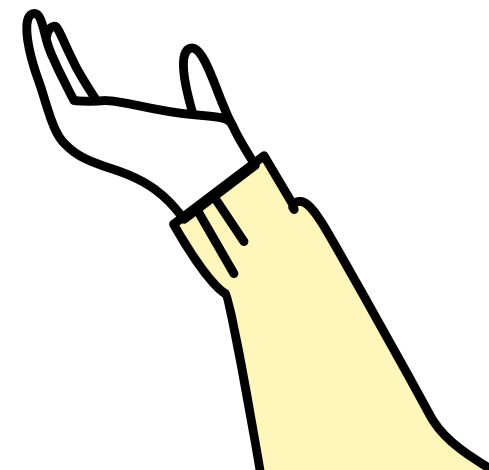
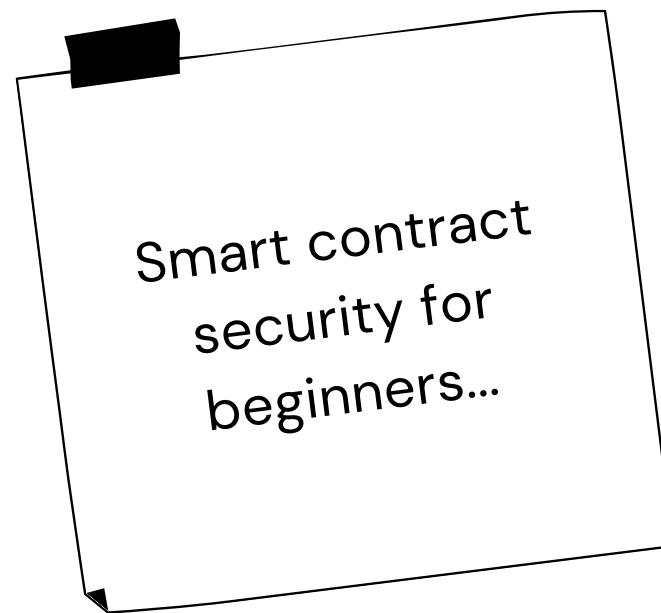


Smart Contract Security



Today's Agenda

1

Introduction to the Smart
Contract Security

2

Secure coding practices

3

Vulnerabilities To Watch Out For

4

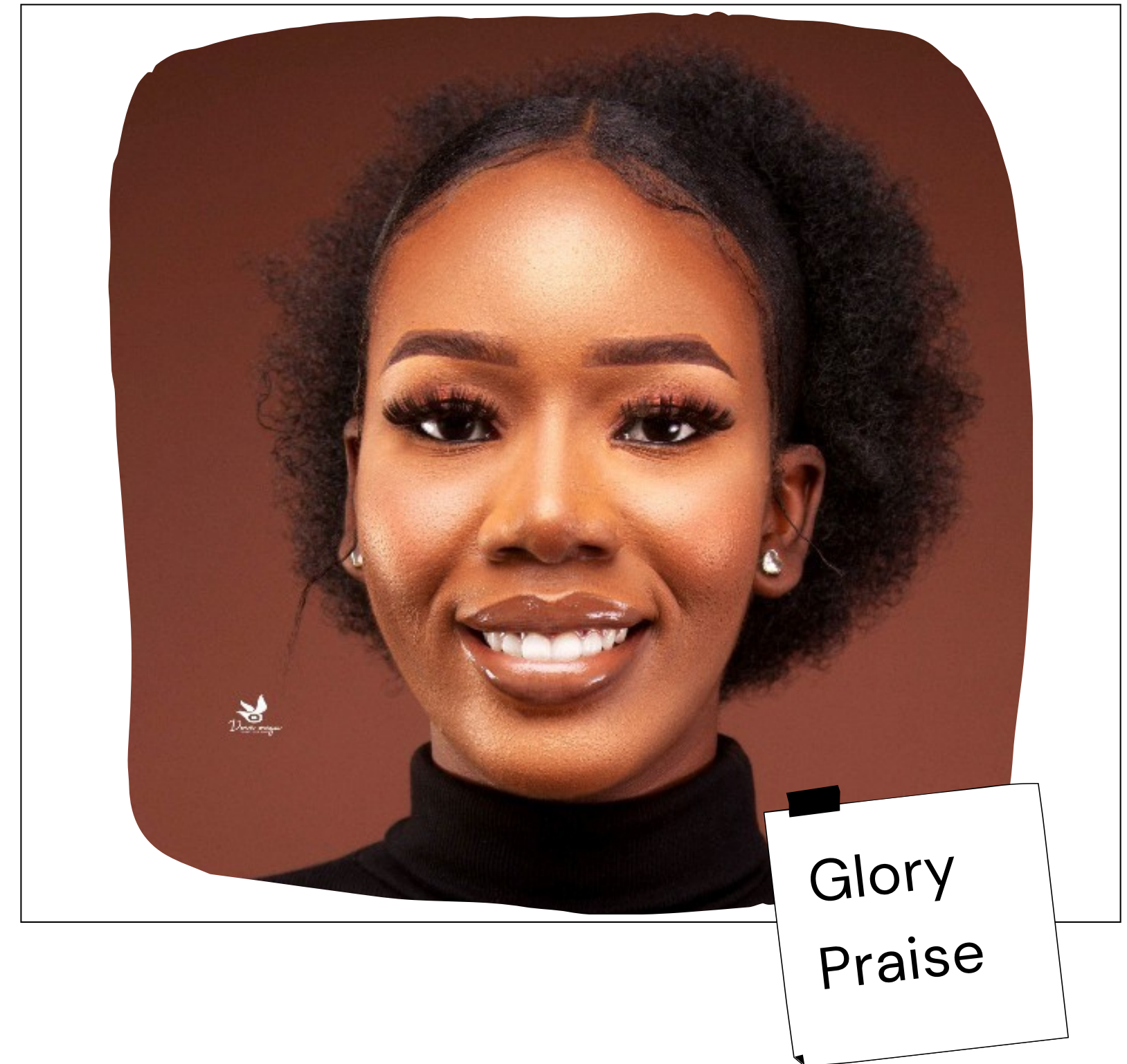
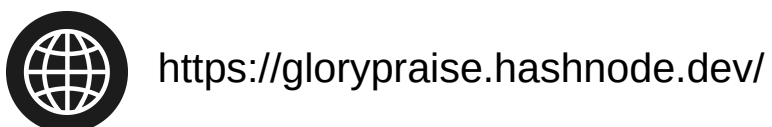
Security Testing

