Advanced Topics in Software Engineering
CSE - 6324 - Section 001

TEAM 4

ITERATION 1
(Written Deliverable)

Team Members

Suvarna, Arjun - 1002024437
Sinari, Navina - 1002072310
Palnati, Netra - 1002030626
Waje, Sanjana - 1002069940

I. Project Objective

Slither is a static analysis framework that provides rich information about Ethereum smart contracts [1]. Slither's plugin architecture lets you integrate new detectors from the command line [2].

We propose to add three new detectors using the Python API [3].

The first detector will be targeted toward the improper usage of the ecrecover() function in solidity smart contracts [4]. The ecrecover function can be used to process meta transactions in smart contracts and verifies signed data coming from someone other than the transaction signer [5], but due to the nature of public key private key signatures, this function is vulnerable to replay attacks [11]. The proposed detector will detect usage of the ecrecover function without nonce or chain id and flag it as a security vulnerability. The detector will also check the validity of the ECDSA signature based on the conditions defined in the Ethereum yellow paper [16]. This detector is based on the pull request (#2015) [17] which checks for nonce protection. Chain ID and ECDSA signature verification are being added to this detector to expand its use cases.

The second detector will detect API Keys stored as plain text within the solidity code (SWC 136) [6]. Hardcoded credentials present a grave security risk and compromise the security setup associated with the keys [6].

The third detector will detect passwords hardcoded into the solidity code (SWC 136) [6]. Hardcoded passwords in solidity smart contracts are not private and can be seen by third parties due to the public nature of smart contracts [7] [8].

II. Project Plan

Task	Start	End	Current Status	Risk Associated
Setup Development Environment	Inception	Inception	Completed	Changing Solidity Standards
Define rules for improper use of ecrecover() detector	Iteration 1	Iteration 1	Completed	Incomplete rule definition
Build improper use of ecrecover() detector (SWC-121)	Iteration 1	Iteration 1	Completed	1. Incomplete rule definition 2. False Positives 3. False Negatives 4. Change in security standards
Test and iterate improper use of ecrecover() detector	Iteration 1	Iteration 2	In Progress	Change in security standards
Define Rules for hardcoded API Key detector	Iteration 1	Iteration 2	In Progress	Change in security standards
Define Rules for hardcoded password detector	Iteration 1	Iteration 2	In Progress	Change in security standards
Review and implement feedback from Iteration 1	Iteration 2	Iteration 2	Not Started	
Build hardcoded API Key detector	Iteration 2	Final Iteration	Not Started	1. False Positives 2. False Negatives
Build hardcoded Password detector	Iteration 2	Final Iteration	Not Started	1. False Positives 2. False Negatives
Review and implement feedback from Iteration 2	Final Iteration	Final Iteration	Not Started	
Generate Report	Final Iteration	Final Iteration	Not Started	

