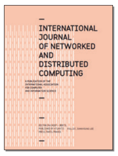
International Journal of Networked and Distributed Computing
Vol. 9(2-3); April–July (2021), *pp.* 86–93
DOI: https://doi.org/10.2991/ijndc.k.210527.001; ISSN 2211-7938; eISSN 2211-7946
https://www.atlantis-press.com/journals/ijndc



Research Article

Bountychain: Toward Decentralizing a Bug Bounty Program with Blockchain and IPFS

Alex Hoffman[*], Phillipe Austria, Chol Hyun Park, Yoohwan Kim

Howared Hughes School of Engineering, Department of Computer Science, University of Nevada, 4505 S. Maryland Parkway, Las Vegas, NV 89154, USA

ARTICLE INFO

Article History
Received 22 February 2021
Accepted 25 May 2021

Keywords
Bug bounty
blockchain
IPFS
decentralized apps
DApps

ABSTRACT

Bug Bounty Programs (BBPs) play an important role in providing and maintaining security in software applications. These programs allow testers to discover and resolve bugs before the general public is aware of them, preventing incidents of widespread abuse. However, they have shown problems such as organizations providing accountability of reporting bugs and nonrecognition of testers. In this paper, we discuss Bountychain, a decentralized application using Ethereum-based Smart Contracts (SCs) and the Interplanetary File System (IPFS), a distributed file storage system. Blockchain and SCs provide a safe, secure and transparent platform for a BBP. Testers can submit bug reports and organizations can accept or reject the defect via the SCs. Transactions on the blockchain serve as a persistent and transparent record of software bugs, while IPFS serves as a long-term storage system for bug details. Thus, Bountychain ensures organization accountability and allows testers to gain irrefutable recognition.

1. INTRODUCTION

This paper is an extension of work originally presented at the 10th Annual Computing and Communication Workshop and Conference [1].

The first Bug Bounty Program (BBP) was launched in 1995 by Netscape [2]. The company created the program to discover pre-release software defects in Netscape Navigator, which gave rise to a new paradigm of security research. Other companies and organizations were slow to adopt a similar program at first; however, adoption began to increase in 2002. Organizations either formed their own BBP or joined existing programs from third-party providers. Today, BBPs provide a platform for people of all skill-levels to ethically find, report, and get paid for discovering security vulnerabilities.

Typically, organizations must create a BBP before individuals can search for vulnerabilities without legal repercussions. Doing so beforehand is potentially a violation of the organization's license agreement [3]. Those who participate in the program are known as bug hunters, bug bounty hunters, security researchers, white-hat hackers, or, for the purposes of this paper, testers. For a tester to be considered ethical or litigation-safe, they should only search for vulnerabilities from companies that have agreed to host such a program.

Once such a program is established, a tester can begin searching for bugs within the bounds of the given BBP. When a bug is discovered, the tester creates a report and submits it through the system as described in the program. The report is evaluated, and the company decides if the submission is acceptable. If so, the tester is typically rewarded for their efforts.

There are two issues that bug bounty programs currently have. First, organizations are not held accountable for poor reporting of security issues. For example, Uber, the ride sharing service, was breached in 2016, exposing the credentials of over 57 million users [4]. However, the company did not disclose the breach until a year after the incident. Breaches involving users' privacy or personally identifying information must not be withheld and must remain transparent according to security breach notification law [5].

Second, testers are susceptible to nonrecognition. A successful BBP relies on good faith and incentivization. Testers hope to receive a recognition along with a reward for finding a vulnerability. In return, an organization is made aware of a security flaw that might cost much more than the bounty payout [6].

An implementation of a BBP on a public blockchain with a successful token economy can guarantee a timely payout while maintaining transparent and persistent records of bug submissions. Thus, it would hold organizations accountable for security breaches and ensure recognition of testers. Additionally, blockchain transactions can prove that security issues were addressed and when they were addressed to help mitigate against any potential investigations or litigation from regulating authorities.

For those reasons we propose Bountychain, a security bounty management program that utilizes Smart Contracts (SCs) on the Ethereum blockchain and a decentralized storage system called the Interplanetary File System (IPFS). Bountychain aims to be an automated, third-party managed platform that provides testers the

[*]*Corresponding author. Email: alex.hoffman@unlv.edu*
Data availability statement: The data that support the findings of this study are available from the corresponding author, [AH] upon reasonable request

ability to easily report security vulnerabilities and gives companies an easy, guaranteed mechanism to send payouts to testers.

In our implementation, the tester chooses the reward value derived from a matrix based on the bug's severity, impact to users and likelihood of occurring [6]. Once the bug report is addressed, the company can accept the report as valid, reject the report as invalid or duplicate, or edit the reward value. If the report is accepted, the smart contract will automatically and immediately distribute payment to the tester. Lastly, any report accepted by the organization is saved on IPFS for immutable storage.

The remainder of this paper is structured as follows. Section 2 presents related studies using blockchain for BBPs. Section 3 discusses background about BBPs, Ethereum, IPFS, and other web technologies used to build Bountychain. In Section 4, we provide a comprehensive overview of Bountychain's architecture, accompanying SCs, and web framework. Section 5 presents some limitations while Section 6 discusses some mitigations. We conclude with Section 7.

