Blockchain-based Bug Bounty Framework

Lital Badash
Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev
litalbe@post.bgu.ac.il

Nachiket Tapas
Department of Engineering, University of Messina
Messina, Italy
ntapas@unime.it

Asaf Nadler
Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev
asafnadl@post.bgu.ac.il

Francesco Longo
Department of Engineering, University of Messina
Messina, Italy
flongo@unime.it

Asaf Shabtai
Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev
shabtaia@bgu.ac.il

ABSTRACT

Bug bounty programs are a popular solution for security researchers to disclose software vulnerabilities in exchange for compensation. They suffer, however, from two main drawbacks that limit their effectiveness: *(i)* they use a trusted intermediary that charges hefty commission fees and may have a conflict of interest with the software vendor, and *(ii)* they may mistreat security researchers by compensating less than guaranteed and no means to appeal against it. In this paper, we propose a permissioned Blockchain-based framework that addresses the drawbacks of existing bug bounty programs. The framework allows a confidential exchange of vulnerabilities and compensations using smart contracts. In cases of policy violation, security researchers can appeal to a trusted group of security experts called arbitrators, that can force the software vendors to compensate the security researchers fairly. A formal evaluation of the proposed framework using TLA+ specification supports the viability of the proposal. A Hyperledger Fabric-based prototype is implemented to simulate the proposed framework. The analysis of the framework uses a game-theoretic notion to argue that if the majority of arbitrators behave honestly, then the rational strategy of software vendors is to compensate security researchers that disclose vulnerabilities accurately. Similarly, rational security researchers do not gain any financial profit by playing unfairly.

CCS CONCEPTS

• **Security and privacy** → *Domain-specific security and privacy architectures*; • **Information systems** → *Collaborative and social computing systems and tools*;

KEYWORDS

Blockchain, Bug Bounty, Software vulnerability

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3441906

ACM Reference Format:
Lital Badash, Nachiket Tapas, Asaf Nadler, Francesco Longo, and Asaf Shabtai. 2021. Blockchain-based Bug Bounty Framework. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3412841.3441906

1 INTRODUCTION

Software vulnerabilities are bugs that can be exploited by hackers to compromise systems and consequently harm organizations' reputation. Successful exploitation of software vulnerabilities may result in corruption of information, disclosure of information, denial of service, and theft of service [7]. This drives software companies to actively search for existing vulnerabilities and fix them before they are exploited. Searching for vulnerabilities is an exhaustive process that requires skills, creativity, and time. Thus, most software developers cannot search for vulnerabilities on their own and rely on third-party experts through bug-bounty programs [9].

Bug-bounty programs (or vulnerability reward program) are an offer made by a software company directly or indirectly through a third-party, to encourage independent security researchers, penetration testers, and white hat hackers [15] to search for software vulnerabilities and disclose them for compensation and recognition. Previous studies [4, 13] demonstrated that security researchers will get higher rewards on average for selling their disclosed vulnerability on black markets with malicious intent [13] compared to bug bounty programs, and are therefore less incentivized to use bug bounty programs. To tackle this problem, modern bug bounty solutions seek better social and economic models for minimizing the gap between legitimate and malicious vulnerability markets [13].

The most popular bug bounty program solutions are managed service platforms such as HackerOne Inc. [1] and BugCrowd Inc.[1] that maintain a community of security researchers and have a reputation for operating successful indirect bug bounty programs. These platforms are growing rapidly through more bounties program and more valuable rewards [12] reaching a 19 million dollars of rewards in 2018, a figure which is nearly the sum of total reward of all preceding years combined [10]. However, software vendors and researchers may hesitate in participating in bug bounties programs through these platforms due to the following concerns: *(i) Costs.*

[1] https://www.bugcrowd.com/

The reward for disclosing vulnerabilities on bug bounty platforms encompasses a non-negligible fee of the platform. For example, HackerOne Inc. charges up to 20% higher fees than direct bug bounties to support the platform operation [1]. *(ii) Transparency.* The lack of transparency in the process of evaluating whether a disclosed vulnerability is rewardable may in turn lead to a conflict-of-interest between software companies, that are the customers of the bug bounty service, and the security researcher community maintained by the bug bounty platform. *(iii) Security.* The direct bug bounty program uses a secure communication channel between the software vendor and the security researcher for exchanging vulnerability information. Indirect bug bounty programs force the inclusion of a third party, thus increasing the risk of sensitive information leakage [22]. *(iv) Fairness.* Current bug bounty platforms do not guarantee fairness, and security researchers face the risk of non-payment [4]. With these platforms often suggesting a reputation mechanism for security-researchers, lack of fairness forces the security researcher to disclose security vulnerabilities to direct sellers they can trust.

We propose a decentralized bug bounty program framework that uses a permissioned blockchain technology to better address the concerns of software companies and security researchers (cost, transparency, security, and fairness) that seek a bug bounty program. A software company, interested in initiating a bug bounty program, posts a smart contract on the blockchain with the program details, including the scope, terms and conditions, submission process, and the proposed reward. The smart contract for a bug bounty program needs to state and publish the program's policy/business rules followed by any bug bounty platform like HackerOne or BugCrowd, and therefore, with precise and unambiguously defined rules [17], defining a smart contract is easy as its counterpart. A security researcher that discovers a vulnerability can disclose it to the software company via the smart contract.

The primary guarantees of the framework are transparency and fair exchange i.e., either a software company verifies the disclosed vulnerabilities and behaves honestly by rewarding the security researcher, or alternatively, the security researcher decides to initiate an arbitration. In the latter case, a committee of trusted arbitrators is set in place to decide disputed cases, and the party found guilty of misbehavior is forced to compensate the other party for a value that exceeds that of the disclosed vulnerability, thus guaranteeing fairness. The permissioned blockchain complements the process of trusted arbitration by providing transparency, immutability, and non-repudiation properties alongside smart contracts execution to bind payment in cases of dispute execution thus filling the gap and ensure trust and fairness. Existing fair exchange proposals involve either a centralized, trusted third party [20] or gradually releasing the commodity in parts [5]. Both approaches have their limitations of trust and the need for secret's verifiability in polynomial time, which is achieved in the proposed framework via the trusted committee as verifiers to any free text vulnerability description, which cannot be performed in polynomial time The fairness of vulnerability disclosure using the suggested framework is analyzed by modeling the process as a game, and demonstrating a Nash equilibrium as proposed by Buttyan *et al.* [5].

To summarize, the main contributions of this research are: *(i)* We propose a decentralized bug bounty framework that uses a permissioned blockchain to guarantee security, transparency, fairness, and reasonable cost. *(ii)* The bug bounty platform uses an arbitration committee to decide disputed cases to ensure fairness and transparency. *(iii)* A formal specification of the proposed framework is presented to evaluate the correctness of the framework. *(iv)* A Hyperledger Fabric-based prototype is implemented to validate the operation of the proposed framework. *(v)* A game-theoretic analysis of the bug bounty platform is presented to prove that vendors and security researchers gain the most profit from this framework if they are rational and honest.

2 BACKGROUND

Bug bounty programs. Bug bounty programs contain a set of details, including the program's scope, the reward for disclosing bugs, and legal constraints. The details must be made publicly available to encourage any security researcher to search for bugs and disclose them in exchange for a reward. The average length of bug bounty programs' details is 481 words [17], which is sufficiently small to fit in blockchain smart contracts.