Who is talking?

Glory Praise Emmanuel is a Blockchain Developer/ Smart Contract Developer, and Technical Writer with experience developing high-performance solutions.

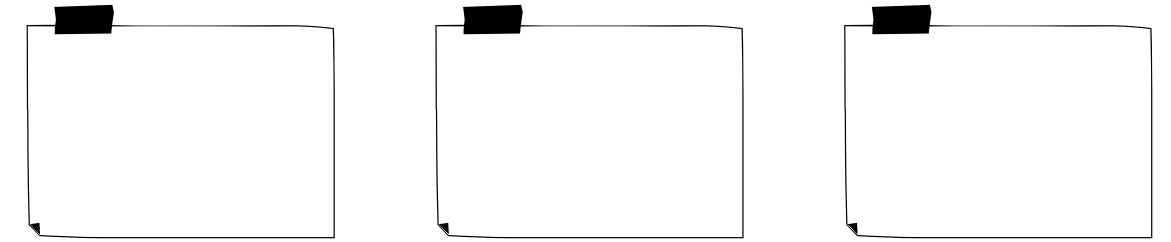
She is passionate about creating tech products and has experience writing performant smart contracts, and developing websites, web apps, and decentralized applications (DApps).

She is deeply committed to educating people about technology and assisting developers in becoming better at what they do.



Icebreaker

Before we start with the session,
let's warm up a little with this icebreaker questions:



What's your frequently used Emoji?

Below are suggestions for icebreaker questions:

- What is a smart contract?
- Have you ever lost money in crypto, eg wallet hack, etc?
- What's the most challenging thing you've done since you started coding?
- What's your favorite programming language?



Let's get started!



Introduction to the Smart Contract Security

Smart contracts are self-executing contracts with the terms of the agreement written directly into code.

Smart contract security refers to the measures taken to protect the integrity and confidentiality of smart contracts on a blockchain network. These measures include identifying and mitigating potential vulnerabilities, such as bugs and programming errors, as well as implementing security best practices during the development and deployment of smart contracts.

Some of the risk areas associated with smart contracts are:

- The potential for unintended consequences due to bugs or programming errors.
- Access control.
- Gas optimization
- Decentralized Oraclization
- Upgradability



What have you learnt
so far?



**Any
Question???**

What would you like
to clarify?