2. RELATED WORKS

Using blockchain for BBP is a new topic, and relatively few studies have been published so far. Canidio et al. [7] proposed VeriOSS, a blockchain-based platform for bug bounties. The authors address a particular challenge in BBPs, which is that organizations lack commitment with respect to the eligibility of the bugs. This leads testers to look for other opportunities in other markets, such as gray and black markets [8]. The goal of VeriOSS is to increase the reward for testers to foster more bug hunting and consequently, decrease the appeal of grey and black markets.

Another study proposed the Hydra Framework, a general, principled approach to modeling and administering bug bounties that incentivizes bug disclosure [9]. The core idea behind the framework is an exploit gap, a program transformation that enables runtime detection and rewarding of critical bugs. The goals of Hydra Framework are to solve issues in BBP such as pricing bounties, due to lack of research giving principled guidance and market inefficiency, caused by the uncertainty of a bug reward being awarded.

Wang et al. [10] proposed a new consensus protocol for a blockchain platform in medical records that considered the capability and credibility of software testers called Proof of Skill. The authors set out to create a new protocol in response to weaknesses in BBPs such as transparency and risk of rewards not being paid out by organizations.

Sentinel protocol is a blockchain-based BBP that helps facilitate decentralized detection and reporting of threat defects [11]. While the operation is like that of other third-party BBPs, this program focuses on the cryptocurrency security market. Sentinel Protocol does not provide their BBP as a service to non-cryptocurrency-based companies.

As far non-blockchain related BBP, several major software companies host their own program. Google, Facebook, and Microsoft run private BBPs [12–14]. Alternatively, companies such as Netflix use trusted third parties to issue bugs. The most notable third-party platforms are HackerOne, Bugcrowd and Cobalt [15–17].

3. BACKGROUND

In this section, we discuss how BBPs operate and the technologies used to develop Bountychain, such as blockchain and IPFS. We specifically used the Ethereum blockchain due to its ability to support SCs.

3.1. Bug Bounty Programs

Bug Bounty Programs leverage crowd sourcing to discover and report software vulnerabilities [18]. They are generally comprised of the following:

- Organization: an organization can be a company, university, group of people etc., who allow their software to be tested to uncover bugs.
- Tester: a tester, also known as a bug hunter, bug bounty hunter, security researcher, or white-hat hacker, searches for bugs found in an organization's software.
- Platform: platform represents a third-party that hosts organizations bounty hunting directives and allows people to sign up to become testers.
- Bug: a bug is a security vulnerability or defect in an organization's software.
- Bug Report: a report written and submitted by a tester describing a bug in detail.
- Rewards: compensation given by the organization to the tester for discovering a bug in their software.

There are currently two types of BBPs: internally managed programs and third-party managed programs [8]. Figure 1 shows how organizations and testers interact between the two types of programs.

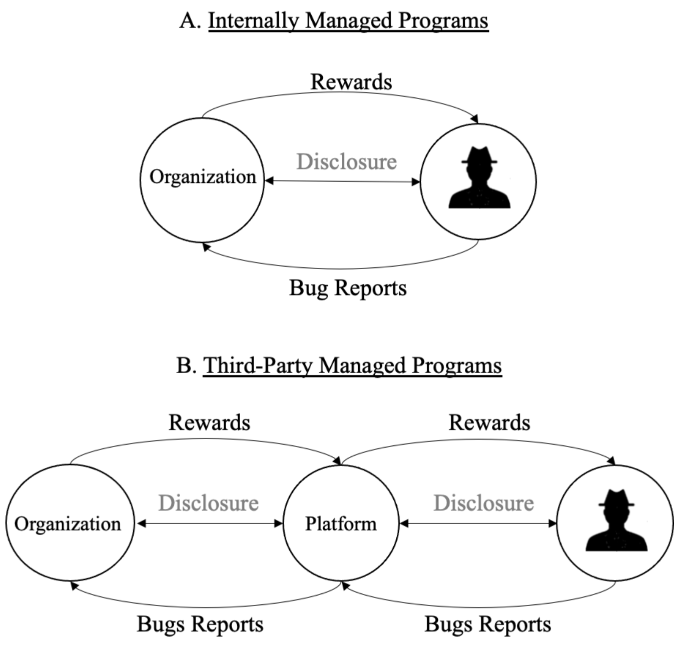


Figure 1 | Bug Bounty Programs types.

Internally managed programs are hosted by an organization. Technology giants such as Facebook, Google and Microsoft host direct programs. Testers submit bug reports directly back to the organization.

Third-party managed programs enable organizations and testers perform actions relative to each entity's distinct needs [19]. Organizations subscribe to platforms and publicly list which of their applications they want tested. Platforms normally charge organizations a fee for the subscription. Testers sign up with the platform and are given the freedom to submit a bug relating to an organization of their choice. Unlike internally managed programs, bug reports are submitted through the platform and forwarded to the corresponding organization.

