (2010)
19. Zhuang, Q., Liu, Y., Chen, L., Ai, Z.: Proof of reputation: a reputation-based consensus protocol for blockchain based systems. In: Proceedings of the 2019 International Electronics Communication Conference, pp. 131–138 (2019)