## 1.Integer Overflow and Underflow

## 1.1Introduction

As the longest and most classic type of vulnerability on smart contract,Integer overflows and underflows are always detected from many contract in a new form.The Solidity language does not support floating-point data types like float double in C. Supports signed or unsigned integers with int/uint lengthening. The variable supports from uint8 to uint256, and int8 to int256. Please note that uint and int default to uint256 and int256. The value range of uint8 is the same as uchar in C, that is, the value range is 0 to 2^8-1, and the range of values supported by uint256 is 0 to 2^256-1, and the rest of the data types and so on.Basically,the data type we use most in solidity is uint256.As mentioned before,it range 0 to 2^256-1 under the binary and 0 to 115792089237316195423570985008687907853269984665640564039457584007913129639935 in decimal.

## 1.2 code demo

In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits – either larger than the maximum or lower than the minimum representable value.Usually,in solidity,flow occur in such situation:

overflow in conversion between unsigned and signed; overflow in addition or multiplication of two (un)signed numbers

underflow in subtraction of two (un)signed numbers;overflow in ++ on a (un)signed number

underflow in -- on a (un)signed number;

overflow in += ;overflow in -= ;overflow in \*= ;overflow in /=

The code of figure 1 and result from figure 2 will show what is flow:

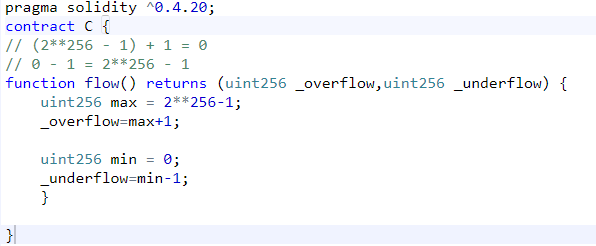


Figure 1:code of flow

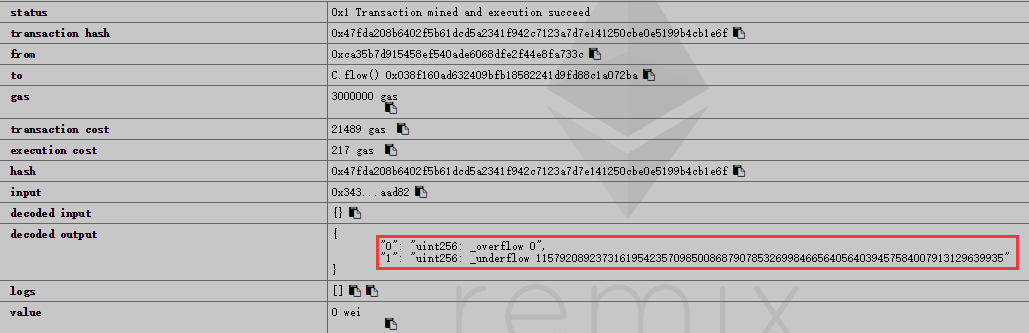


Figure 2: result of flow

As can be seen, after overflowing, \_overflow directly wraps around with a return value of 0. When uint256 takes 0 underflow, it directly wraps around, and the return value is 2^256-1. This is the normal case of integer overflow scenarios in solidity.

## 1.3 Vulnerability demo

In this part,there will be two simple code to show how the vulnerability may happen



As show in figure..,this example is a result of arithmetic on two unsigned integers is an unsigned integer,which means they will never get a true negative number but only wrap around the range of uint256.So this function never pass the check in “if” case.