The process of vulnerability disclosure starts by submitting a private message to the bug bounty platform and waiting for a commonly agreed grace period in which the software company has the chance to verify, evaluate and fix the vulnerability [6]. After the defined grace period, a disclosed vulnerability is referred to as public knowledge. It can be shared publicly to educate other security researchers, demonstrate the software's improved security status, and allow security researchers to file a dispute, if needed, without risking legal liability for publicly discussing the vulnerability.

Blockchain and smart contracts. A blockchain is a distributed public ledger of transactions or digital events among participating parties [8]. Despite being introduced as digital currency, the blockchain is widely adopted in non-financial use cases as an immutable and transparent, general-purpose public ledger of digital events. One of the blockchain's primary extensions is "smart contracts", which are computer programs that can automatically execute the terms of a contract [8]. A smart contract can be programmed to enforce the desired set of rules encoded in its programming language. Smart contracts are identified by an address (a 160-bit identifier) to which users send transactions to execute the contract code [19], and their code is immutable similarly to every digital event on the blockchain.

3 RELATED WORK

Bug bounty programs. The most popular form of bug bounty programs is software-as-a-service (SaaS) programs [21], such as HackerOne and BugCrowd. Companies that offer SaaS bug bounty programs have experience maintaining a community of security researchers, drafting bounty contracts, negotiating terms, and guaranteeing a secure disclosure. Thus, software companies lacking competency can use SaaS bug bounty programs to encourage bug disclosures. The main downside of SaaS bug bounty programs, in contrast to the proposed solution, is the fees required to cover the legal and operational costs.

An additional bug bounty alternative is a self-managed bug bounty that is published and managed directly by software companies. In contrast to SaaS bug bounty programs, a self-managed program saves the cost of third-party services. However, it requires the software company to be actively involved with the security community and possess legal expertise in vulnerability disclosures.

The proposed framework attempts to provide the advantage of software-as-a-service and self-managed bug bounty programs by providing a framework with an active community of security researchers and software companies and has clear and binding legal policies drafted through smart contracts. The fees for the proposed framework are expected to be significantly less than those of SaaS bug bounties while also encouraging security researchers to search for bugs.

Bug bounties using blockchain. The Hydra project [4] proposes a complete bug bounty specifically for immutable smart contracts using a technique referred to as N-of-N version programming (NNVP). NNVP requires that the smart contract would be implemented in N independent versions to identify and automatically remediate bug disclosures. While this is reasonable for immutable smart contracts that consist of a small number of code lines and that offer large amounts of tokens, it is not cost-effective for a generalized bug-bounty program. The work of [23] uses smart contracts for bug bounties (using Software Guard Extensions (SGX)). Their system is focused on the peer-to-peer fare exchange by zero knowledge and sealed glass proofs, in contrast to our suggestion that relies on feedback and arbitration process by the community. Moreover, their suggestion targets only bug bounties for applications that can be ported to SGX.

Blockchain and smart contracts solutions for enhanced security. In a closely related work, researchers propose a system named Desma [14], in which trusted intermediaries operating a marketplace can be replaced with a set of rules encoded in smart contracts. However, in proposed work, dispute resolution can be settled automatically by code. Researchers in [11] suggest a reputation based knowledge sharing system in blockchain for knowledge owner's copyright protection and paid for content service. The project Polyswarm [2] suggest a system to analyze artifacts by researchers. Researchers bid the right to 'answer' if an artifact is malicious or benign by using an automated tool. In the proposed framework, we use the smart contract to publish immutable, time-limited, bug bounty programs, accumulate publicly visible feedback, and manage arbitration in case of a dispute between the parties, in a traceable but confidential manner, while respecting the rules of responsible disclosure [6].

We use the blockchain-based network to publish bug bounty programs as smart contracts and allow security researchers to participate. Simultaneously, our framework's protocols follow a game-theoretic approach to address the fair-exchange issue by accumulative feedback and a unique arbitration process that increase the economically rational researcher's incentives to participate.

4 PROPOSED FRAMEWORK

We propose a framework to support bug bounty programs where software vendors and security researchers can exchange vulnerability information for compensation and recognition while guaranteeing fairness. We refer to the proposed work as a framework rather than a concrete system design because some of the parts are provided as a general guideline and can be implemented in several ways and with different technologies.

4.1 Framework components

The proposed framework consists of the following components.

A permissioned blockchain network, in which vendors commit to rewarding security researchers in exchange for their disclosure of a vulnerability. This is managed using smart contracts. Blockchain network nodes are host machines owned by bug bounty program holders or other stakeholders. As part of the system initialization, each user willing to participate in the system has to register with a trusted registration authority. The initialization prevents Sybil attacks and feedback cheating behaviors. Users won't be able to create multiple identities to gain feedback and thus win more rewards. They have a coin account, and associated accumulative feedback is stored in a smart contract.

Smart contract. Bug bounty program can be phrased in a smart contract. Vendors use smart contracts to publish their bug bounties programs and to reward researchers for their findings. The arbitration process is managed in a different smart contract. The framework handles the arbitration process, which occurs when the researcher and vendor disagree regarding a bug severity that influences the bug's cost. This way, the arbitration process is correctly executed. Another type of contract is for the feedback storage. Feedback is given automatically by the smart contract managing the arbitration process and manually by the participating users and saved corresponding to the user's identity.

Committee of arbitrators. A trusted committee with N (odd) members, who act as arbitrators in case of a dispute between a researcher and a vendor regarding the severity of the bug that determines the reward. At least half of the arbitrators are assumed to be honest, i.e., their verification is considered as the ground truth. The realization of the trusted committee can be a set of trusted machines that are owned by highly professional security researchers that are part of the community and are incentivized to perform this task fairly. For instance: representatives from participated companies and highly reliable users with proven experience.

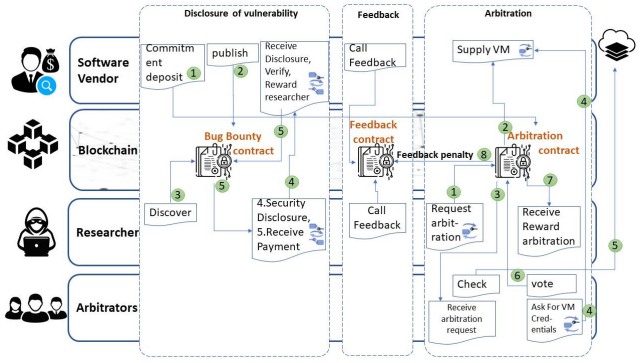

Figure 1: Disclosure of vulnerability, feedback, and arbitration processes.

4.2 Actors

The framework supports three types of actors, namely: vendors, security researchers, and arbitrators. *Software vendor (shortly referred to as vendor).* The goal of vendors in the system is to provide compensation for the disclosure vulnerabilities found in their system. The vendors achieve this goal by publishing a bug bounty program as a smart contract on the blockchain network and committing to adequately compensate for the vulnerability disclosure. The proposed contract defines the program's scope and the rules alongside incentives to encourage security researchers to find vulnerabilities. *Security researcher (shortly referred to as researcher).* The goal of researchers is to disclose vulnerabilities that they find in return for compensation. A researcher that finds a vulnerability sends it within an encrypted transaction to the vendor to assure confidentiality. In turn, the vendor verifies the vulnerability and rewards the researcher as guaranteed in the corresponding bug bounty smart contract. *Arbitrator.* Researchers or professionals can be part of the committee of arbitrators. Their role is to resolve the conflict between a vendor and a researcher in the event of a disagreement regarding a finding's severity that determines the reward.

