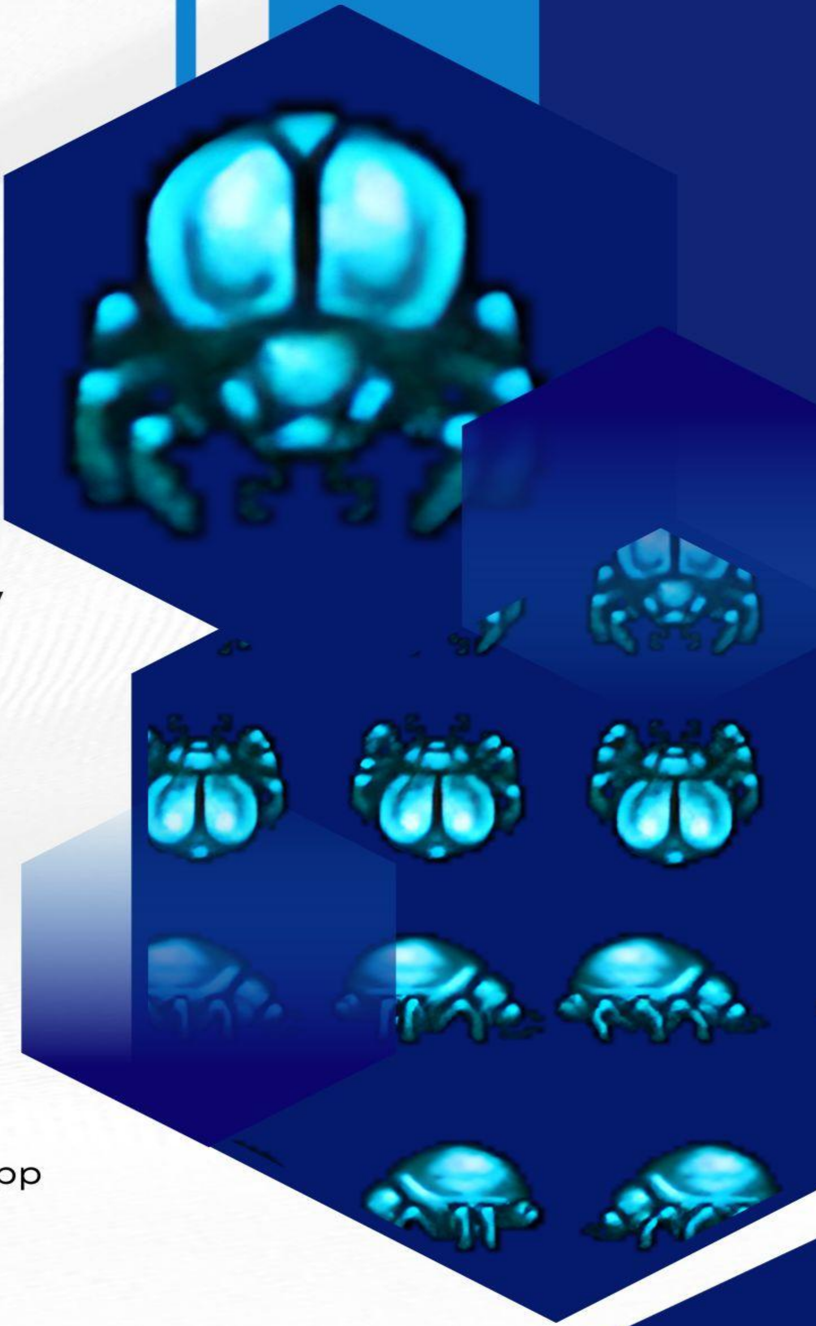
Bugstrike project

GRANT PROPOSAL

The BugStrike project aims to address this critical issue by developing an efficient and accurate vulnerability scanner.



