Distributed Ledger Technology-based Vulnerability Database

by

Huriye Özdemir

Submitted to the Department of Computer
Engineering in partial fulfillment of
the requirements for the degree of
Bachelor of Science

Undergraduate Program in Computer Engineering
Boğaziçi University
Spring 2019

Distributed Ledger Technology-based Vulnerability Database

APPROVED BY:

Prof. Dr. Fatih Alagöz ....
(Project Supervisor)

DATE OF APPROVAL: 01.04.2019

ACKNOWLEDGEMENTS

I would like to thank Prof.Dr.Fatih Alagöz and SATLAB team for their suggestions and comments about the project. I also would like to thank Levent Altay for the support and establishment of the project.

ABSTRACT

Distributed Ledger Technology-based Vulnerability Database

The aim of this project is to create distributed database that enable all users to add vulnerabilities under certain criteria when a vulnerability comes out. The current centralized database which is named National Vulnerability Database (NVD) is insufficient to include all vulnerabilities and store securely providing all security requirements. There are also many vulnerability database platforms such as exploit-db, OSVDB. As a result, Distributed Ledger based vulnerability database will provide more secure, accessible convenient and aggregated database for all interested parties about information security.

ÖZET

Distributed Ledger Technology-based Vulnerability Database

Bu projenin amacı, kullanıcılar tarafından herhangi bir sistemde yeni bir güvenlik açığı keşfedildiğinde, kullanıcıların bu güvenlik açıklarını belirli ölçütler altında sisteme eklemesini sağlayan dağıtılmış bir veritabanı oluşturmaktır. Amerika'daki mevcut merkezi veritabanı Ulusal Güvenlik Açığı Veri Tabanı (NVD), keşfedilen tüm güvenlik açıklarını barındırmak ve güvenlik gereksinimlerini karşılamak konusunda yetersiz kalmaktadır. Exploit-db, OSVDB gibi birçok güvenlik açığı veritabanı platformu vardır. Proje sonucunda dağıtılmış bir yapıda oluşturulacak olan güvenlik açığı veritabanı, bilgi güvenliği ile ilgilenen tüm kullanıcılar ve şirketler için daha güvenli, erişilebilir, kullanışlı ve tek bir yerden erişilebilen toplu bir veritabanı platformu sağlayacaktır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS/ABBREVIATIONS

VDB	Vulnerability Database
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CPE	Common Platform Enumeration
NIST	National Institute of Standards and Technology

1. INTRODUCTION AND MOTIVATION

A vulnerability database is a platform aimed at collecting, maintaining, and disseminating information about discovered vulnerabilities targeting computer systems. Today, the vulnerability databases used to fulfill these objectives are centralized. All of these centralized databases may contain security problems[1] such as SQL injection attacks, mis-configuration of databases, inadequate auditing . They are also lacking in terms of delivering consistent data to users and providing data in a collective and complete manner.

Distributed-ledger technology-based solutions aim at creating a secure structure that will eliminate these security problems and ensure that users can easily access all VDB data and make contributions. This will ensure that all data collected under a single platform is securely accessible.

[1] *https : //en.wikipedia.org/wiki/Vulnerability_database*

2. STATE OF THE ART

A decentralized database for the vulnerabilities has not been created so far. The centralized vulnerability databases used today are created to make public the data related to new vulnerability. Commercial companies have also their own databases. Some of these are as follows:

- Common Vulnerabilities and Exposures (CVE) by MITRE
- National Vulnerability Database (NVD) by NIST
- Common Vulnerability Scoring System (CVSS)
- Common Platform Enumeration (CPE)
- Symantec's DeepSight
- Exploit-DB by Offensive Security
- Microsoft Security Bulletins

These are some of the advantages that differentiate this project from the current databases.

- Decentralization : Eliminating the need for central authority.
- Utilisable : Providing rights for benefits of every user
- Aggregation : Collecting all vulnerabilities on a single platform

3. METHODS

I used Python programming language to implement this project. To implement an interface, I used Flask web framework and created web APIs.

- To perform the mining process, which is necessary for the vulnerabilities to be validated and added to the distributed ledger, I used the Guided Tour Puzzle Protocol, a network bound proof of work mechanism.[2]
- According to the impact score of discovered vulnerability, a score will be added to the user to provide reputation mechanism.