4.3 The framework flow

In our framework, there are three processes: (i) disclosure of vulnerability – proposing bug bounty programs by vendors to the community, (ii) feedback – allowing researchers and vendors to leave their impressions on each other in a specific format, and (3) arbitration process – trusted committee that can make a verdict in case of a dispute. Feedback and the arbitration process are not mandatory. Disclosure of vulnerability allows vendors to propose bug bounty programs in the form of smart contracts in which the vendor commits to provide coins in exchange for a vulnerability disclosure satisfying the program rules and scope. Researchers who want to demonstrate their skills and win the reward join the program, test the vendor's site/artifact/system, and, if a security-related issue is found, encrypt the data and transact with the vendor directly. Once the vendor receives messages from the researchers, he verifies the data. He decides if and how a researcher should be rewarded according to the bug severity estimation model stated in the program. Feedback is to give performance reviews according to agree upon topics to the researchers and the vendors. The feedback is symmetric, should be honest, reflect the researcher's set of knowledge and skills, and the vendor's professional and fair behavior. The arbitration process occurs when a researcher feels the vendor is trying to withdraw his commitment from the smart contract and not pay fairly. A special smart contract is automatically managing the arbitration process, collecting the arbitrators' votes, and transferring payment to the party who got the majority of votes. The architecture of the framework is depicted in Figure 1.

4.4 Disclosure of vulnerability

We denote pk_V, pk_R, pk_A as the public keys of vendor V, researcher R, and arbitrator A respectively. The bug bounty index is pid, determined by the id of the transaction that deployed the smart contract, and the id is unique. The process for disclosing a vulnerability process is the main process of the framework, and it includes both software vendors and security researchers. This process includes multiple consecutive steps: **(i) Commitment:** A vendor v_i willing to participate in the system, is required to make a deposit to the arbitration contract before offering a program. **(ii) Publish:** A vendor v_i releases a new bug bounty program P, by deploying a smart contract to the blockchain network.

$$DeployBugBounty(pk_{vi}, t_{vi}, d_{vi}, m_{vi}, c_{vi})$$

where pk_{vi} denotes the vendor who own and manage the program, t_{vi} denotes the program's expiration date, d_{vi} denotes the program details, m_{vi} denotes the amount of coins deposited to the smart contract for the reward fee, and c_{vi} denotes the program code which essentially says: transfer coins according to the bug impact estimation model to the researcher upon vendor's validation. **(iii) Discovery:** Researchers (denoted R) are watching the blockchain for an indication that a relevant smart contract is deployed by a vendor known by his public key, pk_{vi}. If the researcher is interested, he or she can query for the program details by

$$QueryProgramDetails(pk_{vi})$$

(iv) Disclosure: The Researcher, finds a bug, describe its details, generates a key, named K_{pid}, the key is encrypted by the public key of the involved vendor, and sends the transaction to the vendor. The vulInfo represents the vulnerability information.

$$Enc_{pkvi}(K_{pid})$$

$$SecIssueDisc(Enc_{pk_{vi}}(K_{pid}), Enc_{K_{pid}}(vulInfo))$$

(v) Evaluation: The vendor verifies the security issue sent to him by a researcher on the time frame to decrypt the message, trying the steps to reproduce as suggested by the researcher, estimate bug reward and send a reply to the bug bounty program smart contract where s denotes the bug impact estimation value that set the reward fee. In case of rejection, s is not applicable and the researcher is not getting any reward.

$$Verify(Enc_{pk_{vi}}(vulInfo))$$

$$RewardResearcher(pk_{ri}, s)$$

4.5 Feedback

In our suggested framework, we use feedback messages as a key component that represents the user's hall of fame. Users, both vendors, and researchers accumulate feedback. Bug bounty programs from vendors with positive feedback messages have better chances to get picked by high-rated and relevant researchers. Researchers with positive feedback indicate their knowledge and expertise and have a better chance to be recognized for their professionalism in the community. They have higher probabilities of being noticed by vendors and motivate them to find security flaws and earn more rewards. Feedback is given in a pre-defined format and categories and not as a free text to encourage only positive feedback. Dishonest behavior is noticed and marked by the system. This paper provides the format and a few examples of the category. The framework allows two types of feedback: *User feedback.* Users of the system can leave their impressions on each other using feedback by calling the method managed by the feedback storage smart contract.

$$CallFeedback(user_{pk}, p_{id}, feedback)$$

$user_{pk}$ denotes the public key of the user who receives the feedback, p_{id} denoted the bug bounty index. Vendors fill the feedback values by selecting from a pre-defined list of expertise that they find relevant to the issue raised by the researcher. For example, SQL injection, cross-site scripting (XSS), etc. Researchers fill the feedback values by selecting from a pre-defined list of categories important to the disclosure of the vulnerability process and assign their satisfaction score. For example (responding time, score), (committing to the reward promised, score). *Penalty feedback.* The losing party in

the arbitration process is penalized by penalty feedback. Multiple losses for the vendors or researchers can dramatically affect his attractiveness. This kind of feedback is executed automatically by the arbitration process smart contract using the following method.

$$CallPenaltyFeedback(user_{pk}, p_{id})$$

Feedback is stored and managed by the feedback storage smart contract, which allows anyone to read the system's users' feedback and retrieve relevant researchers by their public keys. Since a researcher can receive more than one feedback on the same expertise, the feedback storage maintains a counter per expertise per pk_{ri}. The vendor's total rating score on each category is the average of all scores on the same category. Penalty feedback is accumulated as well.

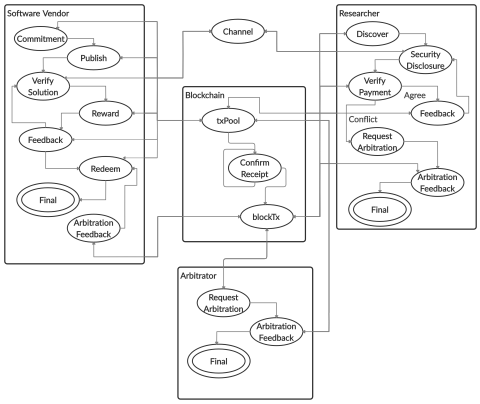

Figure 2: Bug bounty framework state transition diagram.