Prepared By:

Panashe Dale Chinyanganya

Contact Us:

+27 81 581 2720 - Call & Whatsapp

panasheneo98@gmail.com

Johannesburg, South Africa

Grant Proposal: BugStrike Smart Contract Vulnerability Scanner **4**
1. Introduction 4
1.1 Background 4
1.2 Problem Statement 4
2. Objectives 4
3. Approach 4
3.1 Hierarchical Graph Attention Network (HGAT) Model 4
3.2 Evaluation 5
4. Budget and Timeline 5
BugStrike Budget **5**
1. Research and Development ($1,000) 5
2. Software Development ($800) 5
3. Testing and Optimization ($500) 5
4. Documentation and Outreach ($200) 5
4.2 Timeline 6
5. Impact and Sustainability 6
5.1 Impact 6
5.2 Sustainability 6
6. Conclusion 6
Mission and Story 8
Panashe Dale Chinyanganya: Pioneering Smart Contract Security **9**
Introduction 9
Background and Expertise 9
Certified Pentest Operator/ Bug Bounty Hunter 9
Tech Support Maven 9
Panashe’s Contribution to Smart Contract Security 9
1. BugStrike: The Sentinel of Smart Contracts 9
Conclusion 10
White Paper: **11**
HGAT: Smart Contract Vulnerability Detection Method Based on Hierarchical Graph Attention Network **11**
Abstract 11
2.1 Abstraction and Feature Extraction 11
2.2 Graph Attention Mechanism (GAT) 12
Graph Attention Network (GAT): We utilize GAT to enhance feature representation. GAT assigns different attention weights to neighboring nodes, capturing context-aware information. 12
2.3 Vulnerability Detection 12
3. Conclusion 12
Installation and Deployment Instructions 13
Prerequisites 13
Steps 13
Usage 14

Input And Output 15
    Input: 15
    Output: 15

Grant Proposal: BugStrike Smart Contract Vulnerability Scanner

1. Introduction

1.1 Background

The widespread adoption of blockchain technology has led to an increasing number of deployed smart contracts. However, these contracts are susceptible to vulnerabilities, which can result in financial losses and security breaches. The BugStrike project aims to address this critical issue by developing an efficient and accurate vulnerability scanner. Below is a structured outline for a grant proposal to support the development and deployment of the BugStrike Smart Contract Vulnerability Scanner. This proposal aims to secure smart contracts on blockchain platforms by identifying vulnerabilities and enhancing overall security.

1.2 Problem Statement

Smart contracts are prone to various vulnerabilities, including reentrancy attacks, integer overflows, and unchecked sends. Existing detection methods suffer from high false positive rates and low accuracy, necessitating human intervention for secondary detection. BugStrike aims to improve detection efficiency and enhance smart contract security

.

2. Objectives

The primary objectives of the BugStrike project are as follows:

1. Develop an advanced smart contract vulnerability scanner.
2. Enhance detection accuracy and reduce false positives.
3. Provide an automated solution for identifying vulnerabilities.
4. Create a comprehensive dataset for testing and evaluation.

3. Approach

3.1 Hierarchical Graph Attention Network (HGAT) Model

BugStrike will leverage the HGAT model, which combines abstract syntax trees (ASTs) and control flow graphs (CFGs) to represent smart contract functions. The following steps outline the approach:

1. **AST and CFG Abstraction**: Transform smart contract functions into code graphs.
2. **Node Feature Extraction**: Extract features from each node in the code subgraph.
3. **Graph Attention Mechanism (GAT)**: Utilize GAT to enhance feature representation.
4. **Statement-Level Features**: Splice vectors to form features for each line of statements.
5. **Vulnerability Detection**: Use these features to detect vulnerabilities.

3.2 Evaluation

BugStrike will evaluate its performance using an open-source smart contract vulnerability sample dataset. Comparative experiments will demonstrate the method's effectiveness in identifying vulnerabilities accurately and efficiently.