3.2. Blockchain

Blockchain is a data structure invented by Nakamoto [20] and serves as the public transaction ledger for the Bitcoin cryptocurrency. Blockchain functions as an immutable data store and is comprised of cryptographically linked groups of transactions called blocks. Blocks contain the following information: the previous block-hash, transactions, transaction timestamps, and the current block-hash. The current block-hash is a hash of all the data inside the block and serves as the block's identifier.

Because the previous block-hash partially determines the current block-hash, each block is cryptographically connected. The first block, known as the Genesis Block, is the only block that does not contain a previous block-hash. If any data is changed in a block, its block-hash would change; consequently, every subsequent block's hash would also change. This mechanism provides an efficient method to detect data mutation and enables the blockchain to be tamper resistant.

There are two types of blockchains regarding access control; choosing the correct type for an application's purpose is important. The first is a public blockchain, also known as a permissionless blockchain. Anyone can join the network and participate within the blockchain [21]. Bitcoin and Ethereum are examples of a public blockchain.

The second type of blockchain is a permissioned blockchain, also referred to as a private blockchain chain. These blockchains work based on access controls, which restrict the people who can participate in the network [22]. Participants in the network are known, although transactions can remain hidden or anonymized from other participants depending on the implementation of the blockchain. Hyperledger Fabric, Corda and Quorum are examples of permissioned blockchains [23–25].

Recent research discusses blockchains that have attributes of both permissionless and permissioned access known as hybrid blockchains [26–28]. We use the definition given in Freuden [29] to describe a hybrid blockchain and its architecture. A hybrid blockchain consists of a public blockchain and a private network. The private network is comprised of participants who have been invited by a centralized body and the transactions generated by the network are stored and verified on the public blockchain.

3.3. Ethereum

Ethereum is a public, open source blockchain that natively supports SCs. An aim of Ethereum is to provide developers with a tightly integrated end-to-end system to build software on a novel system: a trustful object messaging computer framework [30]. The combination of being a public blockchain chain and the ability of each node to execute SCs makes Ethereum ideal for building Decentralized Applications (DApps).

Smart contracts in Ethereum are defined as applications that run exactly as programmed without any possibility of downtime, censorship, fraud, or third-party interference [30]. Ethereum created its own programming language for developers to write smart contracts called Solidity [31]. Once compiled into bytecode and uploaded to an Ethereum node, each participating node then stores, validates, and executes the SC, which creates a true decentralized application.

Additionally, SCs require a fee, known as gas, to be deployed, validated, and executed. Gas, which is priced in subunits of ether, is consumed in return for computation resources allocated from nodes. Gas costs are directly correlated to the complexity of the SC, internal SC functions, and the desired execution time. Paying more gas will reduce the execution time [32].

3.4. Web3.js

Interaction with Ethereum and a SC from within a DAapp, such as a web application, requires a JavaScript library called Web3.js. Web3.js is the Application Programming Interface (API) for the Ethereum blockchain [33]. The library allows actions such as creating new contracts, deploying contracts, and interacting with existing contracts. Web3.js provide applications the capability to send data to and receive from the Ethereum blockchain. The most common received data is transaction IDs to enable event tracking.

3.5. Interplanetary File System

Interplanetary File System is a protocol that uses a peer-to-peer network to create a distributed file system [34]. Machines running IPFS, called nodes, hold a portion of the overall data, creating a decentralized and resilient network. Similar to BitTorrent, IPFS nodes can store, serve, and share files [35]. If a node goes offline and was serving a file of interest, the file is still available for access as long has one other online node has the file.

Unlike location-based addressing used in traditional file systems, IPFS utilizes content-based addressing. When a node initially serves a file, the file is partitioned into 256 KB pieces called blocks. Files that are smaller than 256 KB are padded with filler bytes. Each block is hashed using a variation of the SHA256 hashing algorithm, then each pair is re-hashed from a Merkle Tree; the root of the tree is the Content Identifier (CID), which is shown in Figure 2. The CID additionally ensures security of the file because any changes to a file's content would change the hash of the file. Thus, changes are readily apparent due to differing file hashes. Most importantly, IPFS utilizes the CID as the file's address on the network.

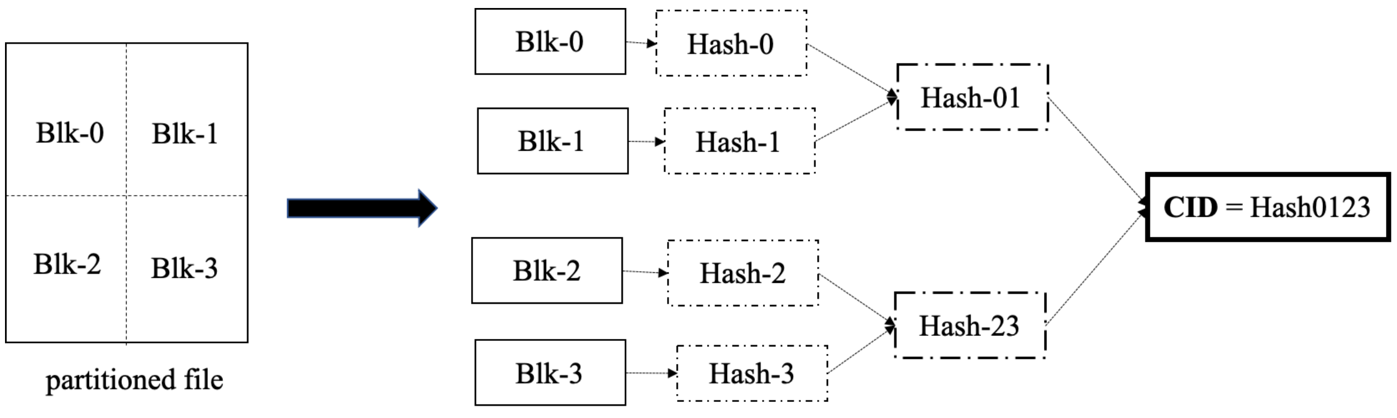


Figure 2 | File partitioned and hashed into a Merkle Tree to form an IPFS file CID.