3.1. Distributed Ledger Structure

The distributed ledger Database Structure is based on several nodes on a network where each saves a copy of the ledger and have updated data. Each node can construct new transactions and the nodes validates by the consensus algorithm. When the validation of transaction is completed, the ledger updates itself and all nodes have the collected data. The security is accomplished by strong hashing algorithms and cryptographic keys.
The vulnerability database with distributed ledger structure does not need a peer to peer transactions between two nodes. In this structure, a node will add a new submit to the vulnerability pool. When the nodes mine and validate this submits, a new vulnerability will bi added to the tree according to the category of the vulnerability.

[2] $https://en.wikipedia.org/wiki/Guided_tour_puzzle_protocol$

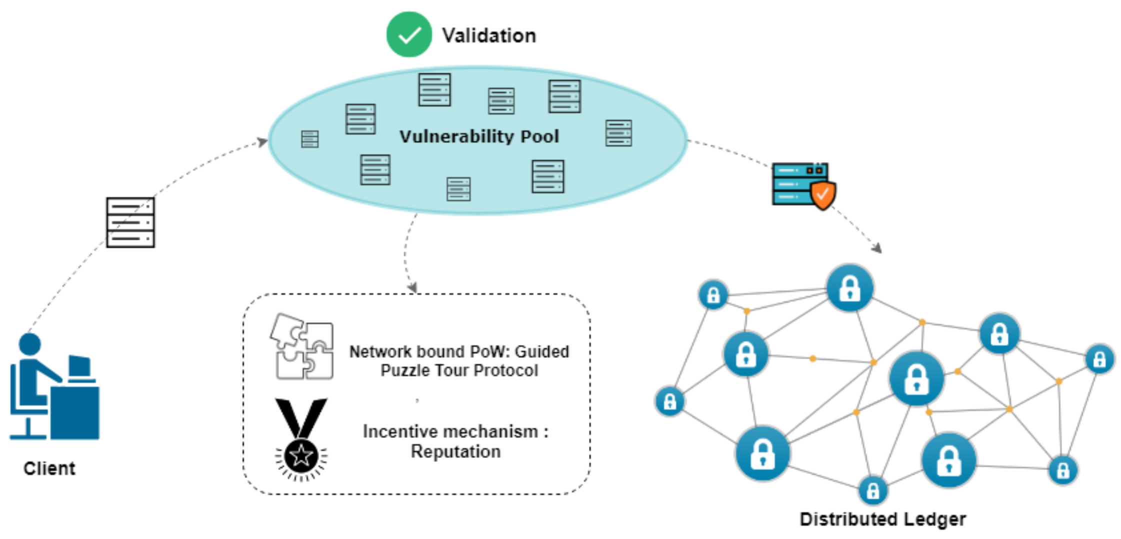

Validation
Vulnerability Pool
Client
Network bound PoW: Guided Puzzle Tour Protocol
Incentive mechanism : Reputation
Distributed Ledger

Figure 3.1. High-level view of the proposed model

3.2. Guided Tour Puzzle Protocol

Guided Tour Puzzle Protocol is a cryptographic protocol that aims to overcome the computation of puzzle that is created by the server. The clients are required to complete multiple roud trips in a sequential order. The important point of this protocol is completing the puzzle in this correct order and find the correct hash pairs that is expected by the server. Tour guides are visited is unknown to the client, and each tour guide has to direct the client towards the next tour guide for the client to complete the tour in correct order. When the client visit each stop it receives a reply that contains a unique token.

3.2.1. Protocol steps

- **Initial puzzle generation :**

  The server replies to the client with a puzzle message that contains computed h_0 and determined the guides that will be visited. The server uses the following formula to compute h_0.

$h_0 = hash(A_x \mid\mid L \mid\mid ts \mid\mid K_s)$

A_x is the client address , L is the length of the tour ,ts is the time stamp and K_s is the secret key.

- **Puzzle solving :**

  A client computes next hashes the following formula:

  $h_l = hash(h_{l-1} \mid\mid l \mid\mid L \mid\mid A_x \mid\mid ts \mid\mid k_{js})$

  h_{l-1} is the previous hash, k_{js} is the shared key between the guides,

- **Puzzle verification :**

  The server and miner computes and solves the puzzle and generate a hash pair to comparison. The server receives a request from client with a puzzle answer $(h'_0\ h'_L)$ and checks the result by comparing with the initial values.