4.6 Arbitration process

The arbitration process should provide trusted verification. That is the basis of the framework's fair exchange. This process is an optional step, initiated by a researcher and using the community to advise in case of a disagreement between a vendor v_i and a researcher r_i regarding the bug's severity. For this purpose, the arbitration process's contract needs to maintain state of the vendor-to-deposit mapping, allow vendors to deposit coins, and enable researchers to find committed vendors. *Initialization:* In the commitment phase of the disclosure of vulnerability process, a vendor is required to make a deposit to the arbitration contract. The amount of coins the vendor submit is the reflective of his guarantee to the researchers. The arbitration contract is essentially guaranteeing that the money deposited by a vendor is either: (A) redeemed by a majority vote of the arbitrators, or (B) released back to the vendor if a time lock he set during deposit expires. *Request Arbitration:* Researcher r_i uses the same key he used during submission, K_{pid}, the key is encrypted by the public key of the involved vendor, $Enc_{pk_{vi}}(K_{pid})$ and send a message to the arbitration process smart contract.

$$AskForArbitration(knowledgepreferences, pid, Enc_{K_{pid}}(vulInfo), Enc_{pk_{vi}}(K_{pid}), (severity\ estimation\ values))$$

Severity estimation values are the two values the arbitrators should vote from. One is the value assigned by the vendor and the other is the value the researcher believe is more suitable. *Arbitration execution:* Upon request for arbitration, the contract executes the following:

- Request the vendor to supply virtual machines and inform arbitrators respectively by:

$$ArbitrationPrepare(pk_{ri}, Enc_{K_{pid}}(vulInfo), pid, numberOf-Arbitrators, Enc_{pk_{vi}}(K_{pid}))$$
$$InformArbitrator(pid, submission\ time, (severity\ estimation\ values), Enc_{K_{pid}}(vulInfo))$$

- Accumulate the arbitrator's votes and set the winning party to the one that got the majority of votes,
- Transfer the money to the wining party, and
- fill feedback penalty on the losing party

The vendor is technically losing in the arbitration if he/she refuses to participate in the process by not supplying the testing environment when needed. This case is marked as special penalty feedback due to its importance.

Upon request to verdict the arbitrator first accept the arbitration and only after that gets access to the environment (vminfo).

$$AcceptArbitration(pid, Enc_{K_{pid}}(vulInfo))$$
$$AccessArbitrationEnv(Enc_{pk_{ai}}(K_{pid}), Enc_{K_{pid}}(vminfo)), Enc_{K_{pid}}(vulInfo))$$

Supplying virtual machine details are necessary only when it is listed as a prerequisite in the program's contract. K_{pid} is used as a symmetric key encryption. Arbitrator extract K_{pid} and decrypt the vminfo, access the VM with the given credentials, and examine the case. Note, the arbitrator receives the vulInfo both from the vendor and the researcher to avoid fraud. Finally the arbitrator send his vote to the arbitration contract by:

$$SendArbitrationDecision(a_i, p_{id}, (impact\ estimation\ value))$$

5 FORMAL SPECIFICATION

In this section, we present a formal specification of the bug bounty framework using TLA+ abstract language [16, 18] for model verification. The system operation is defined as TLA+ definitions, and the correctness of the protocol is verified using the TLC model checking tool. The resulting formal specification is then used as input for the implementation of the framework.

5.1 State Transitions

The framework involves interaction between software vendors (V) and researchers (R) for discovering software vulnerabilities (Figure 2). In case there is a conflict, the arbitrators (A) mediate the conflict and announce a winner. Blockchain acts as a ledger of records containing performed operations and is also used as a communication and payment medium. We begin by initializing each variable with a default value. The variables InitVendor, InitResearcher, InitArbitrator, and InitBlockchain are initial values of the variables for the software vendor, researcher, arbitrator, and blockchain, respectively. The channel variable is used to represent a secure communication between the researcher and vendor for vulnerability disclosure. The state change for the proposed model is defined as $Init \wedge *[\ Next]_{\ vars} \wedge WF\_vars(Next)$ in TLA+ specifications where the WF_vars is for fairness. The possible transitions from the initial state are described in the following equation:

$$\begin{aligned}
&Next \triangleq \\
&\vee \exists v \in Vendor: \\
&\quad \vee Commitment(v) \vee Publish(v) \quad \vee VerifySolution(v) \vee Reward(v) \\
&\quad \vee Redeem(v) \\
&\vee \exists r \in Researcher:
\end{aligned}$$

```
  ∨ Discover(r) ∨ Disclosure(r) ∨ VerifyPayment(r)
∨ ∃ a ∈ Arbitrator :
  ∨ AcceptArbitration ∨ Vote(a) ∨ ArbitrationResult
∨ ConfirmReceipt
∨ (* Disjunct to prevent deadlock on termination *)
  ((∀ ven ∈ Vendor : vState[ven] = V_Final) ∧ Terminating)

Spec ≜
Init ∧ *[Next]_vars ∧ WF_vars(Next)
```

The system can transit to any of the states defined by *Next* which includes the terminating condition. A private channel is created to communicate the vulnerability information securely. Finally, termination is indicated by the expiration of the published bug bounty program, and the vendor can redeem the initial commitment. The researchers may or may not disclose a vulnerability.

5.2 Software vendors

The software vendor is interested in identifying the vulnerability in his/her software. The vendor begins with a *V_Waiting* and then transits through *V_Commitment*, *V_VerifySolution*, *V_Reward*, *V_Feedback*, and *V_Final*. In the end, the vendor is either able to identify the vulnerabilities and can work on fixing them or the time expires, and the vendor is assured that the software is bug-free. *(i) Commitment:* The software vendor begins by committing a pre-defined amount to the blockchain to act as a guarantee in case arbitration is required.

```
Commitment(v) ≜
 ∧ vState[v] = V_Waiting
 ∧ LET
    type == TX_COMMITMENT  pgrmnum == NUMBER
    commitment == COMMITMENT
   IN
    Transaction( type, pgrmnum, v, ⟨ commitment ⟩ )
    ∧ vState' = [ vState EXCEPT ![v] = V_Commitment ]
    ∧ UNCHANGED ⟨ researcherVars, arbitratorVars, blockTx,
      balance, channel ⟩
```

The transaction is identified by *TX_COMMITMENT*. Although the commitment phase is optional in our model, the formal specification begins with the commitment phase as it ensures trust. A researcher may decide not to participate in the bug bounty program in the absence of commitment. *(ii) Publish:* The vendor, after making a commitment on the blockchain, transits to *V_Commitment* state. The vendor publishes the bug bounty program on the blockchain.

```
Publish(v) ≜
 ∧ vState[v] = V_Commitment
 ∧ LET
    type == TX_PUBLISH  pgrmnum == NUMBER
    policy == POLICY  issue == DATE_ISSUE
    expiration == DATE_EXP  bounty == BOUNTY
   IN
    Transaction( type, pgrmnum, v, ⟨ policy, issue, expiration, bounty ⟩ )
    ∧ vState' = [vState EXCEPT ![v] = V_VerifySolution]
    ∧ UNCHANGED ⟨ researcherVars, arbitratorVars, blockTx,
      balance, channel ⟩
```

The publish transaction contains the details related to the bug bounty program like the policy of the program, issue date, expiry date, and the bounty to be awarded to the researcher. The transaction is identified by TX_PUBLISH. Once the transaction is sent to the blockchain, the vendor transits to *V_VerifySolution* state, waiting for any researcher to submit the vulnerability. *(iii) Verify solution:* The vendor in *V_VerifySolution* state observes the secure channel for the vulnerability report. Though the exchange of vulnerability report is done privately, the proof of exchange is published on the blockchain to prevent repudiation by the vendor regarding receiving vulnerability reports.

```
VerifySolution(v) ≜
 ∃ c ∈ channel :
 LET
  enckey == c.key  vulinfo == c.info
 IN
  IF Verify ( enckey, vulinfo ) THEN
   ∧ vState' = [vState EXCEPT ![v] = V_Reward]
   ∧ UNCHANGED ⟨ researcherVars, arbitratorVars, blockchainVars,
     channel ⟩
 ELSE
  UNCHANGED ⟨ vendorVars, researcherVars, arbitratorVars,
  blockchainVars, channel ⟩
```