III. Risk Mitigation Plan

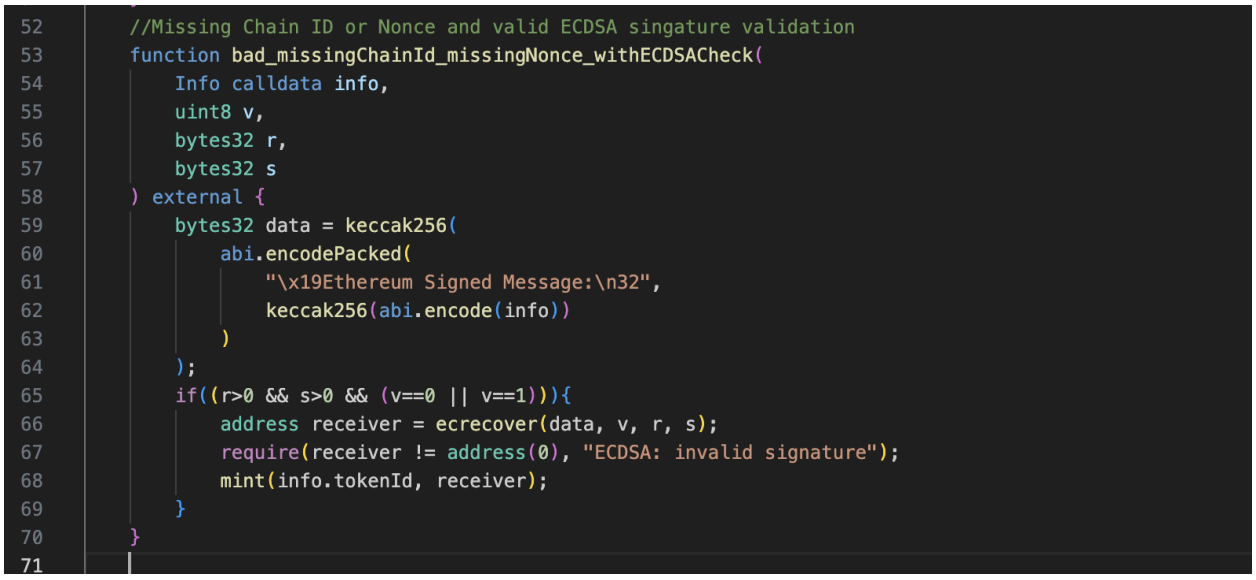
Risk	Risk Description	Mitigation Plan	Risk Exposure
Incomplete rule definition	Rules for detectors could miss edge cases	Iteratively build and test the detectors with rules defined.	There is a 30% probability that this issue might occur, and we would need 10 hours to fix it. Hence, risk exposure would be 3 extra hours.
Changing Solidity Standards	Changing standards in solidity and Ethereum could make the detector and its recommendations obsolete.	Use the latest standards in solidity with long-term support like EIP-712 or EIP 155	There is a 20% probability that this issue might occur, and we would need 10 hours to fix it. Hence, risk exposure would be 2 extra hours.
Missing Slither User's needs from the detector's	Rules defined in the detector may not meet the slither user needs for analyzing smart contracts	Thoroughly research the issue and interact with slither users to better understand detector needs	There is a 10% probability that this issue might occur, and we would need 10 hours to fix it. Hence, risk exposure would be 1 extra hour.
False Positive	The detector might detect that there is a data leak that does not exist.	Test with multiple negative scenarios	We need 4 hours to resolve this problem, which has a 20% chance of happening. Thus, 0.8 hours of risk exposure.
False Negatives	The detector may not detect that there is a data leak that exists.	Test with multiple positive scenarios	There is a 10% chance that this problem would arise, and it would take 4 hours to fix. Hence, risk exposure would be 0.4 extra hours.

IV. Specification and Design

In this iteration, we are checking for the improper usage of the ecrecover() function, which checks for certain conditions. The conditions are:

i. ECDSA check: checks the validity of the signature based on the conditions defined in the Ethereum yellow paper [16].
ii. ChainId check: By adding chain ID into the payload, replay attacks by sending a transaction from a different network as a valid transaction can be prevented [18].