3.3. OWASP Top Ten Classification

I customized the vulnerability database and created a tree that is classified with OWASP Top ten web vulnerabilities. The project aims to provide a distributed ledger for web application vulnerabilities. The client can add a new vulnerability according to these categories. The Open Web Application Security Project (OWASP) is a worldwide not-for-profit charitable organization focused on improving the security of software. The OWASP Top 10 is a document for web application security that contains the most critical security risks to web applications.[3] These are follows:

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities (XXE)

[3] $https : //www.owasp.org/$

- Broken Access Control
- Security Misconfiguration
- Cross-Site Scripting (XSS)
- Insecure Deserialization
- Using Components with Known Vulnerabilities
- Insufficient Logging Monitoring

3.4. Flask

Flask is a micro web framework written in Python. Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.[4]

[4] $https : //en.wikipedia.org/wiki/Flask_(web_framework)$

4. RESULTS

4.1. UML Class Diagram

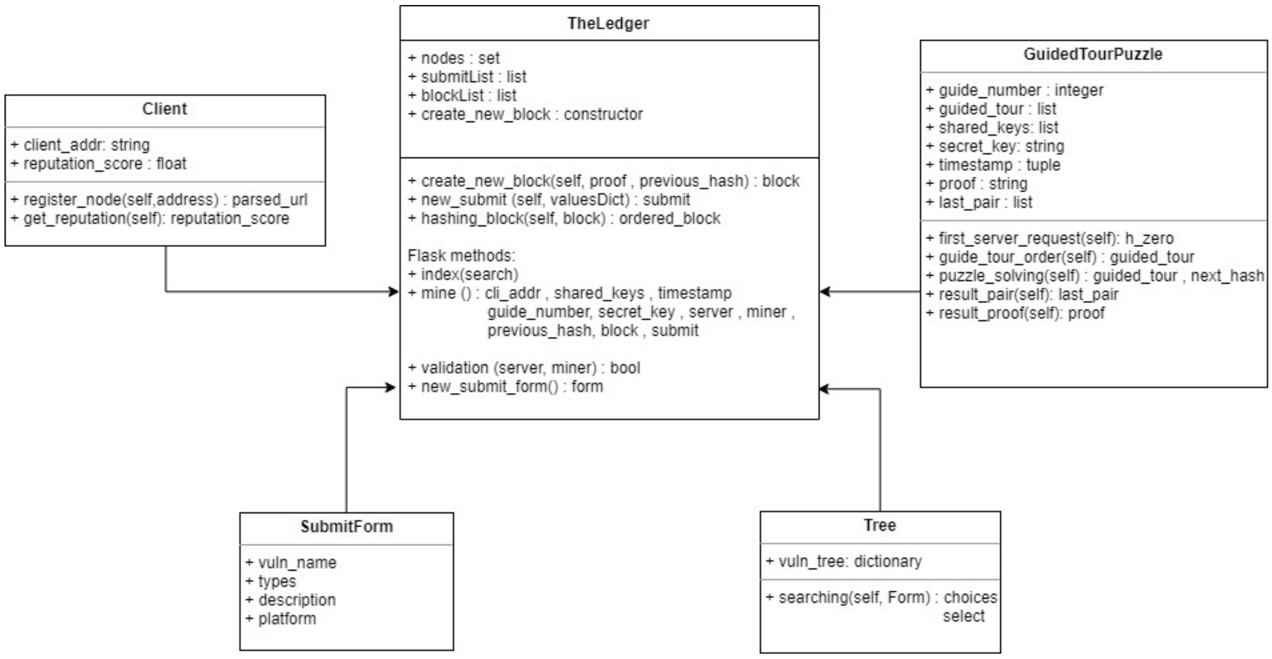
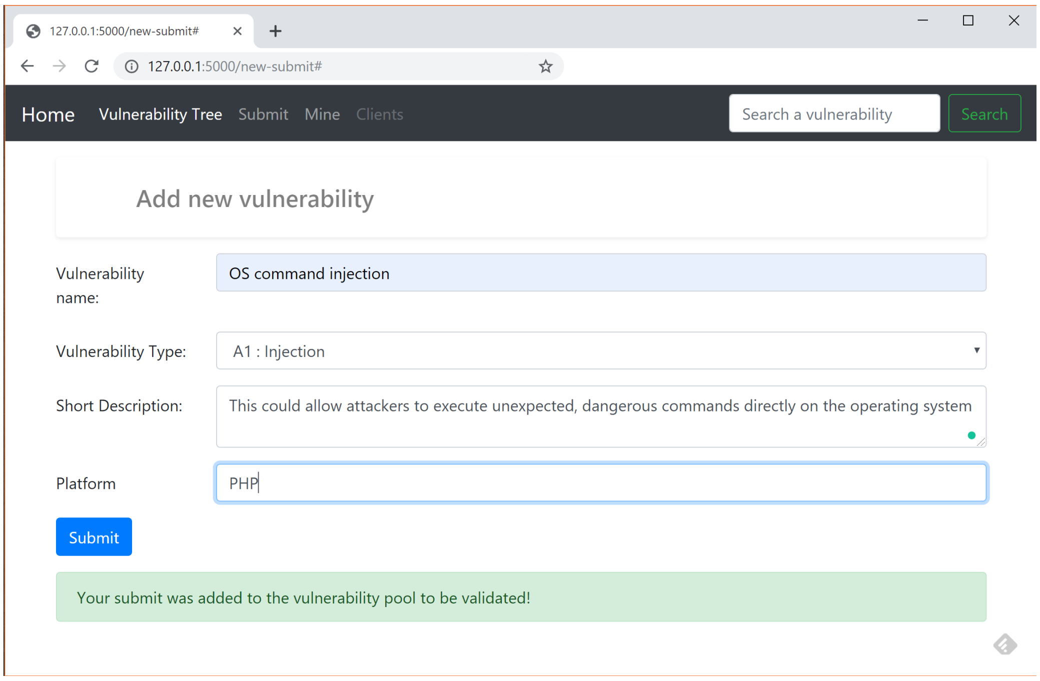