On successful verification of the vulnerability report, the vendor transits to *V_Reward* state. For the sake of simplicity, we have abstracted the verification of the report. If the report is not valid, the vendor remains in the same state, waiting for a valid report. *(iv) Reward:* The vendor, after validating the vulnerability report, rewards the researcher with the bounty as per the severity of the incidence. This may lead to contention between the vendor and the researcher and is dealt with later.

```
Reward(v) ≜
 ∧ vState[v] = V_Reward
 ∧ LET
    type == TX_REWARD  pgrmnum == NUMBER
    researcher == RESEARCHER  issue == DATE_ISSUE
    reward == REWARD
   IN
    Transaction( type, pgrmnum, v, ⟨ researcher, issue, reward ⟩ )
    ∧ vState' = [vState EXCEPT ![v] = V_Feedback]
    ∧ UNCHANGED ⟨ researcherVars, arbitratorVars, blockTx,
      balance, channel ⟩
```

The vendor sends the transaction identified by *TX_REWARD* and containing the researcher who identified the vulnerability, issue date, and the reward. Once done, the vendor moves to the feedback state to provide feedback to the researcher. *(v) Vendor feedback:* On successful acceptance of reward by the researcher, the vendor provides the feedback for the researcher as a transaction on the blockchain. The vendor sends the transaction identified by *TX_FEEDBACK* and containing the researcher who identified the vulnerability, and the feedback. The overall feedback can be identified by accumulating all such transactions. *(vi) Redeem:* This is the final state of the vendor where the vendor recovers the initial commitment. In case there was no arbitration requested by any researcher, the vendor sends an equal and opposite transaction to the commitment transaction. In the case of arbitration, the vendor recovers the remaining amount.

```
Redeem(v) ≜
 ∧ vState[v] = V_Feedback
 ∧ LET
    type == TX_COMMITMENT  pgrmnum == NUMBER
    commitment == COMMITMENT * (−1)
   IN
    Transaction( type, pgrmnum, v, ⟨ commitment ⟩ )
    ∧ vState' = [vState EXCEPT ![v] = V_Final]
    ∧ UNCHANGED ⟨ researcherVars, arbitratorVars, blockTx, balance,
      channel ⟩
```

The vendor sends the transaction identified by *TX_COMMITMENT* and containing the relevant amount to recover. This transaction also marks the completion of a bug bounty program.

5.3 Researchers

The researcher begins with a *R_Waiting* and then transits through *R_Disclosure*, *R_WaitingVerification*, and *R_Feedback*. Optionally, a researcher may transit to *R_Arbitration*, and *R_WaitingArbitration*. In the end, the researchers receive rewards for identifying the vulnerabilities with the software.

Discover: The researcher observes the blockchain looking for the bug bounty programs initiated by the vendors. Based on the vendor commitment, the researcher may decide to participate in the program. Once the researcher decides to take part in the program, the researcher transits to state *R_Disclosure*.

$$
\begin{aligned}
&Discover(r) \triangleq \\
&\quad \exists trans \in blockTx : \\
&\qquad \wedge rState[r] = R\_Waiting \wedge trans.type = TX\_PUBLISH \\
&\qquad \wedge trans.number = NUMBER \wedge trans.issuer = VENDOR \\
&\qquad \wedge rState' = [rState\ EXCEPT\ ![r] = R\_Disclosure] \\
&\qquad \wedge UNCHANGED \langle vendorVars, arbitratorVars, blockchainVars, \\
&\qquad\quad channel \rangle
\end{aligned}
$$

Disclosure: In this state, the researcher identifies the vulnerability in the software, prepares a vulnerability report, and sends the report privately to the vendor. The researcher also records it on the blockchain as a proof of participation.

$$
\begin{aligned}
&Disclosure(r) \triangleq \\
&\quad \wedge rState[r] = R\_Disclosure \\
&\qquad \wedge LET \\
&\qquad\quad enckey == ENC\_KEY \;\; vulinfo == VULNERABILITY \\
&\qquad IN \\
&\qquad\quad SendDisclosure(\ [key\ |->\ enckey, info\ |->\ vulinfo]\ ) \\
&\qquad\quad \wedge rState' = [rState\ EXCEPT\ ![r] = R\_WaitingVerification] \\
&\qquad\quad \wedge UNCHANGED \langle vendorVars, arbitratorVars, blockchainVars \rangle
\end{aligned}
$$

The function SendDisclosure abstracts the secure sending of the vulnerability report. This can be handled by existing web technologies like SSL [2] or more advanced features of blockchain like private data of Hyperledger Fabric.[3] Also, together with the report, the researcher sends the severity of the incidence. Once the researcher sends the report, he/she transits to *R_WaitingVerification* state, waiting for the verification of the solution and receiving the reward.

Verify payment: On receiving the reward transaction from the vendor, the researcher verifies the reward received on the blockchain. There is a possibility of a dispute between the incident severity estimated by the researcher and the vendor. In case there is no dispute, the researcher transits to *R_Feedback* state. In case of a dispute, the researcher requests for mediation from the arbitrators.

$$
\begin{aligned}
&VerifyPayment(r) \triangleq \\
&\quad \exists trans \in blockTx : \\
&\qquad \wedge rState[r] = R\_WaitingVerification \wedge trans.issuer = VENDOR \\
&\qquad \wedge trans.type = TX\_REWARD \wedge trans.number = NUMBER \\
&\qquad \wedge IF\ VerifyReward(\ trans.data[3]\ )\ THEN \\
&\qquad\quad \wedge rState' = [rState\ EXCEPT\ ![r] = R\_Feedback] \\
&\qquad\quad \wedge UNCHANGED \langle vendorVars, arbitratorVars, blockchainVars, \\
&\qquad\qquad channel \rangle \\
&\qquad ELSE \\
&\qquad\quad \wedge LET \\
&\qquad\qquad type == TX\_ARBITRATION \;\; pgrmnum == NUMBER \\
&\qquad\qquad proposal == PROPOSAL \;\; option == OPTION \\
&\qquad\qquad issue == DATE\_ISSUE \;\; expiration == DATE\_EXP \\
&\qquad\quad IN \\
&\qquad\qquad Transaction(\ type, pgrmnum, r, \langle\ proposal, option, issue, \\
&\qquad\qquad\quad expiration\ \rangle\ ) \\
&\qquad\qquad \wedge rState' = [rState\ EXCEPT\ ![r] = R\_WaitingArbitration] \\
&\qquad\qquad \wedge UNCHANGED \langle vendorVars, arbitratorVars, blockTx, balance, \\
&\qquad\qquad\quad channel \rangle
\end{aligned}
$$

The arbitration transaction is identified by *TX_ARBITRATION* and contains details like the details related to the bug bounty program and the identified vulnerability, the researcher and the vendor incident severity, issue, and expiry. Once the transaction is sent to the blockchain, the researcher transits to *R_WaitingArbitration* and waits for the decision of the arbitrators.

Researcher feedback: The researcher in the feedback state provides feedback for the vendor based on his/her interaction. The feedback is in the form of a transaction on the blockchain containing the vendor identification and corresponding feedback. Once the researcher provides the feedback, the researcher may try to find more vulnerabilities until the program expires.

