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

```
kdm@kdm-VirtualBox:~/work/smartchk_test/test_Data$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.3 LTS"
kdm@kdm-VirtualBox:~/work/smartchk_test/test_Data$ java -version
openjdk version "1.8.0_292"
OpenJDK Runtime Environment (build 1.8.0_292-8u292-b10-0ubuntu1~20.04-b10)
OpenJDK 64-Bit Server VM (build 25.292-b10, mixed mode)
```

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)

```
kdm@kdm-VirtualBox:~/work/smartchk_test$ ls
dao_attack  smart_chk_test  test_Data
```

How The Smart Check works

```
kdm@kdm-VirtualBox:~/work/smartchk_test/test_Data$ smartcheck -p ./00d9a.sol
./00d9a.sol
jar:file:/usr/local/lib/node_modules/@smartdec/smartcheck/jdeploy-bundle/smartcheck
patternId: 49bd2a
severity: 1
line: 54
column: 20
content: throw
```

```
SOLIDITY_VISIBILITY :26
SOLIDITY_DEPRECATED_CONSTRUCTIONS :19
SOLIDITY_REVERT_REQUIRE :13
SOLIDITY_VAR :2
SOLIDITY_EXTRA_GAS_IN_LOOPS :1
SOLIDITY_ERC20_APPROVE :2
```

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.

```
line 1392:15 mismatched input '==' expecting ')'
line 1392:24 mismatched input '(' expecting {';', '='}
line 1392:25 mismatched input '0' expecting {'solidity'
```

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

```
ruleId: SOLIDITY_PRAGMAS_VERSION
patternId: 23fc32
severity: 1
line: 20
column: 16
content: ^
```

```
pragma solidity ^0.5.1;
```

: when call pragma solidity version, set to ruleId to SOLIDITY_PRAMAS_VERSION

SOLIDITY_SAFEMATH:

```
ruleId: SOLIDITY_SAFEMATH
patternId: 837cac
severity: 1
line: 101
column: 4
content: usingSafeMathforuint256;
```

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

```
ruleId: SOLIDITY_OVERPOWERED_ROLE
patternId: j83hf7
severity: 2
line: 217
column: 4
content: functionsetAddressToExcludeSenders(addressaddr)publiconlyOwner{excludeSendersAddresses[
addr]=ExcludeAddress({isExist:true});}
```

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

```
ruleId: SOLIDITY_ADDRESS_HARDCODED
patternId: adc165
severity: 1
line: 252
column: 26
content: 0xd03B6ae96CaE26b743A6207DceE7Cbe60a425c70
```

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

```
ruleId: SOLIDITY_GAS_LIMIT_IN_LOOPS
patternId: f6f853
severity: 2
line: 432
column: 8
content: for(uinti=0;i<proxies.length;i++)_IS_ALLOWED_PROXY_[proxies[i]]=true;
```

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

```
ruleId: SOLIDITY_USING_INLINE_ASSEMBLY
patternId: 109cd5
severity: 1
line: 141
column: 4
content: assembly{impl:=sload(slot)}
```

: When we use "assembly" to insert assembly code in solidity grammar, set the ruleId to SOLIDITY_USING_INLINE_ASSEMBLY.

SOLIDITY_CALL_WITHOUT_DATA

```
patternId: om991k
severity: 2
line: 17
column: 27
content: call.value(_weiToWithdraw)()
```

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.

```
kdm@kdm-VirtualBox:~/work/smartchk_test/dao_attack$ smartcheck -p ./EtherStore.sol
```

```
SOLIDITY_PRAGMAS_VERSION :1
SOLIDITY_UPGRADE_TO_050 :1
SOLIDITY_CALL_WITHOUT_DATA :1
```

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