We are using a solidity smart contract as input to our detector. For this iteration, we have written the code into 'ecrecover.sol' which was created in the pull request #2015 [17]. To analyze and detect the improper usages we have defined various functions that satisfy either of these conditions, or none of them, or satisfy both in our file. A few code snippets of the 'ecrecover.sol' are given below.



```
52      //Missing Chain ID or Nonce and valid ECDSA singature validation
53      function bad_missingChainId_missingNonce_withECDSACheck(
54          Info calldata info,
55          uint8 v,
56          bytes32 r,
57          bytes32 s
58      ) external {
59          bytes32 data = keccak256(
60              abi.encodePacked(
61                  "\x19Ethereum Signed Message:\n32",
62                  keccak256(abi.encode(info))
63              )
64          );
65          if((r>0 && s>0 && (v==0 || v==1))){
66              address receiver = ecrecover(data, v, r, s);
67              require(receiver != address(0), "ECDSA: invalid signature");
68              mint(info.tokenId, receiver);
69          }
70      }
71
```

```
110
111     //Valid with Chain Id and ECDSA singature validation
112     function good_withChainId_withECDSACheck(
113         Info calldata info,
114         uint8 v,
115         bytes32 r,
116         bytes32 s
117     ) external {
118         bytes32 hash = keccak256(abi.encode(info,chainId));
119         bytes32 data = keccak256(
120             abi.encodePacked("\x19Ethereum Signed Message:\n32", hash, chainId)
121         );
122         if((r > 0 && s>0 && (v == 0 || v == 1))){
123             address receiver = ecrecover(data, v, r, s);
124             require(receiver != address(0), "ECDSA: invalid signature");
125             mint(info.tokenId, receiver);
126         }
127     }
```

```
145
146     //Valid Nonce and  missing ECDSA singature validation
147     function bad_withNonce_missingECDSACheck(
148         Info calldata info,
149         uint8 v,
150         bytes32 r,
151         bytes32 s
152     ) external {
153         bytes32 data = keccak256(
154             abi.encodePacked(
155                 "\x19Ethereum Signed Message:\n32",
156                 keccak256(abi.encode(info,nonces[msg.sender]++))
157             )
158         );
159             address receiver = ecrecover(data, v, r, s);
160             require(receiver != address(0), "ECDSA: invalid signature");
161             mint(info.tokenId, receiver);
162
163     }
164
```

The detector analyzes the code and detects missing conditions for every function. The output of the detector analysis is given below:

```
navina@Navinas-MacBook-Air code % slither ecrecover.sol --detect ecrecover
'solc --version' running
'solc ecrecover.sol --combined-json abi,ast,bin,bin-runtime,srcmap,srcmap-runtime,userdoc,devdoc,hashes --allow-paths .,/Users/navina/Downloads/Adv SE/Code' running
Compilation warnings/errors on ecrecover.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> ecrecover.sol


INFO:Detectors:
A.bad_withChainID_missingECDSACheck(A.Info,uint8,bytes32,bytes32).receiver (ecrecover.sol#35) lacks a zero-check on :
                - receiver = ecrecover(bytes32,uint8,bytes32,bytes32)(data,v,r,s) (ecrecover.sol#35)
A.bad_withChainID_missingECDSACheck(A.Info,uint8,bytes32,bytes32).receiver (ecrecover.sol#35) lacks a r > 0 ecdsa check on :
                - receiver = ecrecover(bytes32,uint8,bytes32,bytes32)(data,v,r,s) (ecrecover.sol#35)
A.bad_withChainID_missingECDSACheck(A.Info,uint8,bytes32,bytes32).receiver (ecrecover.sol#35) lacks a s > 0 ecdsa check on :
                - receiver = ecrecover(bytes32,uint8,bytes32,bytes32)(data,v,r,s) (ecrecover.sol#35)
A.bad_withChainID_missingECDSACheck(A.Info,uint8,bytes32,bytes32).receiver (ecrecover.sol#35) lacks a v ecdsa check on :
                - receiver = ecrecover(bytes32,uint8,bytes32,bytes32)(data,v,r,s) (ecrecover.sol#35)
A.bad_missingChainID_missingECDSACheck(A.Info,uint8,bytes32,bytes32).receiver (ecrecover.sol#49) lacks a zero-check on :
                - receiver = ecrecover(bytes32,uint8,bytes32,bytes32)(data,v,r,s) (ecrecover.sol#49)
A.bad_missingChainID_missingECDSACheck(A.Info,uint8,bytes32,bytes32) (ecrecover.sol#39-51) lacks a nonce or chainId on :
                - data = keccak256(bytes)(abi.encodePacked(Ethereum Signed Message:
32,hash)) (ecrecover.sol#46-48)
```