Interplanetary File System aims to solve the centralized model used in most services today. When used with blockchain, large amounts of data can be addressed and stored in a secure, decentralized and persistent fashion.

3.6. Metamask

Metamask is a browser add-on that acts as a bridge between the internet browser, Ethereum, and DApps built on Ethereum [36]. In combination with Web3.js, Metamask enables direct Ethereum interaction without needing to keep a local copy of the blockchain, allowing for faster and lighter development environments. The add-on provides wallet management, which makes it possible to pay for gas costs when creating and interacting with SCs. These features allow people to use Ethereum-based DApps as if they were normal web applications.

4. BOUNTYCHAIN ARCHITECTURE

4.1. System Components and Workflow

The steps below and Figure 3 provide a high-level workflow of how Bountychain works as well as why certain technical decisions were made.

1. Register to be a Tester and Discover a Bug: A user signs up on Bountychain to become a tester. A tester submits a bug report through the Bountychain portal once a bug is discovered within an organization's software.
2. Instantiate the Smart Contract: After a bug report is submitted by the tester, Bountychain writes the data into its local database. The local database allows testers to retrieve and make updates to bug reports. Additionally, only testers and organizations can see bug reports. Bountychain then creates a Keccak hash of the bug report. The SC factory is called, and the hash is stored in the contract along with the tester's Ethereum address, the organization's Ethereum address, and a proposed reward value.
3. Organization Views the Bug: After the organization is notified of a new bug, they log into Bountychain to review the report. Additionally, the organization may view the proposed payout value set by the tester.

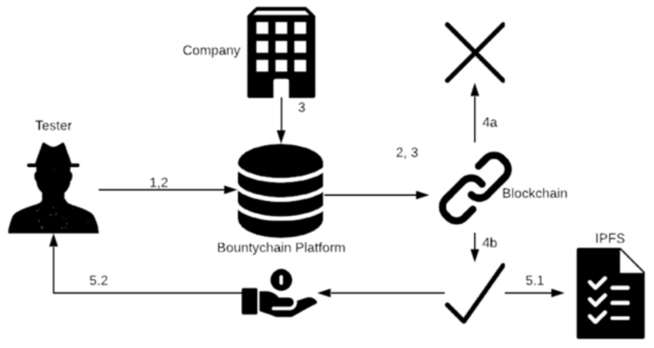


Figure 3 | Bountychain workflow.

4. Organization Rejects (4a) or Accepts/Modifies (4b) the Bug Report: An organization can reject a submission if they determine the bug and its report invalid. This action immediately closes the submission, and the smart contract locks in the rejected state. If the organization decides the bug is valid, they can then apply a patch and deploy it into production. Once deployed, the organization can accept the bug in Bountychain. Organizations may also choose to modify the reward value. If so, the reward value changes, and the bug is automatically accepted. In either case, the smart contract switches to the accepted state.
5. Accepted and Payout: The bug submission and its report are written to IPFS once accepted. The Keccak hash from step 2 is used as the IPFS file name; files shared on IPFS are available to the public, regardless if they have access to the Bountychain platform. In addition, the SC automatically pays out the reward to the tester who discovered the bug.

4.2. Ethereum Smart Contract Implementation

Bountychain utilizes the Ethereum blockchain because of its prominence and native support of SCs. In the platform's primary SC, there exist two contracts for Bountychain to utilize, the *contractFactory* and the *BugSubmission* contract. The *contractFactory* (Algorithm 1) deploys *BugSubmission* contracts (Algorithm 2). A page exists in the front end where testers can fill out a form

```
Algorithm 1 Create Contract
Input: Tester Address
Output: Create Contract
   Constructor for Contract :
1: Tester Address
2: Company Address
3: Hashes Pay
   Submit Contract :
4: Tester Address
5: Company Address
6: Hashes
7: Pay
8: Push contract address
9: emit address of Created Contract
```

Algorithm 1 | *contractFactory.*

```
Algorithm 2: Bounty Submission
  Input: pay amount, bug
  Result: Save bug into IPFS, Reward amount or deny
  Constructor for contract :
  if Company deny bug then
      return Close case
  else
      Record bug in IPFS
      if Company modify pay amount then
          return Company pays modified amount to
           tester.
      else
          return Company pays tester.
      end
  end
```

Algorithm 2 | *BugSubmission* contract.

to submit a bug report. After a tester submits the report, a new instance of the *BugSubmission* contract is created. The following values are collected and saved in the contract: the hashed summary of the report, the tester's public Ethereum address, the organization's public Ethereum address, and the expected compensation. The values are immutable once saved.

