Predicting cyber security vulnerability severity using boosted machine learning ensembles and feature ranking

[1] Mrs P.Madhavi [2] G.Deepika
[1] CSE, NEC., Gudur
2Assistant Professor,CSE Department, NEC, Gudur

Abstract: In this work we propose Dynamic, a monitoring framework to detect reentrancy vulnerabilities in Ethereum smart contracts. The novelty of our framework is that it relies only on transaction metadata and balance data from the block chain system; our approach requires no domain knowledge, code instrumentation, or special execution environment.

Dynamic extracts features from transaction data and uses a machine learning model to classify transactions as benign or harmful. Therefore, not only can we find the contracts that are vulnerable to reentrancy attacks, but we also get an execution trace that reproduces the attack. Smart contracts are decentralized applications running on Block chain.
Keywords: *component; formatting; style; styling; insert*

INTRODUCTION

The concept of smart contract was first proposed by Nick Szabo in 1990s, defined as "A smart contract is a computerized transaction protocol that executes the terms of contract". Nevertheless, at that time, the exploration only stayed at theoretical level due to the lack of trusted execution environments. Since 2009, with the emergence of block-chain technology that was first applied in Bit coin, a reliable execution environment has been offered to smart contracts.

The remote point-to-point value delivery can be realized without any trusted third party through the integration of many technologies such as distributed data storage, consensus protocols and encryption algorithms.
However, as Bit coin system is not Turing-complete, it is unable to handle complex business logic via smart contracts.

Inspired by Bit coin, Vitalik Buterin developed Ethereum in 2013. Ethereum is an open-source distributed computing platform and operating system based on block chain, which features smart contracts. To meet various demands of business scenarios, Ethereum provides a Turing-complete Ethereum Virtual Machine (EVM) that enables developers to deploy decentralized applications (Dapps) on it. Dapps are also called smart contracts.
They are widely applied in many fields like financial services, infrastructures internet of things, health care, and others. By the end of May 2018, the crypto currency market capital has reached about 35 billion US dollars on Ethereum.
Block chain as a newly developed technology is vulnerable in terms of lacking regulations and programmable characteristics. These vulnerabilities can be easily exploited, resulting huge losses.
This project not only demonstrates the application of TF-IDF vectorization and cosine similarity in building a recommendation engine but also highlights the integration of machine learning techniques with a user-friendly interface to create an engaging and practical tool for movie enthusiasts.

SYSTEM PROPOSAL

This work is licensed under a Creative Commons Attribution 4.0 International License

1 EXISTING SYSTEM:

In existing system, we present Dynamit, a dynamic vulnerability detection framework for Ethereum smart contracts. Dynamit detects vulnerable smart contracts by classifying harmful transactions in a blockchain using machine learningon transactional metadata. We achieve 96 % accuracy on a data set of 105 transactions. To further develop Dynamit, we will investigate automatic test-case generation tools such as Vultron. Another direction for future work is to find more features to make the detection more accurate. An example would be to observe bookkeeping variables inside the contracts, and the way they change, as additional indicators of a smart contract being exploited.

. Finally, we will consider analyzing sequences of multiple transactions and applying other types of machine learning to the data, to increase the capabilities of our detector and to analyze other types of vulnerabilities as well.

DISADVANTAGES:

1. It doesn't efficient for large volume of data's
2. Theoretical limits.
3. The process is implemented without removing the unwanted data.

ADVANTAGES:

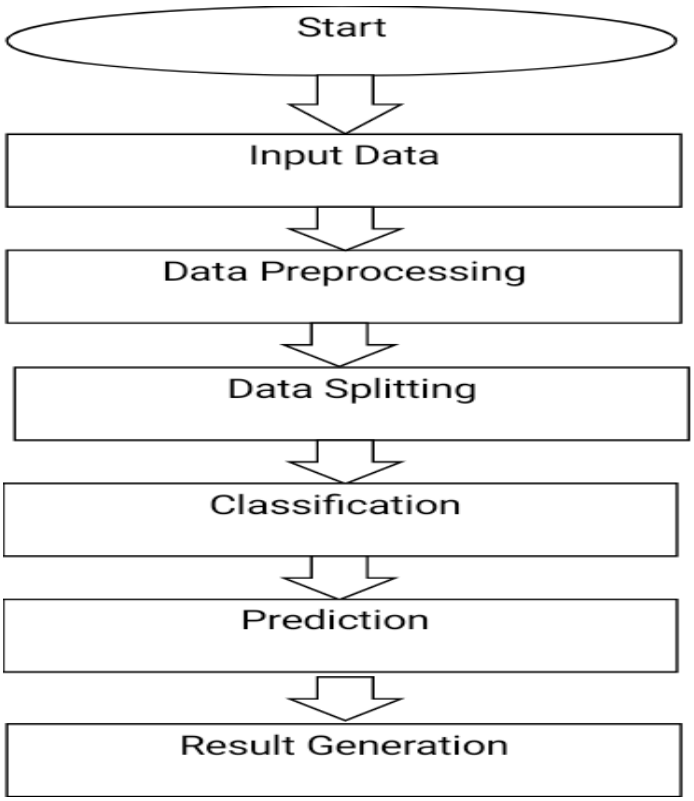
1. It is efficient for large number of datasets.
2. Time consumption is low.
3. The process is implemented with removing the unwanted data.