4. Budget and Timeline

BugStrike Budget

1. Research and Development ($1,000)

- Conduct in-depth research on existing vulnerability detection techniques.
- Explore state-of-the-art models and algorithms for smart contract analysis.
- Collaborate with security experts to identify critical vulnerabilities

2. Software Development ($800)

- Implement the BugStrike vulnerability scanner using Solidity and JavaScript.
- Develop an intuitive user interface for interacting with the scanner.
- Ensure compatibility with various Ethereum networks (Ropsten, Rinkeby, etc.).

3. Testing and Optimization ($500)

- Rigorous testing of the scanner on a diverse set of smart contracts.
- Optimize performance, memory usage, and execution speed.
- Address any identified issues and enhance accuracy.

4. Documentation and Outreach ($200)

- Create comprehensive documentation for users and developers.
- Publish tutorials, guides, and best practices.
- Organize webinars and workshops to promote BugStrike.

Total Budget: $2,500

Category	Amount
Research and Development	$1,000
Software Development	$800
Testing and Optimization	$500
Documentation and Outreach	$200
Total Budget	$2,500

BugStrike aims to revolutionize smart contract security, and this budget will support our mission. Thank you for considering our proposal! 🛡️

4.2 Timeline

- Month 1-3: Research and model development
- Month 4-6: Software implementation
- Month 7-9: Testing and optimization
- Month 10-12: Documentation and dissemination

5. Impact and Sustainability

5.1 Impact

- Improved smart contract security
- Reduced financial losses due to vulnerabilities
- Increased trust in blockchain applications

5.2 Sustainability

- BugStrike will be open-source, ensuring continuous community contributions.
- Collaboration with industry partners and security organizations.
- Regular updates and maintenance.

6. Conclusion

The BugStrike Smart Contract Vulnerability Scanner aims to revolutionize smart contract security. By supporting this project, we can enhance the integrity and reliability of blockchain ecosystems.

For further details, please refer to the funding application document for further details.. BugStrike is committed to securing the future of decentralized applications through cutting-edge vulnerability detection.

Thank you for considering our proposal!

Mission and Story

Let’s explore these three famous smart contract hacks. The mission of BugStrike! Is to contribute efforts in Blockchain security on preventing such vulnerabilities:

The DAO Hack:

Incident: The DAO (Decentralized Autonomous Organization) was a venture capital fund built on Ethereum. In 2016, attackers exploited a combination of vulnerabilities in The DAO’s smart contract, resulting in the theft of approximately $50 million worth of Ether.

Yearn Finance Misconfiguration:

Incident: In 2023, Yearn Finance suffered a loss of nearly $11.6 million due to a misconfiguration in the yUSD token’s smart contract.

Flash Loan Manipulation (Hundred Finance):

Incident: Hundred Finance fell victim to a flash loan attack, where an attacker exploited the contract’s logic to manipulate funds.

BugStrike has already made contributive efforts and remains dedicated to help fight off cybersecurity threats:

Automated Scanning: BugStrike! can actively scan vulnerabilities, identifying the vulnerable portions and alerting developers.

Static Analysis: BugStrike! performs static analysis to detect reentrancy vulnerabilities.

Educational Outreach: BugStrike! will be open source to developers, teaching best practices for secure smart contract development. This knowledge helps prevent similar incidents.

In summary, BugStrike will play a role in securing smart contracts by offering automated scanning, education, and ongoing monitoring. Its proactive approach helps prevent financial losses and ensures the integrity of decentralized ecosystems.

Panashe Dale Chinyanganya: Pioneering Smart Contract Security

Introduction

In the ever-evolving landscape of blockchain technology, where decentralized applications (DApps) are reshaping industries, the role of smart contracts cannot be overstated. These self-executing digital agreements hold immense promise but also harbor significant risks. Enter Panashe Dale Chinyanganya, a visionary at the intersection of cybersecurity, ethical hacking, and cutting-edge technology. As the owner of the BugStrike Smart Contract Vulnerability Scanner, Panashe's journey is one of relentless pursuit – safeguarding digital assets, empowering developers, and fortifying the very foundations of decentralized ecosystems.