The second example explain the soon-to-be-deprecated var keyword. Because while compiling the variable with keyword var is always regarded as the smallest type needed to contain the assigned value, it will become an uint8,which can only value max to 255 to hold the initial value 0. If the loop is meant to iterate more than 255 times, it will never reach that number,keep wrap around from 0 to 255, stop until the execution runs out of gas:

## 1.4 Fix

To fix overflow and underflow is easy,there is already official way to deal with it,using SafeMath library in contract.After import the library from figure 3 and reedit the code like figure 4 and figure 5

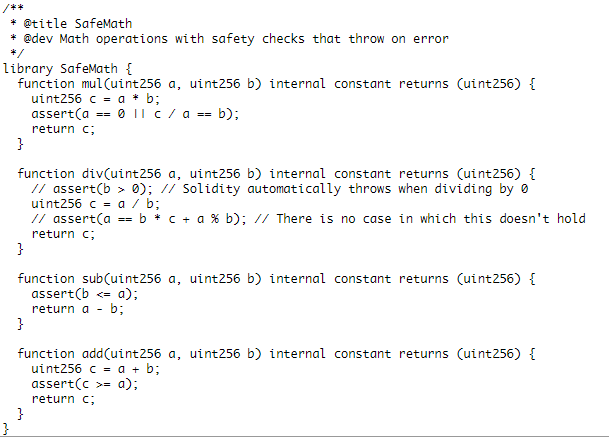


Figure 4:safemath library

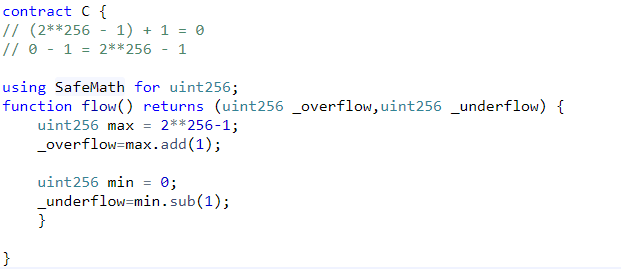


Figure 5:modification on code

If there is still flow happen,the results of this operations will be checked and an error will be thrown stopping the execution of your smart contract.

## 1.5 Malicious Use in Real World

Until now,most attacker use overflow to escape the amount check while transfer balance,some famous case like batchoverflow,multioverflow,and two case will be introduced later.The weird thing is not difficult to implement an attack with flow,the attackers just need to find out a contract with token or transfer function which forget to make sure all the check in the function use safemath,then attackers can try until the input data will flow so that can pass the check.This always do like transfer much of balance to some address with no or very little consumption from msg.sender or from address in the Input parameters.As the result,it create a lot of nonexistent Ethereum or token,which may lead the market break down.

## Proxy Overflow

* 1. Introduction of ERC20

The ERC20 token standard describes the functions and events that an Ethereum token contract has to implement.Basically use like a interface,has the following method-related functions,balanceOf(),transfer(),approve(),some events like Transfer to record balance transaction and more.

## Real World Case

There are quite a few ERC20 tokens affected by this overflow,here choose one of them “SMT” to demonstrate.The address of SMT contract:

https://etherscan.io/address/0x55f93985431fc9304077687a35a1ba103dc1e081#code

ProxyTransfer is a new EIP in a process wherein the transaction fees are paid by the users in the form of tokens and not Ether, as opposed to traditional ERC20 contracts. In a traditional ERC20 contract, even if the user has 100 tokens, he cannot simply operate unless he holds some Ether too. However, this doesn’t really make sense as ‘Tokens themselves hold value and to say that tokens can only act in conjunction with Ether makes them unusable to a certain extent’. To fix this, a group of developers introduced a new proposal(EIP-xyz) where users can spend tokens as Transaction Fees. EIP gained a lot of traction that several DApps tried to incorporate, while the proposal itself is reviewed. However, some of those contracts came with an exploit and hidden cost.

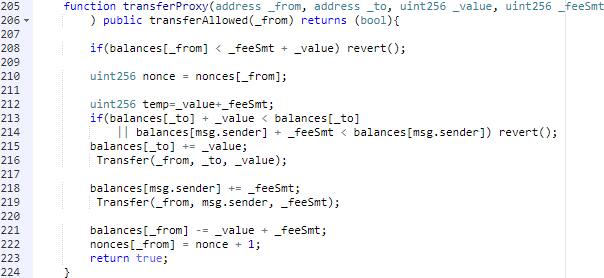


Figure 6: overflow from transferProxy()

To make to convenient for test,here remove the verification code of digital signature from token sender.As the code shown in figure 6,before running this function,system will check modifier of transferAllowed()

