
Tanusree Sharma  
UIUC

Zhixuan Zhou  
UIUC

Andrew Miller  
UIUC

Yang Wang  
UIUC
Motivation

Long-term goal: design tools to identify and mitigate smart contract vulnerabilities

This study: understand how smart contract developers currently deal with security
contract Dao {
    mapping(address => uint256) public balances;
    function deposit() public payable {
        require(msg.value >= 1 ether, "Deposits must be no
            less than 1 Ether");
        balances[msg.sender] += msg.value;
    }
    function withdraw() public {
        // Check user's balance
        require(
            balances[msg.sender] >= 1 ether,
            "Insufficient funds. Cannot withdraw"
        );
        uint256 bal = balances[msg.sender];
        // Withdraw user's balance
        (bool sent, ) = msg.sender.call{value: bal}("”);
        require(sent, "Failed to withdraw sender's balance");
        // Update user's balance.
        balances[msg.sender] = 0;
    }
    function daoBalance() public view returns (uint256) {
        return address(this).balance;
    }
}
Working on a De-fi Project Development

Already vetted by community, so secure
Will have the audit afterward anyways internal / external

Forked codes of popular projects
Uniswap V2

Uniswap V2

Structure code design, format used standard library code

Reengineered
Uniswap V2
Changing logic pair/factory contract

Check lock modifier, look for common vulnerabilities, e.g. reentrancy

Optimized swap, used invariant used by Curve

Oftentimes don’t get time to delve into security vulnerability check

Fast shipping products to grow client/community

A Developer Journey (P1)
Research Questions

RQ1 How do smart contract developers ensure their smart contracts are secure against potential attacks?

RQ2 How do smart contract developers conduct code reviews and whether they are able to identify common smart contract security vulnerabilities in the code?
We conducted an interview and code review session with 29 Smart Contract Developers from 10 countries.
Survey + Code Review

We conducted online survey with **171** Smart Contract Developers

<table>
<thead>
<tr>
<th>Gender</th>
<th>Occupation</th>
</tr>
</thead>
<tbody>
<tr>
<td>69%</td>
<td>Male</td>
</tr>
<tr>
<td>31%</td>
<td>Female</td>
</tr>
<tr>
<td>79%</td>
<td>Full-time DeFi smart contract Developer</td>
</tr>
<tr>
<td>44%</td>
<td>Smart Contract Protocol development</td>
</tr>
<tr>
<td>35%</td>
<td>Smart Contract Development</td>
</tr>
<tr>
<td>13%</td>
<td>Smart Contract Research/security assessment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Years of Experience</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1 years</td>
<td>7.6%</td>
</tr>
<tr>
<td>1-3 years</td>
<td>25.7%</td>
</tr>
<tr>
<td>+3 years</td>
<td>66.7%</td>
</tr>
</tbody>
</table>
Results

- Security Perceptions
- Security Practices
- Security Behaviors
“Security was not a priority”

“If you’re planning to do an audit anyway, it kind of makes sense from a business perspective to ship code and then run it through multiple audits, instead of having your internal team [...] review the security at the same time.”

- P8
“Smart of Contract Security is Hard”

``
Contract work[s] like state machine when send a transaction. It only appears like state changes. But in regular program, you can differentiate read-only calls and state changes. Solidity can not do that.” – P19

Image source: https://www.flaticon.com/
Developers had broadly 3 common practices for security in smart contracts
Software engineering best practices

Importance of code refactoring & using vetted libraries

“write the most simple code that you can and draw the diagram to visualize the flow of smart contract code design” - P20

Common software testing techniques

Code reviews, input validations, and static analyses

“Having internal team for code review... in this culture of moving fast and breaking things. Also audits from external entities. - P10”

Specialized strategies

Creating own bytecode dictionary

“I created own bytecode (error code) dictionary to represent different cases of reverting transactions in his smart contracts for an NFT (non-fungible token) project - P18.”
Frequently used **Truffle testing suite, Remi, Hardhat, Slither, MythX**

Existing **symbolic execution based tools**, are limited in identifying edge cases.

**Use of Security Tooling, Limitations of smart contract security tools, & Code Review Practice**

**Manual inspection (64%)** was frequently used method for smart contract security
Developers Security Practices in Action
Overall, 55% of (16 out of 29) identify one or more vulnerabilities. 28% (N=8) of identified both (all) vulnerabilities.

20.5% (n: 171) identified vulnerability.
it is withdrawing if the amount is less than the amount to just return false and subtracts the amount before it does the accounting before it’s sending anything out, which is pretty crucial for preventing someone re-entering the function, which would be bad. - P14”
Design implications
Education & Standards

Hands-on exercises or labs, incorporate education teachable moments in Compilers, Security tools, IDEs, Testnets
Design implications

User interfaces & user experience

Actionable insights through Error / warning messages - zooming into where exactly the problems are in the code and how significant the effect can be

Image source: https://www.flaticon.com/
Thank you!

Contact:  
$tsharma6@illinois.edu$

@Tanusree_Sharma

Paper QR Code:

Key Takeaways

❖ **Limitation in tooling**
  ➢ Tailored Education, Standards, hands on Lab based on experience level
  ➢ Hierarchical and self explainable Error Message in security/ development platform
  ➢ Comprehensibility of Code libraries, symbolic execution tooling

❖ **Future Research can explore**
  ➢ Impact of Smart Contract Development Culture’s impact on security
  ➢ Comparison study with developers of different smart contracts language (e.g. solidity, vyper, etc)