TheLedger
+ nodes : set
+ submitList : list
+ blockList : list
+ create_new_block : constructor
+ create_new_block(self, proof , previous_hash) : block
+ new_submit (self, valuesDict) : submit
+ hashing_block(self, block) : ordered_block
Flask methods:
+ index(search)
+ mine () : cli_addr , shared_keys , timestamp guide_number, secret_key , server , miner , previous_hash, block , submit
+ validation (server, miner) : bool
+ new_submit_form() : form
Client
+ client_addr: string
+ reputation_score : float
+ register_node(self,address) : parsed_url
+ get_reputation(self): reputation_score
GuidedTourPuzzle
+ guide_number : integer
+ guided_tour : list
+ shared_keys: list
+ secret_key: string
+ timestamp : tuple
+ proof : string
+ last_pair : list
+ first_server_request(self): h_zero
+ guide_tour_order(self) : guided_tour
+ puzzle_solving(self) : guided_tour , next_hash
+ result_pair(self): last_pair
+ result_proof(self): proof
SubmitForm
+ vuln_name
+ types
+ description
+ platform
Tree
+ vuln_tree: dictionary
+ searching(self, Form) : choices select

Figure 4.1. UML class diagram

4.2. INTERFACE

First, the client adds a new vulnerability by filling the required data on the submit tab. Until the new mining process starts, new vulnerabilities are added to a single block with an index number.



Add new vulnerability

Vulnerability name: OS command injection

Vulnerability Type: A1 : Injection

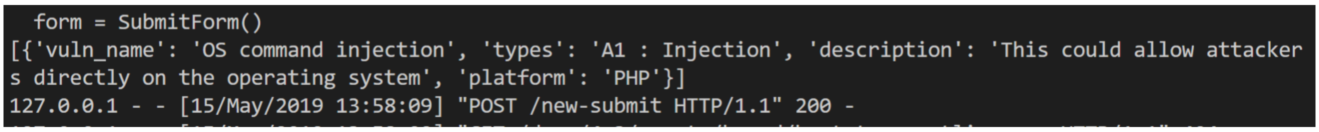
Short Description: This could allow attackers to execute unexpected, dangerous commands directly on the operating system

Platform PHP

Your submit was added to the vulnerability pool to be validated!

Figure 4.2. New submission process

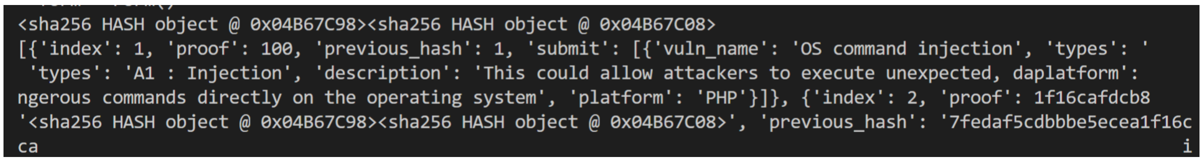
After submission process, the new vulnerability is added to submit list of the distributed ledger.



```
  form = SubmitForm()
[{'vuln_name': 'OS command injection', 'types': 'A1 : Injection', 'description': 'This could allow attacker
s directly on the operating system', 'platform': 'PHP'}]
127.0.0.1 - - [15/May/2019 13:58:09] "POST /new-submit HTTP/1.1" 200 -
```