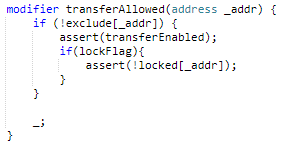


Figure 7: transferAllowed modifier

Basically after the contract created on the chain,the owner will set transferEnabled to true.If the \_from address is in exclude mapping,it can pass this check.If not,the address can pass if it is not been locked by the owner in the locked mapping.To summary,the transferProxy function is allowed to run if the from address hasn’t been locked and the contract is allowed to transfer.Then the overflow occurs in the first “if” case in figure 6.Both \_fee and \_value are input parameters which could be controlled by the attacher. If \_fee + \_value happens to be 0,the check in line 208 could be passed.It means the attacker could transfer huge amount of tokens to an address (line 215) with zero balance. Also, a huge amount fee would be transferred to the msg.sender in line 219.But in line 221,because overflow,the from address would not reduce any amount of balance.So this vulnerability does not lead to generating countless tokens which should not be exist.

## demo

Using the uint data from this transaction to do test,

<https://etherscan.io/tx/0x1abab4c8db9a30e703114528e31dee129a3a758f7f8abc3b6494aad3d304e43f>

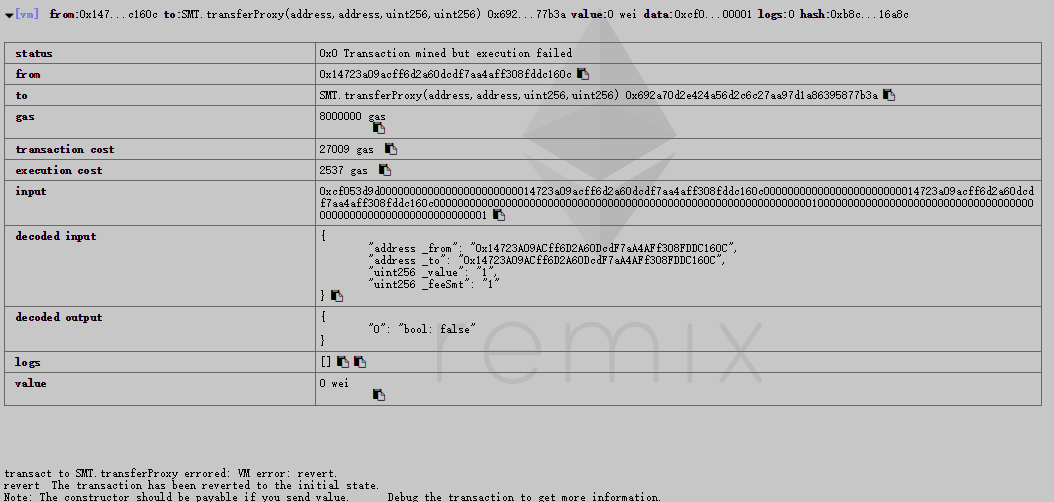
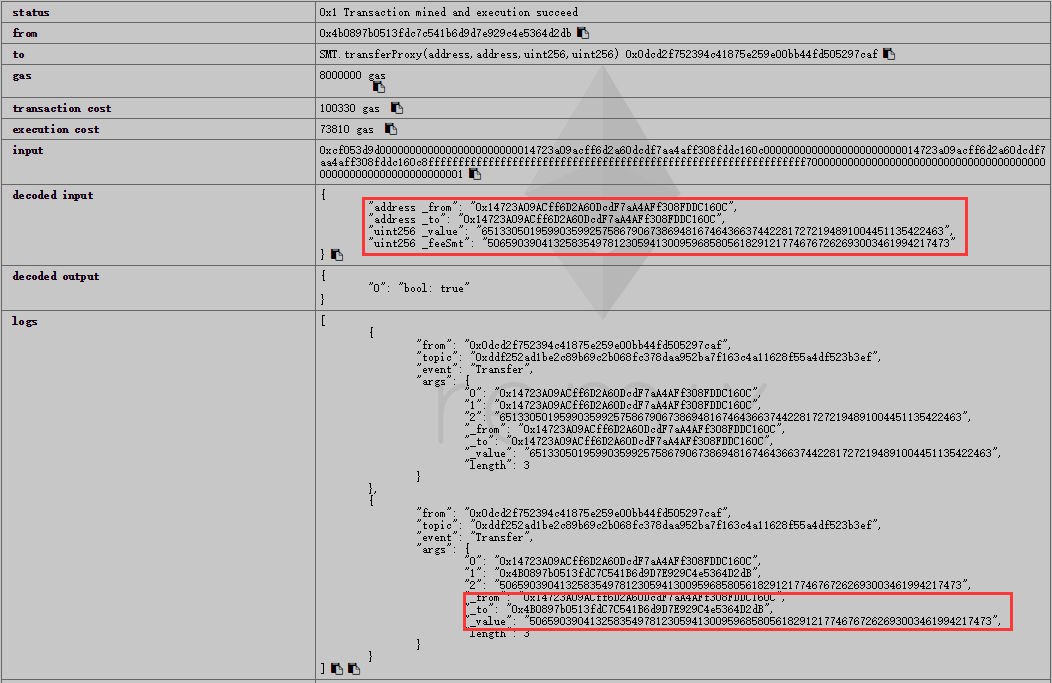