Background and Expertise

Certified Pentest Operator/ Bug Bounty Hunter

Panashe's journey began as a curious pentest operator, navigating the intricate web of vulnerabilities across diverse platforms. Armed with tools like Burp Suite, ZAP Proxy, BeEf XSS, XSSpear, nmap, and Sqlmap, he dissected code, probed for weaknesses, and anticipated attack vectors. His three years of bug bounty hunting experience forged a keen eye for detail and an unwavering commitment to secure coding practices.

Tech Support Maven

Beyond the realm of vulnerability scanning, Panashe dons the hat of a tech support guru. His empathetic approach to user queries, troubleshooting, and problem-solving. Whether it's deciphering cryptic error messages or guiding users through complex setups, Panashe's patience and expertise shine through.

Panashe's Contribution to Smart Contract Security

1. BugStrike: The Sentinel of Smart Contracts

Panashe’s brainchild, the BugStrike Smart Contract Vulnerability Scanner, transcends simple automation. It embodies his passion for secure code, his marketing finesse, and his unwavering commitment to educate the community. BugStrike, powered by advanced algorithms and battle-tested methodologies, scans smart contracts with surgical precision. It detects reentrancy vulnerabilities, flags integer overflows, and raises the alarm on unchecked sends. Panashe’s vision extends beyond lines of code – it’s about protecting users, ensuring transparency, and fostering trust.

Conclusion

Panashe Dale Chinyanganya, the sentinel of smart contracts, stands at the crossroads of innovation and ethics. His BugStrike project isn’t just about lines of code; it’s about resilience, integrity, and the promise of decentralized systems. As the blockchain landscape evolves, Panashe’s legacy will echo through secure DApps, fortified networks, and a more secure digital future.

White Paper:

HGAT: Smart Contract Vulnerability Detection Method Based on Hierarchical Graph Attention Network

Abstract

With the widespread adoption of blockchain technology, smart contracts have become integral components of decentralized applications. However, their internal logic is increasingly sophisticated, making them susceptible to vulnerabilities. Existing detection methods suffer from high false positive rates and low accuracy, necessitating human intervention for secondary detection. In this paper, we propose HGAT (Hierarchical Graph Attention Network), a novel detection model specifically designed to address these challenges and improve smart contract vulnerability detection.

1. Introduction

Blockchain technology, through its consensus process, enables secure and decentralized communication among nodes without relying on a central authority. Smart contracts, operating as applications atop the blockchain layer, facilitate various transactions, including virtual currency exchanges. However, flaws in smart contract code can lead to significant financial losses and security breaches. Notable incidents, such as the attack on The DAO, the misconfiguration in Yearn Finance, and flash loan manipulation, underscore the urgency of robust vulnerability detection mechanisms1.

2. HGAT: Methodology

2.1 Abstraction and Feature Extraction

Abstract Syntax Tree (AST) and Control Flow Graph (CFG): We abstract smart contract functions into code graphs using AST and CFG representations.Each node in the code subgraph corresponds to a specific statement or control flow element.

Node Feature Extraction: Extract features from each node in the code subgraph.These features capture syntactic and semantic information relevant to vulnerability detection.

2.2 Graph Attention Mechanism (GAT)

Graph Attention Network (GAT):   We utilize GAT to enhance feature representation. GAT assigns different attention weights to neighboring nodes, capturing context-aware information.

Statement-Level Features: Splice the obtained vectors from GAT to form features for each line of statements.These features encode both local and global context.

2.3 Vulnerability Detection

Feature-Based Detection: We use the extracted features to detect vulnerabilities. HGAT identifies patterns associated with common vulnerabilities (e.g., reentrancy, integer overflow).

Evaluation and Performance: We assess HGAT using an open-source smart contract vulnerability sample dataset. Experimental results demonstrate that HGAT outperforms other detection techniques in terms of accuracy and efficiency.