5.4 Arbitrators

The arbitrators are responsible for resolving the dispute between the vendor and the researcher. The consortium of arbitrators vote on a particular dispute and declare the result based on majority voting. Also, at the end of each arbitration, the arbitrator provides negative feedback to the losing entity. The arbitrators begin A_Waiting and then transit to *A_Vote*.

Accept arbitration: The arbitrators accept the arbitration and transit to the voting state.

$$
\begin{aligned}
&AcceptArbitration \triangleq \\
&\quad \exists trans \in blockTx : \\
&\qquad \wedge trans.type = TX\_ARBITRATION \wedge trans.number = NUMBER \\
&\qquad \wedge trans.issuer = RESEARCHER \\
&\qquad \wedge aState' = [x \in Arbitrator\ |->\ A\_Vote] \\
&\qquad \wedge UNCHANGED \langle numVoted, vendorVars, researcherVars, \\
&\qquad\quad blockchainVars, channel \rangle
\end{aligned}
$$

Vote: The arbitrators begin voting on the proposal proposed by the arbitration transaction, choosing between the options included in the transaction. The voting status of the arbitrators is tracked to identify the end of the voting process.

$$
\begin{aligned}
&Vote(a) \triangleq \\
&\quad \wedge aState[a] = A\_Vote \wedge numVoted' = numVoted + 1 \\
&\quad \wedge aState' = [aState\ EXCEPT\ ![a] = A\_Result] \\
&\quad \wedge UNCHANGED \langle vendorVars, researcherVars, blockchainVars, \\
&\qquad channel \rangle
\end{aligned}
$$

Arbitration result: Once all the arbitrators have voted, the result of the arbitration is sent to the blockchain identified by TX_ARBITRATIONRESULT. Also, corresponding feedback is sent to the blockchain.

$$
\begin{aligned}
&ArbitrationResult \triangleq \\
&\quad \wedge IF\ numVoted = Cardinality(Arbitrator)\ THEN \\
&\qquad \wedge LET \\
&\qquad\quad type == TX\_ARBITRATIONRESULT \;\; pgrmnum == NUMBER \\
&\qquad\quad issuer == RESEARCHER \;\; result == RESULT \;\; winner == WINNER \\
&\qquad\quad looser == LOOSER \;\; feedback == FEEDBACK \\
&\qquad IN \\
&\qquad\quad Transaction(\ type, pgrmnum, issuer, \langle\ result\ \rangle\ ) \\
&\qquad\quad Transaction(\ type, pgrmnum, winner, \langle\ feedback\ \rangle\ ) \\
&\qquad\quad Transaction(\ type, pgrmnum, looser, \langle\ feedback * -1\ \rangle\ ) \\
&\qquad\quad \wedge UNCHANGED \langle vendorVars, researcherVars, arbitratorVars,
\end{aligned}
$$

[2] https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-5906-5_223
[3] https://hyperledger-fabric.readthedocs.io/en/release-2.0/private-data/private-data.html

$blockTx, balance, channel\rangle$
ELSE
$\wedge UNCHANGED \langle vendorVars, researcherVars, arbitratorVars, blockchainVars, channel\rangle$

5.5 Model checking

The TLC model evaluation involves parsing the state space to find critical states, e.g., states leading to deadlock conditions or violation of invariants. We performed simulation on TLA+ model verification tool to check the correctness of the proposed system and the critical conditions. We ran the experiment on a Windows 10-based Intel i7 quad-core machine with 32 GB RAM. The evaluation is performed for: *(i)* the normal situation with no arbitration, and *(ii)* the situation involving arbitration. The results are presented in Table 1.

The normal situation does not involve any contention between the researchers and vendors. The researcher accepts the decision taken by the vendor. For the normal situation, the simulation process took 11s and visited 31,161 distinct states. In arbitration, where there is a conflict between researchers and vendors, due to the static nature of the simulation, the simulation process did not terminate. However, the simulation did not find a deadlock for the duration of 3600s. Here we present the statistics for the first 10 minutes. Based on the results presented in the Table 1, we can conclude that the proposed scheme is error-free.

6 IMPLEMENTATION AND SIMULATION

The proposed prototype is based on a permissioned - enterprise blockchain called Hyperledger fabric [3], release-1.4 Inc.[4] The architecture consists of three organizations, Vendors, Researchers, and Arbitrators Researchers exchange vulnerability disclosures for bounties under arbitrators' supervision. In Hyperledger Fabric, organizations own the network and contribute their resources in the form of nodes. Since there is no real word 'researchers' or 'arbitrators' organizations, their nodes are contributed by all participating vendor companies. All three organizations are using the same channel, i.e., one instance of a blockchain. Each organization has one peer node, maintaining a copy of the ledger and is part of the channel. Vendors also maintain a private data collection on the Network channel, allowing it to preserve the privacy of the vulnerability disclosure data by controlling who can access the data, where it is stored, and for how long. The hash value of the private data is written to all of the peer's ledgers and serves as evidence of the transaction, state validation and integrity check in case that the private data is shared with another party in the network, in our use case with the arbitrators. The Hyperledger fabric infrastructure needs few more settings: Orderer node, responsible for transaction ordering with other nodes in ordering service, CouchDB service, peers' state db, and a certificate authority (CA). Each peer with its ledger, the orderer, and the CA runs in their docker container.

[4]https://hyperledger-fabric.readthedocs.io/en/release-1.4/

Table 1: TLA+ simulation results.

Situation	Time	Depth	States Found	Distinct States	Errors
Normal	11'	30	335,459	31,161	0
Arbitration	600'	816	274,486,189	20,908,899	0

A smart contract on Fabric is known as a chaincode. Once the bug bounty smart contracts and arbitrator's contracts are installed on the networks' peer nodes, the embedded logic is ready to be executed via transactions. The user applications interact via transactions to either read or update the ledger state. We used several user applications, written in Java or JavaScript, to invoke the chaincode transactions. Each user has a wallet, stored in the local file system, containing digital identities with certifications issued to him by his organization's trusted CA. To create a new Bug Bounty smart program contract in memory, the application connects to the channel with credentials of permissioned user from Vendor's organization and invoke the following transaction that creates a bug bounty programs and saves its instance to the ledger database.

```
@Transaction
public BugBountySmartProgram issue(BugBountySmart-
ProgramContext ctx, String policy, String owner,
String issueDateTime, int bountyValue, String exp-
irationDateTime, String programNumber);
```

For simplification, the program's policy is represented as a string. The arbitration smart contract similarly allows security researchers to create an arbitration instance with the program's id and voting option. Then, each arbitrator can use its application to invoke the vote transaction, which saves the arbitrators' choice and updates the ledger state.

7 SECURITY PROPERTIES

The proposed framework has the following security properties: *Confidential disclosure and arbitration.* Communication regarding vulnerability disclosure is listed publicly on the blockchain, but it is encrypted so that only the disclosing researcher and relevant vendor can decrypt it (as explained in subsection 4.4). *Immutable and transparent log of communication.* Every message between security researchers and vendors is logged into the blockchain, and accordingly, messages are immutable and transparent, though encrypted. The logging of communication between security researchers and vendors, without specifying its content is a common mean used by bug bounty programs to encourage communication and is not regarded as a data leakage. In cases in which the logging is not desired, one can implement the framework on a private blockchain, where the communication is not logged until a the vendor decides it is safe according to responsible disclosure procedure. *Resiliency to denial of service.* The main components of the framework are the smart contracts and the blockchain, that are resistant to denial-of-service under the assumption of an honest majority. *Feedback transparency.* The accumulative feedback messages are publicly accessible to every researcher, thus discouraging vendors from avoiding participation in arbitration process, and conversely encourage researchers to disclose vulnerability for rewards (as explained in detail in Section 4.5). *Fairness.* A researcher that discloses a valid vulnerability is always rewarded by the relevant vendor, under the assumption of an honest arbitration process (as demonstrated in Section 8).