Figure 8: test without overflow



Msg.sender

Figure 9:test with overflow

All the address send from case have no balance on this contract,so from figure 8,the transaction will revert to initial state after check .After using data which will lead overflow in figure 9,the function return true means transaction success and balance transferred.As the third red rectangle,exactly a big amount transfer from \_from to msg.sender.As contract’s balances make up users’ amount,check the balance of these two address.

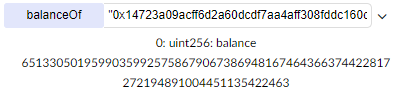
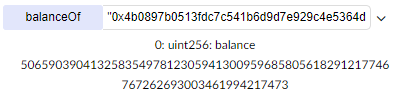


Figure 10: balance of \_from and msg.sender

As data from figure 10,it is clear that both two address increase great amount of balance but without decrease any amount in \_from.Actually the balances’ increase depend on case:”balances[address]+=value;”,so from line 215,218,221 in figure 6,only increase on these address but no decrease on \_from address because overflow,it reduce 0.

According this demo,obviously it create huge amount of non-existing Ethereum to attacker’s account,leading collapse of Ethereum market.Still kind of contracts like GG Token, Mesh and so on ,are exposed to proxyoverflow vulnerability.

## Fix

Almost every contract that has Integer flow vulnerability can be fixed by the same way using safemath library,including this proxyoverflow.

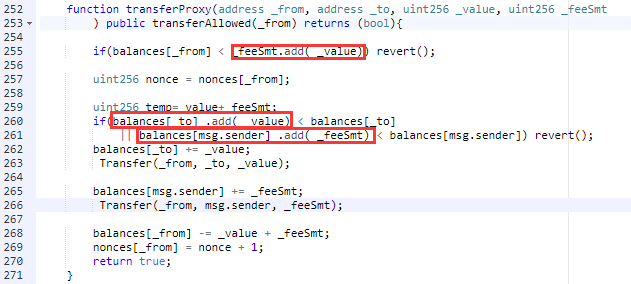


Figure 10:modification on transferProxy function

Following the official method,after importing the safemath library,the necessary step to edit the code is to use add method from safemath to do the addition operation.If the result will overflow,it will revert the transaction back to initial state.