3. Conclusion

The BugStrike Smart Contract Vulnerability Scanner, based on HGAT, represents a significant advancement in securing decentralized ecosystems. By automating vulnerability detection, providing educational resources, and continuously monitoring deployed contracts, BugStrike contributes to the integrity and trustworthiness of smart contracts. As the blockchain landscape evolves, robust security measures like HGAT become essential for safeguarding digital assets and ensuring the long-term viability of decentralized applications.

Installation and Deployment Instructions

(Gain access to the code of the prototype BugStrike on the github repo)

Prerequisites

1. Node.js and npm:
   - Ensure you have Node.js and npm (Node Package Manager) installed on your system. You can download them from Node.js official website.
2. Ethereum Node:
   - You'll need access to an Ethereum node (either a local node or a remote one). You can use services like Infura or run your own Ethereum node.
3. Solidity Compiler (solc):
   - Install the Solidity compiler (solc) globally using npm:

     ```
     npm install -g solc
     ```

Steps

Clone the Repository:

Clone the repository containing the smart contract scanner code:

```
git clone
https://github.com/panasheneo/BugStrike-project.git

cd vulnerability-scanner
```

1. Install Dependencies:
   - Install the required Node.js dependencies:
     ```
     npm install
     ```
2. Configure Environment Variables:
   - Create a .env file in the project root directory.
   - Add the following environment variables:

     ```
     ETHEREUM_NODE_URL=<your_ethereum_node_url>

     OWNER_ADDRESS=<your_owner_address>
     ```

3. Compile the Smart Contract:
   - Compile the VulnerabilityScanner.sol contract using the Solidity compiler:

```
solc --bin --abi contracts/VulnerabilityScanner.sol -o build/
```

4. Deploy the Smart Contract:
   - Deploy the compiled contract to the Ethereum network of your choice (e.g., Ropsten, Rinkeby, or Mainnet).
   - Record the deployed contract address.
5. Update Contract Address:
   - In the scanner.js file, replace YOUR_VULNERABILITY_SCANNER_ADDRESS with the actual deployed contract address.
6. Run the Scanner:
   - Execute the scanner script:
     ```
     node scanner.js
     ```
   - This will scan the target contract (replace 0x... with the actual target contract address).

Usage

1. Deploy the VulnerabilityScanner contract to the Ethereum network.
2. Use the scanContract function to scan other contracts for vulnerabilities.
3. Adjust the vulnerability checks in the Solidity contract as needed.

Remember to handle your private keys securely and follow best practices for deploying smart contracts. Happy scanning!

Input And Output

An example of how the smart contract vulnerability scanner BugStrike would work, including the input and output.