V. Code and Tests

i. Setup:

There are various versions available for the solc compiler [19].

```
navina@Navinas-MacBook-Air code % solc-select install
Available versions to install:
0.3.6
0.4.0
0.4.1
0.4.2
```

We are using version 0.8.18, which can be installed using the following command.

```
navina@Navinas-MacBook-Air code % solc-select install 0.8.18
Installing solc '0.8.18'...
Version '0.8.18' installed.
```

To use the specific version, we need to use the following command.

```
navina@Navinas-MacBook-Air code % solc-select use 0.8.18
Switched global version to 0.8.18
```

The next step is to add our ecrecover python file, ‘ecrecover.py’ which has the detector code, in ‘all_detectors.py’ which would be in the local directory where Slither is installed [2]. In our case, the path is: ‘/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/slither/detectors/all_detectors.py‘ and then move our detector code to the local directory path ‘/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/slither/detectors/statements’.

After the installations and setup, we are taking ‘ecrecover.sol’ as input and running slither only for the ecrecover detector using the following command.

```
navina@Navinas-MacBook-Air code % slither ecrecover.sol --detect ecrecover
```

ii. Code:

The detector code snippets are as follows:

The code is built on ecrecover.py from pull request #2015 [17]

The detector looks for uses of ecrecover() which if returns true will check if keccak256 function is called and checks for the nonce or chain id conditions as well as the ECDSA signature validation.

```
147   #Checks the ECDSA validation for the r value and returns true or false
148   def _r_ecdsa_validation(var: LocalVariable, function: Function) -> bool:
149       for node in function.nodes:
150           if node.contains_if():
151               for ir in node.irs:
152                   expression = str(ir.expression)
153                   if isinstance(ir, Binary):
154                       if (
155                           ir.type == BinaryType.GREATER or ir.type == BinaryType.LESS
156                       ) and "r > 0" in expression or "0 < r" in expression:
157                           return True
158       return False
```

```
159
160   #Checks the ECDSA validation for the s value and returns true or false
161   def _s_ecdsa_validation(var: LocalVariable, function: Function) -> bool:
162       for node in function.nodes:
163           if node.contains_if():
164               for ir in node.irs:
165                   expression = str(ir.expression)
166                   if isinstance(ir, Binary):
167                       if (
168                           ir.type == BinaryType.GREATER or ir.type == BinaryType.LESS
169                       ) and ("s > 0") in expression or ("0 < s") in expression:
170                           return True
171       return False
```

```
172
173   #Checks the ECDSA validation for the v value and returns true or false
174   def _v_ecdsa_validation(var: LocalVariable, function: Function) -> bool:
175       for node in function.nodes:
176           if node.contains_if():
177               for ir in node.irs:
178                   expression = str(ir.expression)
179                   if isinstance(ir, Binary):
180                       if (
181                           ir.type == BinaryType.EQUAL
182                       ) and ("v == 0" in expression) or ("v == 1" in expression):
183                           return True
184       return False
185
```

```
    #If ecrecover function is called
    if SolidityFunction("ecrecover(bytes32,uint8,bytes32,bytes32)") in function.solidity_calls:
        address_list = []
        var_nodes = defaultdict(list)
        for node in function.nodes:
            if SolidityFunction("ecrecover(bytes32,uint8,bytes32,bytes32)") in node.solidity_calls:
                for var in node.local_variables_written:
                    if "ecrecover(bytes32,uint8,bytes32,bytes32)" in str(var.expression):
                        address_list.append(var)
                        var_nodes[var].append(node)

            #If keccak256 hash function is called
            if SolidityFunction("keccak256(bytes)") in node.solidity_calls:
                for ir in node.irs:
                    if isinstance(ir, SolidityCall) and ir.function == SolidityFunction(
                        "keccak256(bytes)"
                    ):
                        # Checking if nonce or chainId is present in the expression
                        if "nonce" not in str(ir.expression) and "chainId" not in str(ir.expression):
                            nonce_results.append(node)