Figure 4.3. After submission process

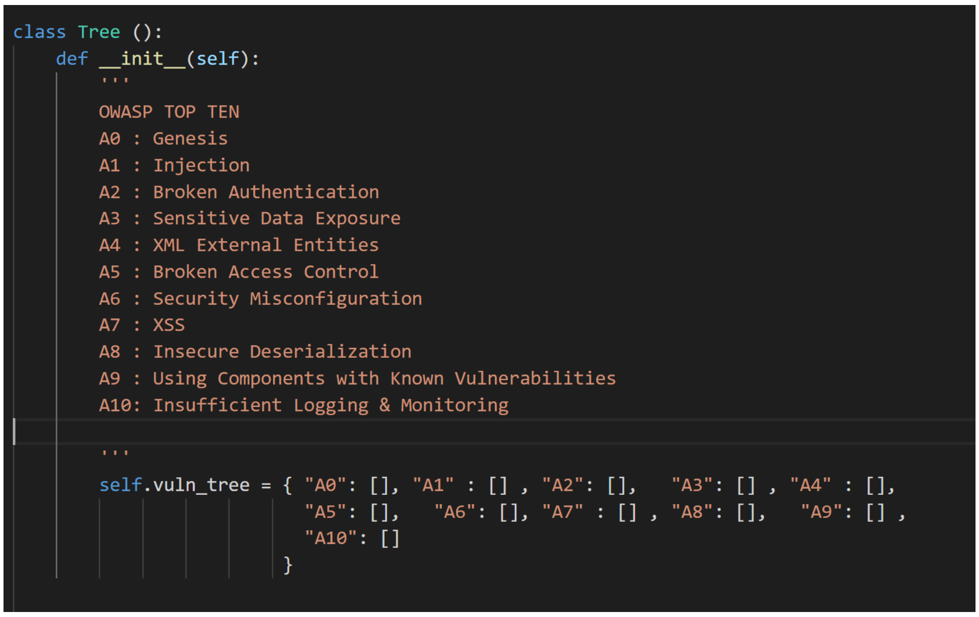
The vulnerabilities in the submit list is added to block to be validated. The block consist of index number, proof value, hash of the previous block and submits.



```
<sha256 HASH object @ 0x04B67C98><sha256 HASH object @ 0x04B67C08>
[{'index': 1, 'proof': 100, 'previous_hash': 1, 'submit': [{'vuln_name': 'OS command injection', 'types': '
 'types': 'A1 : Injection', 'description': 'This could allow attackers to execute unexpected, daplatform':
ngerous commands directly on the operating system', 'platform': 'PHP'}]}, {'index': 2, 'proof': 1f16cafdcb8
'<sha256 HASH object @ 0x04B67C98><sha256 HASH object @ 0x04B67C08>', 'previous_hash': '7fedaf5cdbbbe5ecea1f16c
ca                                                                                                            i
```

Figure 4.4. Created block content

After mining process, the vulnerabilities will be added to tree OWASP Top Ten vulnerability classifications. The tree structure is follows:

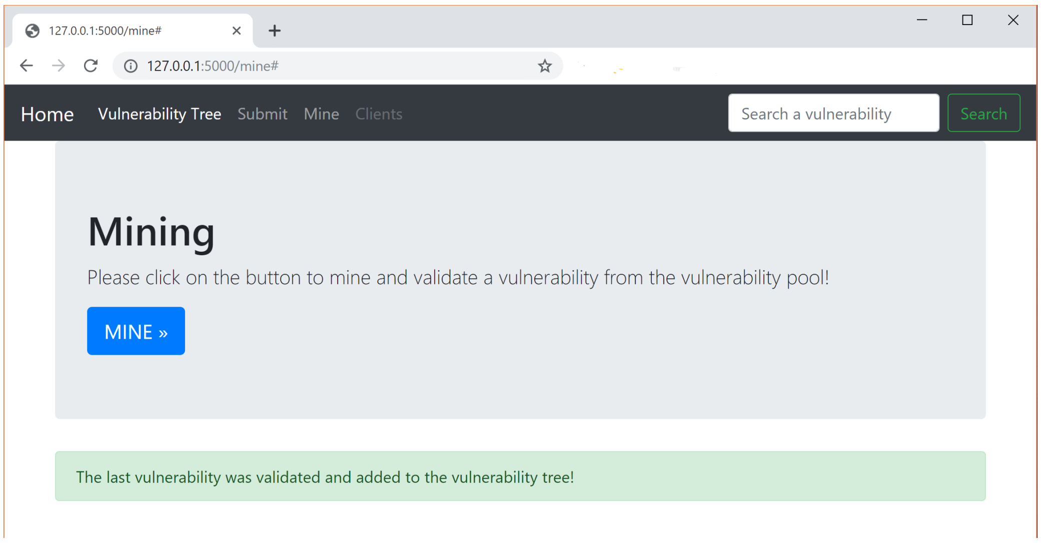


```
class Tree ():
    def __init__(self):
        '''
        OWASP TOP TEN
        A0 : Genesis
        A1 : Injection
        A2 : Broken Authentication
        A3 : Sensitive Data Exposure
        A4 : XML External Entities
        A5 : Broken Access Control
        A6 : Security Misconfiguration
        A7 : XSS
        A8 : Insecure Deserialization
        A9 : Using Components with Known Vulnerabilities
        A10: Insufficient Logging & Monitoring

        '''
        self.vuln_tree = { "A0": [], "A1" : [] , "A2": [],   "A3": [] , "A4" : [],
                           "A5": [],   "A6": [], "A7" : [] , "A8": [],   "A9": [] ,
                           "A10": []
                         }
```