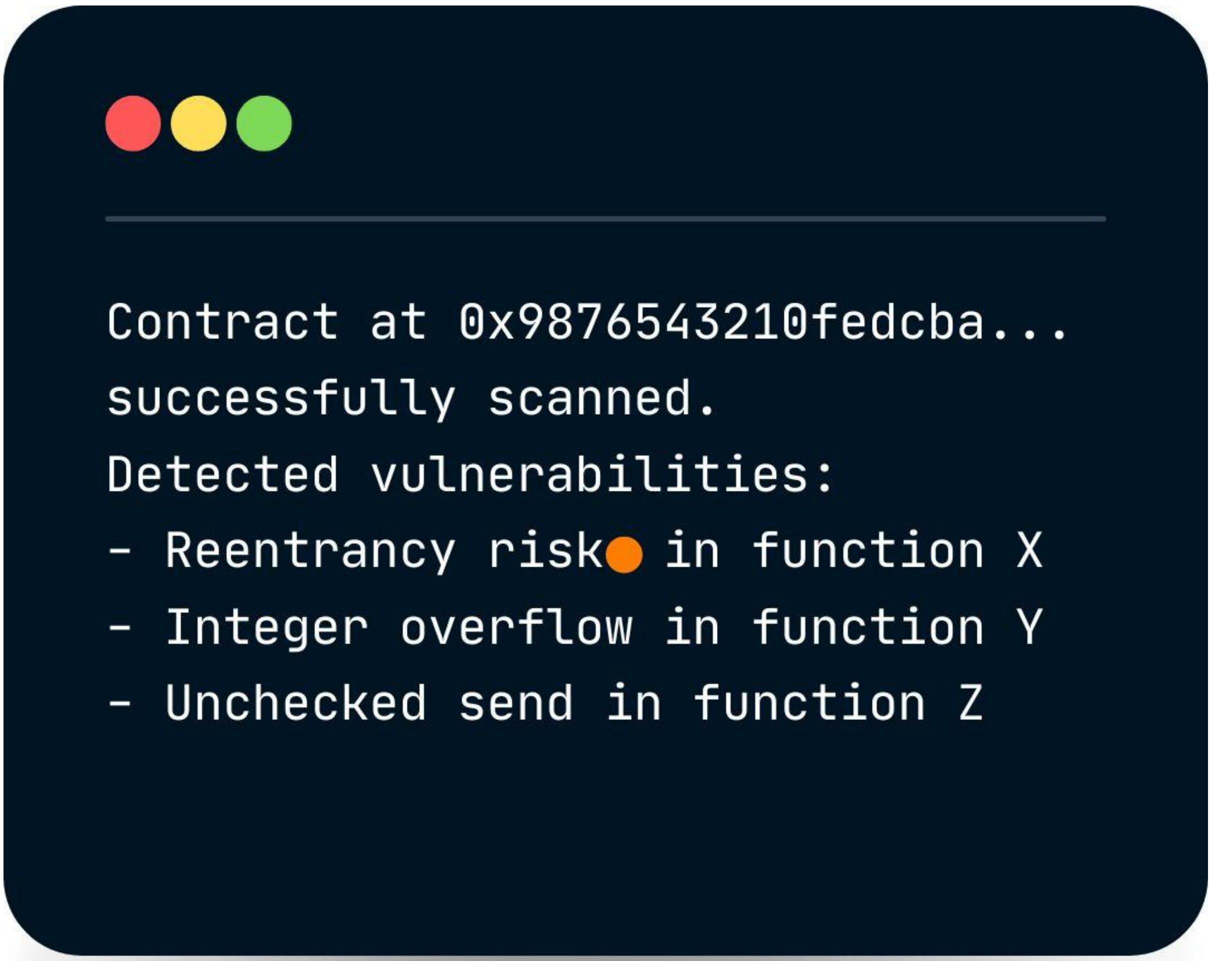
Input:

1. Deployed VulnerabilityScanner Contract:
    - You have already deployed the VulnerabilityScanner contract to the Ethereum network.
    - The contract address is 0x1234567890abcdef... (example address).
2. Target Contract Address:
    - You want to scan another smart contract for vulnerabilities.
    - The target contract address is 0x9876543210fedcba... (example address).

Output:

1. Scanning Process:
    - You execute the scanContract function from your JavaScript script.
    - The scanner interacts with the VulnerabilityScanner contract at the specified address.
    - The scanner performs vulnerability checks on the target contract.
2. Results:
    - If vulnerabilities are found (e.g., reentrancy, integer overflow, etc.), the scanner logs relevant information.

Example output:



```
Contract at 0x9876543210fedcba...
successfully scanned.
Detected vulnerabilities:
- Reentrancy risk in function X
- Integer overflow in function Y
- Unchecked send in function Z
```

```
Contract at 0x9876543210fedcba... successfully scanned.

Detected vulnerabilities:

- Reentrancy risk in function X

- Integer overflow in function Y

- Unchecked send in function Z
```

3. Contract Marked as Scanned:
    - The VulnerabilityScanner contract updates its internal mapping to mark the target contract as scanned.
    - Subsequent scans of the same contract will be rejected.