The web interface for the Dapp provides organizations the ability to read bug submissions and the ability to accept, modify, or reject unresolved contracts. A contract can be modified when the company perceives the report is valid but the severity and/or frequency which it occurs was not accurately assessed by the tester who reported it. Thus, the organization can give a reason for modification and change the payout value. A modified contract is treated the same as an accepted contract.

Denying a bug submission is simplistic because it does not transfer funds. It sets the summary reason to the reason the organization denied the bug and sets the status to contract denied. A contract is no longer active once in the accepted state or the denied state, which prevents tampering or nefarious actions by malicious actors.

4.3. Web Front End

Bountychain's web front end was developed using a popular JavaScript library called React. Additionally, the Axios library was used for interacting with the API server. The front end was deployed on an AWS micro instance running Ubuntu 18.04 LTS.

4.4. API Server

The prototype was built in Python 3.6. The Tornado Web Server [37] and Tornado-JSON package [38] were used to serve RESTful APIs and interact with IPFS. A PostgreSQL database was used to store structured data. These choices enabled a permissioned system that help protect user privacy and provide high throughput. Both the API server and PostgreSQL database were deployed in an AWS micro instance running Ubuntu 18.04 LTS.

5. LIMITATIONS AND SOLUTIONS

Although the first working blockchain was created in 2008 by the person or group named Nakamoto [20], it has been theorized since the early 1990s, it continues to have limitations and drawbacks. Blockchain is ideal for creating immutable records, but there is a cost to perpetually archiving these records. In most implementations on a public blockchain, the entire blockchain is stored locally on every mining node. The constant growth of a blockchain and the requirement to perpetually expend energy to mine to verify new transactions limits the potential to store data on the blockchain. Effectively, it adds a cost burden for every transaction on the blockchain.

In this implementation or any such future work, a tester would preferably not have to pay to submit a bug to an organization. Due to the fundamental rules of Ethereum as well as our desire to prevent this implementation from requiring our involvement, testers must pay for a *BugSubmission* SC to be put on the blockchain. Gas cost to deploy the *BugSubmission* SC was reduced via optimization of the SC, but costs could not be fully eliminated. We were also not able to shoulder the burden of the gas costs due to the inherent security properties of the Ethereum blockchain because only the originator of an SC, in this case a tester, can pay for the SC to be deployed [39]. While there are workarounds for this, it would mean putting more burden and management overhead on the Bountychain solution. This would also have the ramification of increasing gas costs due to additional computations. At the same time, the DApp operator would need additional security overhead as well as turn an otherwise transparent process opaque. Thus, outside of using a different blockchain than Ethereum, the gas cost must be paid by the users of the system rather than the system itself, which is why we emphasize the gas cost reduction at this stage. While Ethereum is not the only blockchain with the advantages of having SC interaction, advantages and disadvantages of using alternate blockchains is reserved for a future work.

Another limitation of the Ethereum blockchain is that it is not directly human-readable. One must be able to work with bytecode [30]. It is a different experience to try to interpret and interact with an address on the Ethereum chain relative to the ease of invoking a web URL and interacting with a website. The storage mechanism was not designed for readability, so the Ethereum bytecode must be somehow translated to human-readable text to see what is stored in any given transaction. A layer must be added between the consumer, e.g. tester or organization, and the blockchain to facilitate it. Web3.js library and the Metamask browser add-on were utilized in our implementation. These two technologies enable simple, highly readable interaction with the Ethereum blockchain, but there are inherent security risks with using additional technologies. That subject is also saved for a future work.

The property of the Ethereum blockchain that is the most limiting as well as the most useful is immutability. For example, all information about reported bugs would be stored on the blockchain in perpetuity, even if the Bountychain system was terminated. The primary issue is the cost of this storage, even though it is this inherent property that makes using blockchain so compelling. Additionally, even though the records are not directly human-readable, the fact that the records are transparent to the public and can be interpreted with software assistance is beneficial. By only storing the hash of the bug on the chain, we were able to take full advantage of these properties while mitigating the cost factor. Conversely, this means we cannot rely solely on blockchain. While it would be ideal to have a system that could exist entirely on the chain, the factors of cost, security, and privacy makes such a solution impractical.

6. DISCUSSION

6.1. Gas Cost Limitations

The primary limitation of Ethereum is gas cost. Interacting with SCs requires paying gas fees and contracts with greater complexity use more gas. Table 1 shows the cost to deploy an SC with specific variables. To collect the gas costs, we deployed a SC that only contained the variable as specified in each row along with a single setter function. We recorded the gas cost to deploy the SC with that single variable, and the gas cost to call the setter function to set the variable. While optimized and non-optimized contracts produce the same results, non-optimized contracts are more costly. Non-optimized SCs are a primary source of unnecessary gas costs [40].

Storing a bug submission hash is a useful demonstration of SC optimization. We opted to store the hash as two byte32 variables rather than a string of 80 characters; this saved approximately 21,164 gas units, a 19.6% reduction. Setting a variable as type uint costs approximately twice that of setting an enum variable. Interestingly, smaller data types are not necessarily cheaper in gas.