Here, we have to achieve more than 98% accuracy for above mentioned machine learning algorithms.

This work is licensed under a Creative Commons Attribution 4.0 International License

SYSTEM DIAGRAMS

SYSTEM ARCHITECTURE:



USE CASE DIAGRAM:

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

A use case is a list of actions or event steps typically defining the interactions between a role (known in the Unified Modelling Language (UML) as an actor) and a system to achieve a goal. The actor can be a human or other external system.

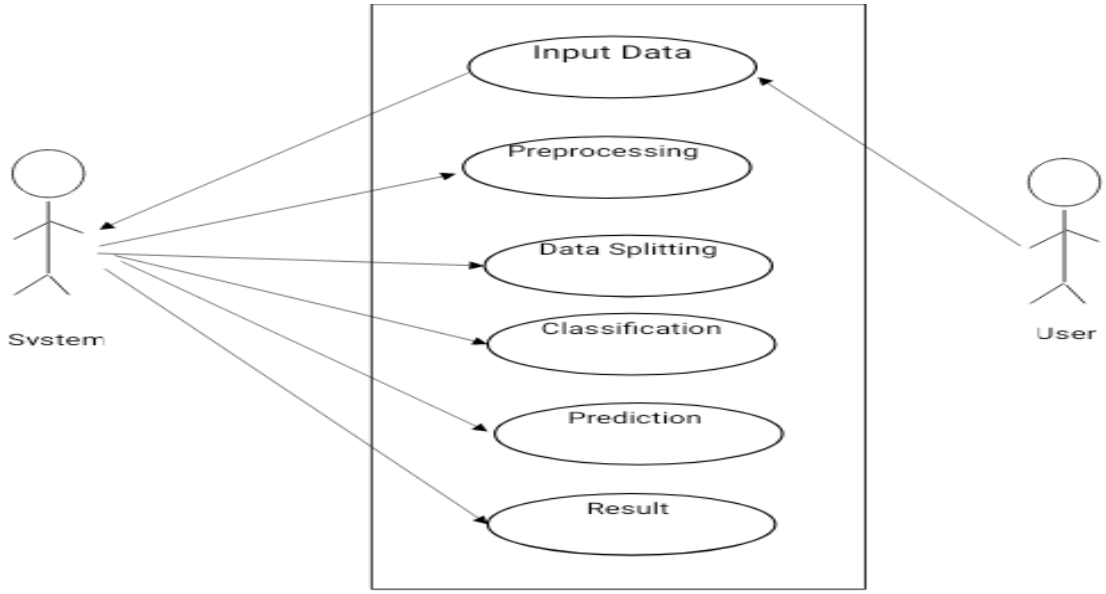
UML use case diagrams are ideal for:

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modelling the basic flow of events in a use case

Notations:

This work is licensed under a Creative Commons Attribution 4.0 International License

- **Use cases**: Horizontally shaped ovals that represent the different uses that a user might have.
- **Actors**: Stick figures that represent the people actually employing the use cases.
- **Associations**: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
- **System boundary boxes**: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
- **Packages**: A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders



SEQUENCE DIAGRAM:

Sequence diagrams document the interactions between classes to achieve a result, such as a use case. Because UML is designed for object-oriented programming, these communications between classes are known as messages. The Sequence diagram lists objects horizontally, and time vertically, and models these messages over time.

Graphical Notation: In a Sequence diagram, classes and actors are listed as columns, with vertical lifelines indicating the lifetime of the object over time.

