**Analyze the result of vulnerability Detection using Smart Check**

**2017313135 소프트웨어학과 권동민**

**About what is Smart Check and why is it necessary**

“Solidity” is the programming language for smart contract. The concept of smart contract existed before by using programming, but it has grown very rapidly since the advent of blockchain technology. This is because many network participants guarantee the transaction, this proves that the transaction is secure.

But There is vulnerability inevitably because it is also a type of program.(Can be hacked) There are some people who make their own smart contract rule, but it can be used by everyone only when the contract is not vulnerable. But it is too hard to achieve.

Smart Check is the simple library to check vulnerability of smart contract.(.sol files)

**Analyze vulnerability of test Data using smart check**

**Test code environment**

텍스트이(가) 표시된 사진

자동 생성된 설명

Linux version : Ubuntu 20.04.3 LTS

Jdk version : 1.8.0\_292

Text editor(for solidity test) : visual studio code for linux

Test Data : dao\_attack code(in pdf file), smart\_chk\_test(some codes in github), test\_Data(Data1.zip)



**How The Smart Check works**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

If I execute above command, we can see some information(rule id, pattern id, line, content …) about that file/directory. And in the last of the print lines, there are the number by which rule-id is identified.

In git-hub of SmartCheck, there is the file “Solidity.g4” that is written by antlr4. In other words, I can infer that Smart Check analyzes the grammar of solidity and make parsing tree, after that, find some pattern of parsed terminal words and set the rule-id and other information. Actually, when there are some difference between lexer&parser rule and codes, it prints some parsing error like as follows.

텍스트, 오렌지, 닫기, 확대이(가) 표시된 사진

자동 생성된 설명

**Frequent rule-id by testing sample\_data(Data1.zip)**

There are many rule-id in Smart Check library. But I just explain some representative rule-id that is shown in test code frequently.

SOLIDITY\_PRAGMAS\_VERSION

텍스트이(가) 표시된 사진

자동 생성된 설명



: when call pragma solidity version, set to ruleId to SOLIDITY\_PRAMAS\_VERSION

SOLIDITY\_SAFEMATH:

텍스트이(가) 표시된 사진

자동 생성된 설명

When call the safeMath functions, SmartCheck set the rule-id to SOLIDITY\_SAFEMATH. If we use safeMath function in our own solidity, we can avoid overflow & underflow.

SOLIDITY\_OVERPOWERED\_ROLE

텍스트이(가) 표시된 사진

자동 생성된 설명

When check all the contents of that messages, I can find that all function had “only Owner” in their content. SmartCheck judges that “only Owner” is the overpowered role in that case.

SOLIDITY\_ADDRESS\_HARDCODED

텍스트이(가) 표시된 사진

자동 생성된 설명

If addresses is written in hardcoded value directly, SmartCheck set the ruleId to SOLIDITY\_ADDRESS\_HARDCODED. It is very dangerous because there is a risk of being attacked by hackers.(they can get the address from that code)

SOLIDITY\_GAS\_LIMIT\_IN\_LOOPS

텍스트이(가) 표시된 사진

자동 생성된 설명

Each operation consumes some gas to operate, So in loops, if the consumed gas is more than Limit, Set the ruleId to SOLIDITY\_GAS\_LIMIT\_IN\_LOOPS.

SOLIDITY\_USING\_INLINE\_ASSEMBLY

텍스트이(가) 표시된 사진

자동 생성된 설명

: When we use “assembly” to insert assembly code in solidity grammar, set the ruleId to SOLIDITY\_USING\_INLINE\_ASSEMBLY.

SOLIDITY\_CALL\_WITHOUT\_DATA

텍스트이(가) 표시된 사진

자동 생성된 설명

When function calls, if the value of that function is empty, then set to ruleID to SOLIDITY\_CALL\_WITHOUT\_DATA.

**Limit of SmartCheck : my opinion**

After I used that library, I felt that it is similar with parser operation. So It can find vulnerability in the codes by analyze parsing tree(just check the content’s structure) So it seems that it can’t check run-time vulnerability. Actually in Dao-attack codes, the result is as follows.



텍스트이(가) 표시된 사진

자동 생성된 설명

In that case, EtherStore.sol has vulnerability that can call function recursively in internally, but it can’t be detected by using SmartCheck.