Table 1 Gas cost comparisons

Data type	Gas to deploy	Gas to set
string (2000 chars)	268,328	1,443,697
string (280 chars)	268,328	242,592
string (180 chars)	268,328	175,516
string (100 chars)	268,328	129,988
string (80 chars)	268,328	108,510
byte32	113,355	43,673
byte8	157,381	42,404
uint	113,355	42,009
enum	95,919	26,686

Continuing with the previous example, if the bug submission hash is stored as eight byte8 variables, equivalent in storage space to the two byte32 variables, the gas cost would have been 339,232, a 1503% increase.

An optimized SC saved significant gas both to deploy and update itself. Using Table 1, we optimized our SC to be as efficient as possible, especially with regard to utilizing enums and minimizing functions. Double-spending on deploying SCs and initializing data were avoided by having a contract factory. The cost of deploying the *contractFactory*, paid by Bountychain, was reduced by 6.87% through contract optimization. Deploying individual *BugSubmission* contracts, paid by testers, was reduced by 5.94% (Figure 4). This is significant for development because of the goal to maintain a hands-off approach. The *contractFactory* is only deployed once and costs a small amount of gas to create a template that generates as many bug submissions as needed. The gas reduction to deploy *BugSubmission* contracts is also important. Over time, the optimized contract saves testers a significant amount of gas costs.

6.2. Storage Limitations

Managing the cost of storage is a key component in our implementation. In Table 1, string has a high deployment cost, and the gas to set increases as length of the string increases. Storing a short bug report summary on the Ethereum blockchain costs 1,443,697 in gas. Today, one gas is about 42 gwei and at \$3.4E-7/gwei; the cost to store such report is estimated at \$20.[1] This cost is large compared to traditional server storage, where one can buy several gigabytes of storage for the same price. Additionally, there is also no bytecode-to-human-readable translation required in traditional datastores, which is another advantage such datastores have over storing data in a SC. High storage costs and the need for bytecode translation make medium-to-large file storage in the Ethereum impractical.

For that reason, IPFS was chosen to work in conjunction with the Ethereum SCs. IPFS allows us to host plain-text bug summaries for a fraction of the cost of storing it in the chain itself, and this cost is not required to be incurred by the platform, tester, nor organization. IPFS helps solve most public blockchains' storage issues.

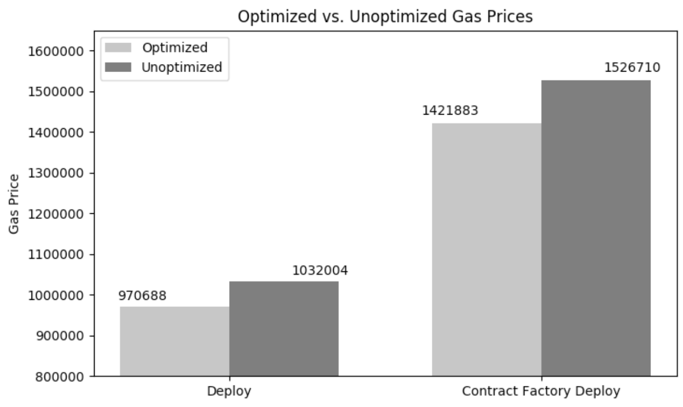


Figure 4 Contract and *contractFactory* deploy gas costs.

[1]https://www.cryps.info/en/Gwei_to_USD.

6.3. Incentives in a Token Economy

The Token Economy is a cryptocurrency-based system of incentives that builds and supports beneficial conduct from actors in a blockchain ecosystem [22]. In such a system, all parties are incentivized to perform and promote ethical behavior to produce and maintain mutually beneficial outcomes, which why we chose Ethereum for this project. Because Ethereum is widely supported, the chances of a sudden loss of the entire ecosystem is unlikely, whereas a newly created blockchain built specifically for this project would have a lesser chance of surviving. Hence, building upon Ethereum provides more trust and reliability than a newly developed system.

While the incentives for individuals have been discussed, testers can also be paid automatically, quickly, and reliably. Blockchain systems like Ethereum are also excellent options for the unbanked. People are shielded from wire transfer and international currency conversion fees. Finally, APIs could allow third parties to publish accepted bugs to the Bountychain system. The intent would be to provide a single aggregated, cross-platform system for testers to receive recognition, rather than functioning as a payment processing system. Furthermore, the reports could be stored on IPFS to ensure availability in an immutable and reliable manner.

There are multiple potential incentives for organizations to participate. As mentioned earlier, organizations can use bug reports as evidence in an investigation. Financially, Bountychain provides several cost-saving incentives. Most organizations pay for third-party organizations to manage their BBP. Bountychain reduces that financial burden. In addition, organizations that perform internal bounty management programs will have a lower cost of management overhead both in reducing platform management and in financial management for processing payments.