8 FAIR EXCHANGE USING GAME THEORY

The game theoretic definition of a fair protocol is defined as a case in which no party can improve its utilization by a dishonest behavior, i.e., not following the protocol as specified [20]. In this section we attempt to prove that the proposed platform, when modeled

as game, is fair based on the above definition. For the sake of the proof, the proposed platform is modeled as a game tree model [5].

Game tree models are directed, labeled tree data structures that model every possible action of a party until reaching a leaf node, thus indicating the termination of the game. Every leaf node is a associated with payoff that can be either positive or negative, and the utilization of a party is define as the payoff of a single game. The security researcher and vendors are assumed to be rational i.e., they aim to increase their payoffs by any means, including deviating from the protocol.

In this section, we analyze the payoffs for every possible case of exchanging vulnerabilities for rewards using the framework through a game tree model. We argue that given only two assumption, rationality and feedback transparency, the honest behavior yields the optimal payoff i.e., the protocol for exchanging vulnerabilities for rewards is a Nash equilibrium.

The proposed platform has two parties, security researcher and software company, that wish to exchange a vulnerability information disclosure for a reward. The game plays as follows. Firstly, the software company, denoted by V, publishes a bug bounty smart contract, with x tokens deposited at it. Then, the security researcher, denoted by R, provides information regarding a vulnerability that matches scope of of the bug bounty contract, and its expected reward is $y \mid y \leq x$. After the disclosure,V can either properly reward R or reject the finding. Finally, R can terminate or ask an arbitration process, in which he can either lose or win. The game tree of the described protocol is illustrated in Figure 3.

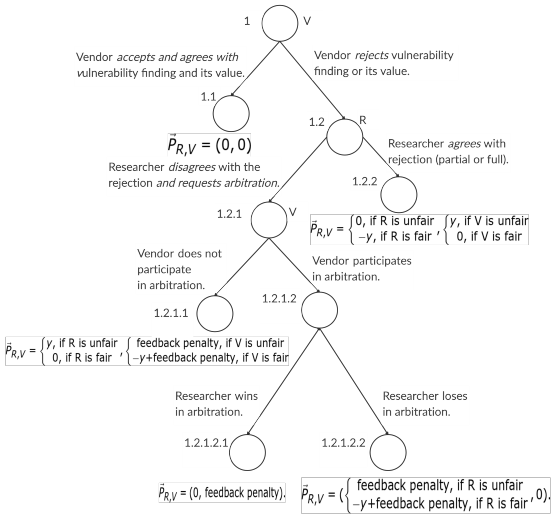

Figure 3: Proposed framework protocol modeled as a game tree.

8.1 Formal notations

Following [5] formal definition, let P be a two-party exchange protocol. Let us consider the game representation of the protocol and denote the two players by A and B. Denote the strategy pair that corresponds to the faithful execution of the protocol by (S_A^*, S_B^*). P is said to be Nash equilibrium fair if:

1. $\underset{P}{\rightarrow}(S_A^*, S_B^*) = (0, 0)$ *and*
2. (S_A^*, S_B^*) *is a Nash equilibrium*

where the requirement for $\underset{P}{\rightarrow}(S_A^*, S_B^*) = (0,0)$ is that:

$$\forall S_B \in \sum_B : p_B(S_A^*, S_B) \leq 0 \quad \textit{and}$$
$$\forall S_A \in \sum_A : p_A(S_A, S_B^*) \leq 0$$

where $\sum_i$ stands for the player strategy set, and p_i stands for the player payoff.

The bug bounty game has two underlying assumptions: *(i)* Both parties are rational i.e., parties will always select the action that results in the maximal payoff. *(ii)* A feedback penalty for software companies (vendors) is worse than every token penalty which is reasonable assuming a recurring game.

8.2 Cases

The game starts by V publishing the bug bounty contract, and R disclosing a security vulnerability within the bug bounty scope to V. V has two possible actions. The honest action, in which V rewards R, and the dishonest action in which V reject the disclosed vulnerability. These two moves are represented by the two edges starting from vertex 1.

8.2.1 Valid vulnerability for a reward. If V rewards R, then the game terminates in vertex 1.1 with a payoff of 0 for R and 0 for V because the vulnerability was exchanged for an equivalent reward, therefore there's no gain nor loss to any of the parties.

8.2.2 Vulnerability, without a reward. If however, V doesn't reward R and rejects the disclosed vulnerability, then R has two possible actions represented by the two edges starting from vertex 1.2. R can either terminate the game or ask for an arbitration process. By terminating the game, R agree with V's rejection and the game terminates in vertex 1.2.2 with the following possible payoffs:

$$\underset{P_{R,V}}{\longrightarrow} = (\begin{cases} 0, & R \textit{ is unfair} \\ -y, & R \textit{ is fair;} \end{cases}, \begin{cases} y, & V \textit{ is unfair} \\ 0, & V \textit{ is fair;} \end{cases}) \tag{1}$$

8.2.3 Valid vulnerability, without a reward. If R disclosed a valid vulnerability but V either rejected it entirely or paid less than the expected reward of y tokens, then R lost at most y tokens. Conversely, in this case, V gained at most y tokens advantage.

8.2.4 Invalid vulnerability, without a reward. Another possible case is that V is refuses to reward R because the vulnerability is invalid, in which V's payoff is 0. Similarly, if R discloses an invalid vulnerability to V, the payoff of R is also 0, because the invalid vulnerability is not worthy of a reward.

8.2.5 Researcher asks for arbitration. If R asks for an arbitration process (see vertex 1.2), then V has two possible actions represented by the two edges starting from vertex 1.2.1, either ignore or participate in the arbitration process

8.2.6 Vendor ignores arbitration. If V decides to ignore the arbitration process, then the game terminates in vertex 1.2.1.1 with the following possible payoffs:

$$\underset{P_{R,V}}{\longrightarrow} = (\begin{cases} y, & R \textit{ is unfair} \\ 0, & R \textit{ is fair;} \end{cases}, \begin{cases} \textit{feedback penalty,} & V \textit{ is unfair} \\ -y + \textit{feedback penalty,} & V \textit{ is fair;} \end{cases}) \tag{2}$$

If V is playing fair and refuses to participate in arbitration, it means that the finding is not valuable to V or not following the program's rules and not worthy for him to supply a testing environment in case the arbitrators can't check in their own environment. Therefore, in this case V's payoff is feedback penalty as he can not ignore framework rules.

8.2.7 Vendor participates in arbitration. If V's chose to participate in the arbitration process (see vertex 1.2.1), there are two possible outcomes, either R wins and V loses, or the opposite. These two outcomes are represented by the two edges starting from vertex 1.2.1.2.