Figure 4.5. Tree structure

When the mining process request comes from the miner, guided tour puzzle protocol computes the proof and try to mine the last block in the block list of the ledger.



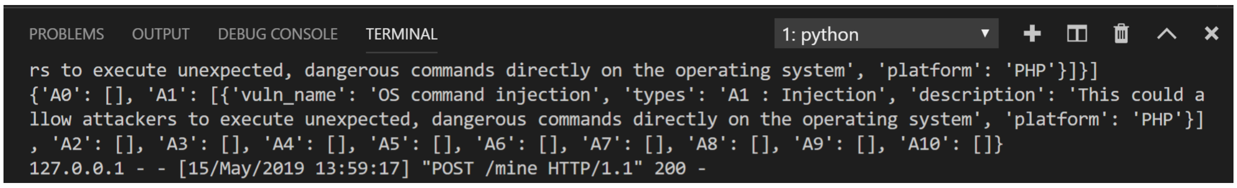
Mining

Please click on the button to mine and validate a vulnerability from the vulnerability pool!

The last vulnerability was validated and added to the vulnerability tree!

Figure 4.6. Mining process

When the block is validated, the new submit added to the tree. The tree structure is as follows:



```
rs to execute unexpected, dangerous commands directly on the operating system', 'platform': 'PHP'}]}]
{'A0': [], 'A1': [{'vuln_name': 'OS command injection', 'types': 'A1 : Injection', 'description': 'This could a
llow attackers to execute unexpected, dangerous commands directly on the operating system', 'platform': 'PHP'}]
, 'A2': [], 'A3': [], 'A4': [], 'A5': [], 'A6': [], 'A7': [], 'A8': [], 'A9': [], 'A10': []}
127.0.0.1 - - [15/May/2019 13:59:17] "POST /mine HTTP/1.1" 200 -
```

Figure 4.7. After mining process

5. CONCLUSION AND DISCUSSION

In this study, I created the distributed ledger based web vulnerability database platform. To eliminate security problems and to provide collective data, this platform provide that users can easily access all VDB data, add a new vulnerability and be also miner in the system.

Guided tour puzzle protocol is used to perform proof of work algorithm. It is based on puzzle solving correctly in a sequential order. The new submits from the client are added the blocks and after mining process, vulnerabilities are validated and added to the tree that is classified according to OWASP top Ten web vulnerability document.

6. FUTURE WORK

This study is only created for web application vulnerabilities. The project can be extended by adding different types of vulnerability to make a more comprehensive database. The new required standards can be added for the submission process of the client. The search mechanism may be faster with strong search algorithms. Rewarding and incentive mechanism can be developed and enhanced for the clients.

REFERENCES

1. https://nvd.nist.gov/

2. https://www.cvedetails.com/

3. https://cve.mitre.org/

4. https://www.exploit-db.com/

5. https://en.wikipedia.org/wiki/Vulnerability_database

6. https://www.owasp.org/

7. https://en.wikipedia.org/wiki/Flask_(web_framework)

8. http://flask.pocoo.org/

9. http://www.wikizero.biz/wiki/en/Guided_tour_puzzle_protocol

10. http://d-scholarship.pitt.edu/24944/1/mehmud_abliz_dissertation.pdf