Secure coding practices

1. Design proper access controls:

Giving every network participant access to contract functions can cause problems, especially if it means anyone can perform sensitive operations (e.g., minting new tokens). Access control is an important aspect of smart contract security as it ensures that only authorized individuals or groups can perform certain actions within a smart contract. This can include setting up a whitelist of approved addresses, or implementing multi-sig (multisignature) transactions, which require multiple parties to sign off on a transaction before it can be executed.

Another way to implement access control in smart contracts is through the use of roles. Roles are predefined sets of permissions that can be assigned to specific addresses. For example, an administrator role may be given the ability to modify the contract's state, while a regular user role may only be able to make transfers.



Code Example

```
constructor(){
owner = msg.sender;
}

modifier onlyOwner() {
require(msg.sender == owner, "Not owner");
_;
}

function createGrant(address _beneficiary, uint _time) external payable onlyOwner returns(uint){
require(msg.value > 0, "Zero ether not allowed");
BP.ammountAllocated = msg.value;
BP.time = _time;

return _id;
}
```

Secure coding practices

2. Ask for a review of your code

An independent review is a process in which a third-party developer or team of developers review your contract's code for potential vulnerabilities or bugs. This type of review is important for several reasons:

- It helps to identify potential security issues that may have been overlooked during the development process.
- An independent review provides a fresh perspective on the code.
- It helps to build trust and credibility for the contract. By having an independent review, the contract's users can have confidence that the contract has been thoroughly reviewed and is secure.

Ways of getting review of your code includes bug bounties, auditing, doing peer code reviews etc.



Secure coding practices

3. Use `require()`, `assert()`, and `revert()` statements to guard contract operations:

It is important to use `require()`, `assert()`, and `revert()` statements to guard contract operations in order to ensure the security of the contract. These statements are used to perform checks on the input and state of the contract before executing certain operations. They help to prevent potential security issues such as reentrancy attacks, unauthorized access, and unexpected behavior of the contract.

- `require` is used to validate inputs and conditions before execution.
- `assert` is used to check for code that should never be false. Failing assertion probably means that there is a bug.
- `revert()` is used abort execution and revert state changes

Code Example

```
require(!lock, "Cannot withdraw while another withdrawal  
is processing");
```

```
assert(myAmount < __check);
```

Secure coding practices

4. Test smart contracts and verify code correctness:

Testing and verifying the code correctness of smart contracts are important steps in ensuring the security of the contracts.

- **Testing:** Testing is the process of executing the contract code in a controlled environment to ensure that it behaves as expected. It helps to identify bugs and vulnerabilities in the code before the contract is deployed on the blockchain network. There are different types of tests that can be performed on a smart contract, including unit tests, integration tests, and end-to-end tests. Unit tests focus on testing individual functions or modules of the contract, integration tests focus on testing the interaction between different parts of the contract, and end-to-end tests focus on testing the contract as a whole.
- **Code Verification:** Code verification is the process of reviewing and analyzing the contract's code to ensure that it is correct and secure. It involves a detailed analysis of the contract's logic, function calls, and data storage. It also includes a review of the contract's architecture, design patterns, and coding standards. This process is usually done by an independent team or person who has experience and expertise in smart contract security.



Secure coding practices

5. Implement robust disaster recovery plans:

Implementing robust disaster recovery plans with respect to smart contract security is an important aspect of ensuring the security and reliability of smart contracts.

- **Contract upgrades:** Smart contracts are software programs, and as such, they may require updates or upgrades from time to time. These upgrades may be necessary to fix bugs, add new features, or address security vulnerabilities. A disaster recovery plan should include a procedure for upgrading the contract in a controlled and tested manner.
- **Emergency stops:** An emergency stop is a mechanism that allows the contract to be halted or paused in the event of an unexpected failure or security issue. This can be useful in situations where a critical vulnerability is discovered or where the contract is behaving in an unexpected manner. A disaster recovery plan should include a procedure for triggering an emergency stop and a process for restoring the contract to normal operation once the issue has been resolved.
- **Event monitoring:** Smart contracts execute on a blockchain network and generate a large number of events and transactions. Event monitoring is the process of tracking and analyzing these events to detect potential security issues or unexpected behavior. A disaster recovery plan should include a procedure for setting up event monitoring systems and for responding to any issues that are detected.



What have you learnt
so far?



**Any
Question???**

What would you like
to clarify?