Object: Objects are instances of classes, and are arranged horizontally. The pictorial representation for an Object is a class (a rectangle) with the name prefixed by the object.

Lifeline The Lifeline identifies the existence of the object over time. The notation 2for a Lifeline is a vertical dotted

This work is licensed under a Creative Commons Attribution 4.0 International License

line extending from an object.

SOFTWARE DESCRIPTION:

Python

- Python is one of those rare languages which can claim to be both *simple* and powerful. You will find yourself pleasantly surprised to see how easy it is to concentrate on the solution to the problem rather than the syntax and structure of the language you are programming in. The official introduction to Python is Python is aneasy to learn, powerful programming language.

TESTING TECHNIQUES/STRATEGIES:

- **WHITE BOX TESTING:**

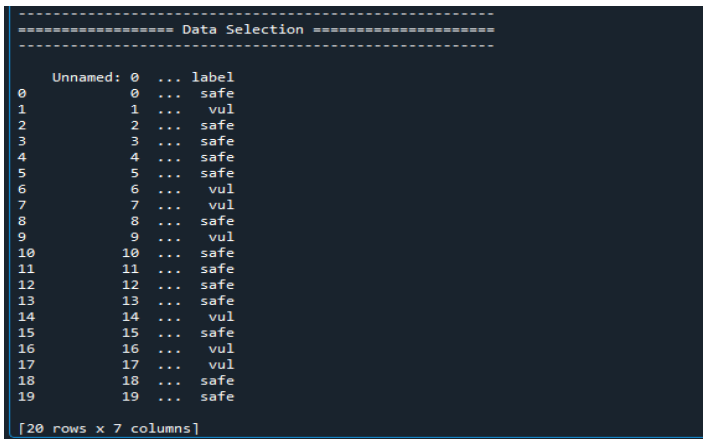
White Box testing is a test case design method that uses the control structure of the procedural design to drive cases. Using the white box testing methods, we Derived test cases that guarantee that all independent paths within a module have been exercised at least once.

- **BLACK BOX TESTING:**

1. Black box testing is done to find incorrect or missing function
2. Interface error
3. Errors in external database access
4. Performance errors.
5. Initialization and termination errors

6 In 'functional testing', is performed to validate an application conforms to its specifications of correctly performs allits required functions. So this testing is also called 'black box testing'.

RESULTS



```
------------------------------------------------------
================== Data Selection ====================
------------------------------------------------------

    Unnamed: 0  ... label
0            0  ...  safe
1            1  ...   vul
2            2  ...  safe
3            3  ...  safe
4            4  ...  safe
5            5  ...  safe
6            6  ...   vul
7            7  ...   vul
8            8  ...  safe
9            9  ...   vul
10          10  ...  safe
11          11  ...  safe
12          12  ...  safe
13          13  ...  safe
14          14  ...   vul
15          15  ...  safe
16          16  ...   vul
17          17  ...   vul
18          18  ...  safe
19          19  ...  safe

[20 rows x 7 columns]
```


This work is licensed under a Creative Commons Attribution 4.0 International License

```
-------------------------------------------------------
============= Before Checking Missing Values ==========
-------------------------------------------------------

Unnamed: 0                0
tx_hash                   0
gas_used                  0
victim_balance_delta      0
attacker_balance_delta    0
call_stack_depth          0
label                     0
dtype: int64
```

REFERENCES

1) [Afzal et al., 2020] Zeeshan Afzal, Anna Brunstrom, Stefan ¨ Lindskog, and Johan Garcia. Using features of encrypted network traffic to detect malware. In 25th Nordic Conference on Secure IT Systems, LNCS, 2020.

2) [Ashizawa et al., 2021] Nami Ashizawa, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura. Eth2Vec: Learning contract-wide code representations for vulnerability detection on ethereum smart contracts. arXiv preprint arXiv:2101.02377, 2021.

3) [Atzei et al., 2017] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on Ethereum smart contracts (SoK). In Matteo Maffei and Mark Ryan, editors, Principles of Security and Trust, pages 164–186, Berlin,Heidelberg, 2017. Springer Berlin Heidelberg.

4) [coinmarketcap.com, 2021] coinmarketcap.com. Cryptocurrency Prices, Charts And Market Capitalizations, 2021.

5) [Dannen, 2017] Chris Dannen. Introducing Ethereum and Solidity, volume 1. Springer, 2017.

This work is licensed under a Creative Commons Attribution 4.0 International License