```

iii. Test Cases

No.	Test Case	Expected Output
1.	Running the pull request (#2015) on the solidity code	Analyzing the usage of nonce and zero address check
2.	Implementing the chainId check on the pull request	Analyzing slither detector with chainId check
3.	Implementing the ECDSA check on the pull request	Analyzing slither detector with ECDSA check

VI. Competitors

i. Slither:- The current stable build of slither (0.9.6) [9], has a robust set of public detectors like Dangerous usage of tx. origin and Unchecked low-level calls [10]. But Slither currently does not have a public detector for API Keys, Passwords, or a public detector for improper usage of the ecrecover function as described in the GitHub Issue #1950 [4].

ii. Mythril:- Mythril is a security analysis tool for Ethereum smart contracts [12]. It has a set of modules to detect vulnerabilities and issues in the Ethereum solidity code like External Calls and Unprotected Ether Withdrawal [13]. But there are no modules to detect unprotected use of the ecrecover function or hardcoded credentials.

iii. MythX:- MythX is a tool that scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts [14]. However, it does not have a detector available for improper use of ecrecover or hardcoded API keys and passwords [15]. MythX is a closed-source tool.

VII. Customers and Users

User	**Role**	**Feedback**
Mehul Hivlekar	New User to Smart Contracts and Solidity	The new detector covers many use cases and adds checks before the smart contract is deployed.
Shruthaja Patali Rao	CSE-6324-001 Team 5	SWC-121 has evolving standards to mitigate and must be monitored to keep the detector effective
Devyani Singh	CSE-6324-001 Team 8	More cases can be explored by the detector to enhance coverage.

VIII. GitHub Repository

https://github.com/arjunsuvarna1/CSE6324_Team4_Fall23

IX. References

[1] Feist, Josselin & Grieco, Gustavo & Groce, Alex. (2019). Slither: A Static Analysis Framework For Smart Contracts.

[2] Adding a new detector - https://github.com/crytic/slither/wiki/Adding-a-new-detector

[3] Python API - https://github.com/crytic/slither/wiki/Python-API

[4] Improper usage of ecrecover - https://github.com/crytic/slither/issues/1950

[5] What is ecrecover in Solidity? - https://soliditydeveloper.com/ecrecover

[6] Unencrypted Private Data On-Chain - https://swcregistry.io/docs/SWC-136/

[7] CWE-798: Use of Hard-coded Credentials - https://cwe.mitre.org/data/definitions/798.html

[8] Sensitive Information Disclosure in Smart Contracts - https://knowledge-base.secureflag.com/vulnerabilities/sensitive_information_exposure/sensitive_information_disclosure_smart_contracts.html

[9] Slither - https://github.com/crytic/slither

[10] Slither Detector Documentation - https://github.com/crytic/slither/wiki/Detector-Documentation

[11] SWC-121: Missing Protection against Signature Replay Attacks - https://swcregistry.io/docs/SWC-121/

[12] Mythril - https://github.com/Consensys/mythril

[13] Mythril Analysis Modules - https://mythril-classic.readthedocs.io/en/develop/module-list.html

[14] MythX - https://mythx.io/about/

[15] MythX Detectors - https://mythx.io/detectors/

[16] Ethereum: A Secure Decentralized Generalized Transaction Ledger - https://ethereum.github.io/yellowpaper/paper.pdf

[17] Detector for ecrecover return value validation and use of nonces - https://github.com/crytic/slither/pull/2015

[18] Preventing Replay Attacks Post Ethereum Merge - https://quantstamp.com/blog/preventing-replay-attacks-post-ethereum-merge

[19] Installing the Solidity Compiler - https://docs.soliditylang.org/en/v0.8.9/installing-solidity.html