There are several incentives for organizations to act fairly. The purpose of organizations joining BBPs is to incentivize testers to help ethically discover vulnerabilities. Testers help to find bugs that could otherwise be exploited; hence, organizations have a strong reason to keep such skilled people happy and acting ethically. An organization that does not follow through with paying the reward is unlikely to engender good will with testers. Testers who are not treated fairly may seek alternate markets for any bugs they find, which, as noted earlier, could lead to bugs being sold on the black market. Additionally, organizations want to attract the best testers they can to find security flaws. Recording an organization's blockchain address and reward amount in each report allows the public to see how often and how much each organization pays out. The organizations that pay testers most frequently and lucratively will gain more testing support, and those organizations that try to cheat or regularly depreciate its tester's report values will likely lose support and credibility.

7. CONCLUSION

In this paper we propose Bountychain, a decentralized BBP built on the Ethereum blockchain and IPFS. We discuss Bountychain's implementation and addressed the motivation for choosing Ethereum and IPFS. Ethereum SCs allows organizations and testers to make automated agreements, while IPFS stores data in a distributed and highly available manner. Additionally, we highlight how Bountychain solves two issues faced by current BBPs: inadequate security vulnerability reporting by organizations and insufficient compensation and recognition given to testers. Transparency is inherent to the public Ethereum blockchain, and once a defect is submitted to the system, Ethereum SCs ensure tester recognition and rewards.

High gas costs and low storage on the Ethereum blockchain was an initial issue for Bountychain. Efficient programming and prudent data types resolves high gas costs. Moreover, IPFS allowes for mass storage which is not offered by Ethereum. Overall, our model solves the current underlying issues of traditional BBPs. However, additional research on blockchain and IPFS is needed to ensure Bountychain is capable of becoming a model that modern BBPs can follow.

7.1. Future Works

For comparison, we could implement the same system on Hyper Ledger Fabric instead of Ethereum blockchain. There are positives and negatives to using Hyper Ledger Fabric. As a positive, Hyper Ledger Fabric can perform more transactions per second. It can perform than 3500 transactions per second, while Ethereum can only perform nine transaction per second. As a negative, Hyper Ledger is private. While it might be for convenient for an organization to make all bug bounties private, this goes against the philosophy of the BBPs in general. Additionally, Ethereum can be made public or private. Hyper Ledger does not have a cryptocurrency system. Therefore, if we build same system with Hyper Ledger, we need to find the way to pay testers. Such an implementation would be for comparative results rather than demonstrating Hyper Ledger Fabric to be a better solution.

CONFLICTS OF INTEREST

The authors declare they have no conflicts of interest.

AUTHORS' CONTRIBUTION

AH and PA contributed in study conceptualization and writing (review & editing) the manuscript, formal analysis and writing (original draft). AH and CHP contributed in data curation. YK supervised the project. AH contributed in final editing.

REFERENCES

[1] A. Hoffman, E. Becerril-Blas, K. Moreno, Y. Kim, Decentralized security bounty management on blockchain and IPFS, 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), IEEE, Las Vegas, NV, USA, 2020, pp. 241–247.

[2] E. Friis-Jensen, The history of bug bounty programs, Medium, 2014. Available from: https://blog.cobalt.io/the-history-of-bug-bounty-programs-50def4dcaab3 (accessed October 22, 2019).

[3] M.A. Davidson, No, you really can't, Oracle, 2015. Available from: https://web.archive.org/web/20150811052336/https://blogs.oracle.com/maryanndavidson/entry/no_you_really_can_t (accessed October 22, 2019).

[4] B. Chappell, Uber pays $148 million over yearlong cover-up of data breach, NPR, 2018. Available from: https://www.npr.org/2018/09/27/652119109/uber-pays-148-million-over-year-long-cover-up-of-data-breach (accessed October 22, 2019).

[5] National Conference of State Legislatures, Security breach notification laws, 2020. Available from: https://www.ncsl.org/research/telecommunications-and-information-technology/security-breach-notification-laws.aspx (accessed September 18, 2020).

[6] B. Zhou, I. Neamtiu, R. Gupta, Experience report: how do bug characteristics differ across severity classes: a multi-platform study, 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), IEEE, Gaithersbury, MD, USA, 2015, pp. 507–517.

[7] A. Canidio, G. Costa, L. Galletta, VeriOSS: using the blockchain to foster bug bounty programs, 2nd International Conference on Blockchain Economics, Security and Protocols, Schloss Dagstuhl, Germany, 2021, pp. 1–14.

[8] A. Hoffman, H. Berghel, Moral hazards in cyber vulnerability markets, Computer 52 (2019), 83–88.

[9] L. Breidenbach, P. Daian, F. Tramèr, A. Juels, Enter the hydra: towards principled bug bounties and exploit-resistant smart contracts, Proceedings of the 27th USENIX Conference on Security Symposium, 2018, pp. 1335–1352.

[10] Y. Wang, R. Samavi, N. Sood, Blockchain-based marketplace for software testing, 2019 17th International Conference on Privacy, Security and Trust (PST), IEEE, Fredericton, NB, Canada, 2019, pp. 1–3.

[11] Uppsala Security, Protect Your Cryptocurrencies with Advanced Software Solutions from Uppsala Security! Available from: https://uppsalasecurity.com/ (accessed October 22, 2019).

[12] Google Application Security, Google Security Reward Programs. Available from: https://www.google.com/about/appsecurity/programs-home/ (accessed October 22, 2019).