8.2.8 Researcher wins arbitration. If R wins, then the game terminates in vertex 1.2.1.2.1 with the following payoffs:

$$\xrightarrow[P_{R,V}]{} = (0, \textit{feedback penalty}) \tag{3}$$

If R wins for a valid vulnerability, both parties payoff is 0 for the fair trade.

8.2.9 Researcher loses arbitration. If R loses, then the game terminates in vertex 1.2.1.2.2 with the following payoffs:

$$\xrightarrow[P_{R,V}]{} = (\begin{cases} 0 + \textit{feedback penalty}, & R \textit{ is unfair} \\ -y + \textit{feedback penalty}, & R \textit{ is fair}; \end{cases}, 0) \tag{4}$$

If R disclosed a valid vulnerability, then he actually lost maximum y tokens that he worked for. Otherwise, if R disclosed an invalid vulnerability, there's no lose for R. Either way, R is punished with feedback penalty, and V wins in a case in which no trade was ever made, therefore, the payoff is 0 for both parties.

8.3 Summary: Nash Equilibrium

The bug bounty game has only two scenarios in which one of the party improves its utilization compared to the honest scenario: *(i)* V rejects despite R being honest, and R doesn't ask for arbitration (see vertex 1.2.2). This state is unreachable based on the first assumption (rational parties), because an rational honest R must ask for arbitration in which it'll gain an additional payoff. *(ii)* V is dishonest, R asks for arbitration, but V refuses to participate (see vertex 1.2.1.1). This state is unreachable based on the second assumption (feedback penalty is worse than token penalty), because the dishonest vendors must participate in the arbitration to incur in a penalty of tokens rather than feedback. To conclude, neither the vendor or the researcher will gain anything from playing unfairly according to this protocol, and hence the protocol provides the Nash equilibrium fairness.

9 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a secure bug bounty framework that guarantees decentralized trust, fairness, and potentially low commission fees. The system ensures these properties by using smart contracts that enforce the policies agreed by the participating entities, and trusted committee that forces a software vendor to pay the penalty in case of misbehavior. A TLA+ specification based formal evaluation of the proposed framework indicates the viability of the proposal. A Hyperledger Fabric-based prototype simulates the proposed framework in a real scenario. A game-theoretic analysis of the proposed framework indicated that rational parties' use of the framework guarantees honest use of the protocol, thus ensuring fairness and low fees. In the future, we would like to investigate the scalability of the proposal and the costs associated with maintaining such a network. Hyperledger Caliper tool can be used to calculate and compare such statistics to existing solutions.

Acknowledgments

This research was partially supported by the CONCORDIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 830927.

REFERENCES

[1] [n. d.]. HackerOne. https://www.hackerone.com/

[2] [n. d.]. polyswarm. https://polyswarm.io/

[3] Elli Androulaki et. al. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. *Proceedings of the 13th EuroSys Conference, EuroSys 2018* 2018-January (2018). https://doi.org/10.1145/3190508.3190538 arXiv:1801.10228

[4] Lorenz Breidenbach, Phil Daian, Floriantra Er, and Ari Juels. 2018. Enter the Hydra: Towards Principled Bug Bounties and Exploit-Resistant Smart Contracts. *USENIX Security* (2018), 1335–1352. https://www.usenix.org/conference/usenixsecurity18/presentation/breindenbach{%}0Ahttps://thehydra.io/paper.pdf

[5] Levente Buttyan and Jean-Pierre Hubaux. 1999. Toward a Formal Model of Fair Exchange - a Game Theoretic Approach. December 1999 (1999), 1–16.

[6] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. 2007. Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge. *IEEE Transactions on Software Engineering* 33, 3 (2007), 171–185. https://doi.org/10.1109/TSE.2007.26

[7] Cohen Fred. 1999. Simulating cyber attacks, defences, and consequences. *Computers& Security* 18, 6 (1999), 479–518.

[8] M Crosby, P Pattanayak, S Verma, and V Kalyanaraman. 2016. Applied Innovation Review. *Applied Innovation Review* 2 (2016), 5–20.

[9] William Crumpler and James A Lewis. 2019. The Cybersecurity Workforce Gap. *Center for Strategic and International Studies* July 2016 (2019), 1–10. http://www.isaca.org/Knowledge-Center/

[10] Hackerone. 2019. *Hacker Report.* Technical Report. 54 pages.

[11] Shuang Hu, Lin Hou, Gongliang Chen, Jian Weng, and Jianhua Li. 2018. Reputation-based distributed knowledge sharing system in blockchain. *ACM International Conference Proceeding Series* (2018), 476–481. https://doi.org/10.1145/3286978.3286981

[12] Vijayan Jai. 2017. Bug Bounty Programs are Growing Up Fast and Paying More. https://www.darkreading.com/vulnerabilities---threats/bug-bounty-programs-are-growing-up-fast-and-paying-more/d/d-id/1328428

[13] Jay P Kesan and Carol M Hayes. 2016. Bugs in the Market: Creating a Legitimate, Transparent, and Vendor-Focused Market for Software Vulnerabilities. *Ariz. L. Rev.* 58 (2016), 753.

[14] Markus Klems, Jacob Eberhardt, Stefan Tai, Steffen Härtlein, Simon Buchholz, and Ahmed Tidjani. 2017. Trustless Intermediation in Blockchain-Based Decentralized Service Marketplaces BT - Service-Oriented Computing, Michael Maximilien, Antonio Vallecillo, Jianmin Wang, and Marc Oriol (Eds.). Springer International Publishing, Cham, 731–739.

[15] Andreas Kuehn and Milton Mueller. 2018. Analyzing Bug Bounty Programs: An Institutional Perspective on the Economics of Software Vulnerabilities. *SSRN Electronic Journal* (2018), 1–16. https://doi.org/10.2139/ssrn.2418812

[16] Leslie Lamport. 2002. *Specifying systems: the TLA+ language and tools for hardware and software engineers.* Addison-Wesley Longman Publishing Co., Inc.

[17] Aron Laszka, Mingyi Zhao, Akash Malbari, and Jens Grossklags. 2018. The rules of engagement for bug bounty programs. In *International Conference on Financial Cryptography and Data Security*. Springer, 138–159.

[18] Tianxiang Lu, Stephan Merz, and Christoph Weidenbach. 2011. Towards verification of the pastry protocol using TLA+. In *Formal Techniques for Distributed Systems*. Springer, 244–258.

[19] Loi Luu, Duc Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. *Proceedings of the ACM Conference on Computer and Communications Security* 24-28-Octo (2016), 254–269. https://doi.org/10.1145/2976749.2978309

[20] Adrienne Margolis. 2000. Fair exchange. *Engineer* 289, 7510 (2000), 22. https://doi.org/10.7748/ns2014.04.28.32.27.s32

[21] Luciana Obregon. 2019. Information Security Reading Room A Technical Approach at Securing SaaS using Cloud Institute. (2019).

[22] Pascal Schulz, Leonardo Martucci, and Donald F Ross. 2014. Penetration Testing of Web Applications in a Bug Bounty Program. (2014). http://www.diva-portal.org/smash/get/diva2:723516/FULLTEXT02

[23] Florian Tramèr, Fan Zhang, Huang Lin, Jean Pierre Hubaux, Ari Juels, and Elaine Shi. 2017. Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge. *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017* (2017), 19–34. https://doi.org/10.1109/EuroSP.2017.28