Vulnerabilities To Watch Out For

Some vulnerabilities to watch out for in smart contracts include:

- Reentrancy: This vulnerability occurs when a contract allows an attacker to repeatedly call a function, potentially draining the contract's balance.
- Unchecked-send: This vulnerability occurs when a contract sends Ether to an untrusted address without checking if the address is valid.
- Timestamp Dependence: This vulnerability occurs when a contract relies on the timestamp provided by the blockchain, which can be manipulated by an attacker.
- Integer Overflow/Underflow: This vulnerability occurs when a contract performs arithmetic operations that result in a value that is outside the range of the data type used to represent it.
- Uninitialized Storage Pointers: This vulnerability occurs when a contract uses a storage pointer that has not been initialized, potentially revealing sensitive information.
- Lack of Exception handling: This vulnerability occurs when a contract does not have proper exception handling mechanisms in place, which can lead to unexpected behavior and security issues.
- Unrestricted access: This vulnerability occurs when a contract does not have proper access control mechanisms in place, allowing unauthorized parties to access and manipulate data.

Visit my blog for more vulnerabilities: <https://glorypraise.hashnode.dev/>

What have you learnt
so far?



**Any
Question???**

What would you like
to clarify?

Security Testing

Smart contract testing means performing detailed analysis and evaluation of a smart contract to assess the quality of its source code during the development cycle. Testing a smart contract makes it easier to identify bugs and vulnerabilities and reduces the possibility of software errors that could lead to costly exploits. Strategies for testing smart contracts can be classified into two broad categories: automated testing and manual testing.

- Automated testing: Automated testing involves using automated tools to carry out scripted testing of smart contracts. This technique relies on automated software that can execute repeated tests to find defects in smart contracts. Automated testing is efficient, uses fewer resources, and promises higher levels of coverage than manual analysis. Automated testing tools can also be configured with test data, allowing them to compare predicted behaviors with actual results.
- Manual testing: Manual testing is human-aided and involves an individual who executes testing steps manually. Code audits, where developers and/or auditors, go over every line of contract code, are an example of manual testing for smart contracts. Manual testing of smart contracts requires considerable skill and a considerable investment of time, money, and effort. Moreover, manual testing can sometimes be susceptible to the problems of human error.



Security Testing

AUTOMATED TESTING FOR SMART CONTRACTS

- **Functional testing:** Functional testing verifies the functionality of a smart contract and provides assurance that each function in the code works as expected. Functional testing requires understanding how your smart contract should behave in certain conditions. Then you can test each function by running computations with selected values and comparing the returned output with the expected output. Functional testing covers three methods: **unit testing**, **integration testing**, and **system testing**.
- **Static/dynamic analysis:** Static analysis and dynamic analysis are two automated testing methods for evaluating the security qualities of smart contracts. Static analysis examines the source code or bytecode of a smart contract before execution. This means you can debug contract code without actually running the program. Dynamic analysis techniques require executing the smart contract in a runtime environment to identify issues in your code. Fuzzing is an example of a dynamic analysis technique for testing contracts. During fuzz testing, a fuzzer feeds your smart contract with malformed and invalid data and monitors how the contract responds to those inputs.



Security Testing

MANUAL TESTING FOR SMART CONTRACTS

- Code audits: A code audit is a detailed evaluation of a smart contract's source code to uncover possible failure-points, security flaws, and poor development practices. While code audits can be automated, we refer to human-aided code analysis here. Code audits require an attacker mindset to map out possible attack vectors in smart contracts. Even if you run automated audits, analyzing every line of source code is a minimum requirement for writing secure smart contracts.
- **Bug bounties:** A bug bounty is a financial reward given to an individual who discovers a vulnerability or bug in a program's code and reports it to developers. Bug bounties are similar to audits since it involves asking others to help find defects in smart contracts. The major difference is that bug bounty programs are open to the wider developer/hacker community.
- Examples of bug bounties platforms: are code4arena, immunefi



What have you learnt
so far?

**Any
Question???**

What would you like
to clarify?

**The mind is just like a muscle
— the more you exercise it,
the stronger it gets and the
more it can expand.**

Idowu Koyenikan

Thank you!



Glory Praise Emmanuel is a Blockchain Developer/ Smart Contract Developer, and Technical Writer with experience developing high-performance solutions.


She is passionate about creating tech products and has experience writing performant smart contracts, and developing websites, web apps, and decentralized applications (DApps).

She is deeply committed to educating people about technology and assisting developers in becoming better at what they do.

 @emmaglorypraise

 Glory Praise Emmanuel

 @emmaglorypraise

 <https://glorypraise.hashnode.dev/>