[13] Facebook, Whitehat program. Available from: https://www.facebook.com/whitehat (accessed October 22, 2019).

[14] Microsoft, Microsoft bug bounty program. Available from: https://www.microsoft.com/en-us/msrc/bounty (accessed October 22, 2019).

[15] Hackerone, Hacker-powered security testing & bug bounty. Available from: https://www.hackerone.com/ (accessed September 20, 2020).

[16] Bugcrowd, Crowdsourced cybersecurity platform. Available from: https://www.bugcrowd.com/ (accessed September 20, 2020).

[17] Cobalt, Pentest as a service. Available from: https://cobalt.io/ (accessed September 20, 2020).

[18] S.S. Malladi, H.C. Subramanian, Bug bounty programs for cybersecurity: practices, issues, and recommendations, IEEE Softw. 37 (2020), 31–39.

[19] H. Fryer, E. Simperl. Web Science Challenges in Researching Bug Bounties, Proceedings of the 2017 ACM on Web Science Conference (WebSci '17), Association for Computing Machinery, New York, NY, USA, 2017, pp. 273–277.

[20] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, Bitcoin.org, Oct. 2008. Available from: https://bitcoin.org/bitcoin.pdf (accessed October 21, 2019).

[21] T.K. Sharma, Public vs. private blockchain : a comprehensive comparison, Blockchain Council. Available from: https://www.blockchain-council.org/blockchain/public-vs-private-blockchain-a-comprehensive-comparison/ (accessed October 5, 2020).

[22] K. Liu, Token economics #2: comparison review of token economy, Hackernoon, 2019. Available from: https://hackernoon.com/token-economics-2-comparison-review-of-token-economy-8759dd70783 (accessed October 4, 2020).

[23] The Linux Foundation, Hyperledger fabric. Available from: https://www.hyperledger.org/use/fabric (accessed September 27, 2020).

[24] Corda, Open source blockchain platform for business. Available from: https://www.corda.net/ (accessed September 27, 2020).

[25] ConsenSys, ConsenSys quorum. Available from: https://consensys.net/quorum/ (accessed September 27, 2020).

[26] G. Sagirlar, B. Carminati, E. Ferrari, J.D. Sheehan, E. Ragnoli, Hybrid-IoT: hybrid blockchain architecture for internet of things - PoW sub-blockchains, 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), IEEE, Halifax, NS, Canada, 2018, pp. 1007–1016.

[27] Z. Liu, S. Tang, S.S.M. Chow, Z. Liu, Y. Long, Fork-free hybrid consensus with flexible proof-of-activity, Futur. Gener. Comput. Syst. 96 (2019), 515–524.

[28] S. Zhu, Z. Cai, H. Hu, Y. Li, W. Li, zkCrowd: a hybrid blockchain-based crowdsourcing platform, IEEE Trans. Ind. Inform. 16 (2020), 4196–4205.

[29] D. Freuden, Hybrid blockchain: the best of both chains, Hackernoon, 2018. Available from: https://hackernoon.com/hybrid-blockchain-the-best-of-both-chains-78518507449a (accessed September 25, 2020).

[30] G. Wood, Ethereum: a secure decentralised generalised transaction ledger, Ethereum Proj. Yellow Pap., 2014, pp. 1–39.

[31] Ethereum Foundation, ethereum/solidity: solidity, the contract-oriented programming language, Github. Available from: https://github.com/ethereum/solidity (accessed October 5, 2020).

[32] V.P. Ranganthan, R. Dantu, A. Paul, P. Mears, K. Morozov, A decentralized marketplace application on the ethereum blockchain, 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), IEEE, Philadelphia, PA, USA, 2018, pp. 90–97.

[33] Ethereum, "ethereum/web3.js: Ethereum JavaScript API," Github, 2014. Available from: https://github.com/ethereum/web3.js/ (accessed October 4, 2020).

[34] J. Benet, IPFS-content addressed, Versioned, P2P File System, [Online], 2014. Available from: https://arxiv.org/abs/1407.3561 (accessed September 21, 2020).

[35] Q. Zheng, Y. Li, P. Chen, X. Dong, An innovative IPFS-based storage model for blockchain, 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), IEEE, Santiago, Chile, 2018, pp. 704–708.

[36] A. Davis, D. Finlay, MetaMask, ConsenSys, 2016. Available from: https://metamask.io/index.html (accessed October 4, 2020).

[37] B. Darnell, Facebook, B. Taylor, Tornado Web Server, 2009. Available from: https://www.tornadoweb.org/en/stable/ (accessed October 4, 2020).

[38] Hfaran, Tornado-JSON: a simple JSON API framework based on Tornado, Github, 2013. Available from: https://github.com/hfaran/Tornado-JSON (accessed October 4, 2020).

[39] V. Buterin, A next generation smart contract & decentralized application platform, [Online], 2015. Available from: https://ethereum.org/en/whitepaper/ (accessed September 21, 2020).

[40] T. Chen, Z. Li, H. Zhou, J. Chen, X. Luo, X. Li, et al., Towards saving money in using smart contracts, 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER), IEEE, Gothenburg, Sweden, 2018, pp. 81–84.