## 3 B**urnOverflow**

## 3.1 Real World Case

The vulnerability of overflow always lead to generate tokens out of nowhere including this burnoverflow.This case seems pretty simply because contract programmer just implement the logic of the standard ERC-20 transfer() function by calling his customized function with conventional check,still with some negligence.Here is the contract address:

<https://etherscan.io/address/0xB5335e24d0aB29C190AB8C2B459238Da1153cEBA#code>

Now let’s analyze the code structure:

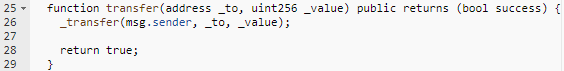


Figure 11:implementation of standard transfer function

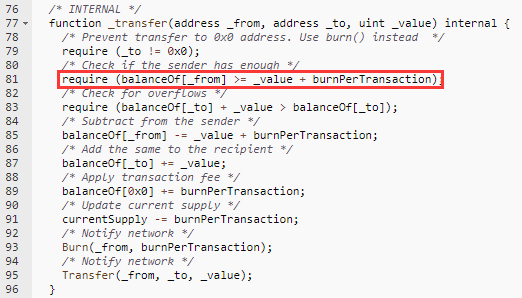


Figure 12:detail of \_transfer function

As show in figure 11 and 12,though the \_transfer function is internal,but it is called by public function \_transfer.Making the same mistake,in this contract the programmer did not use safemath to check data in line 81,while the constant variable “burnPerTransaction” is set as2 by default,so the attacker can make a situation that \_value + burnPerTransaction = 0 . As the balance of \_to is less than 2, the check in line 85 could be passed. Then, the balance of \_from is decremented by 0 (\_value + burnPerTransaction) in line 85. Finally, the tremendous amount of HXG token is added to balanceOf[\_to] in line 87.Very similar with proxyoverflow,it create a plenty of nowhere token.

## 3.2 Demo

Using the uint data from this transaction to do test,

<https://etherscan.io/tx/0xac483f3be64994a7002805c6fc576a1a4dcef6bb90f0d1194b2cc5edc6cc0b0f>

Note that the addresses used during test have no initial balance on this contract,except the contract creator has 420000000000000 balance from the constructor,assigned from the constant variable “initialsupply”.

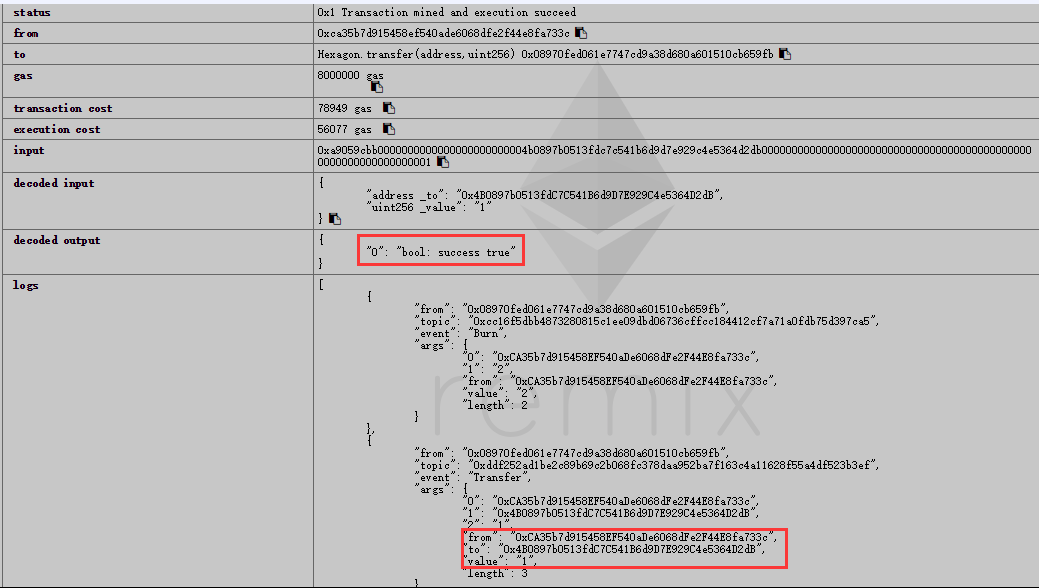


Figure 14:test with normal input data and success result



Figure 15: balance of “to” address after transaction

As the transaction initiator is contract creator who has balance on the contract,it can finish the transaction to send balance to other address.So after the transaction check the public mapping variable balancedOf with the address just send balance,can successfully find out 1 value balance.

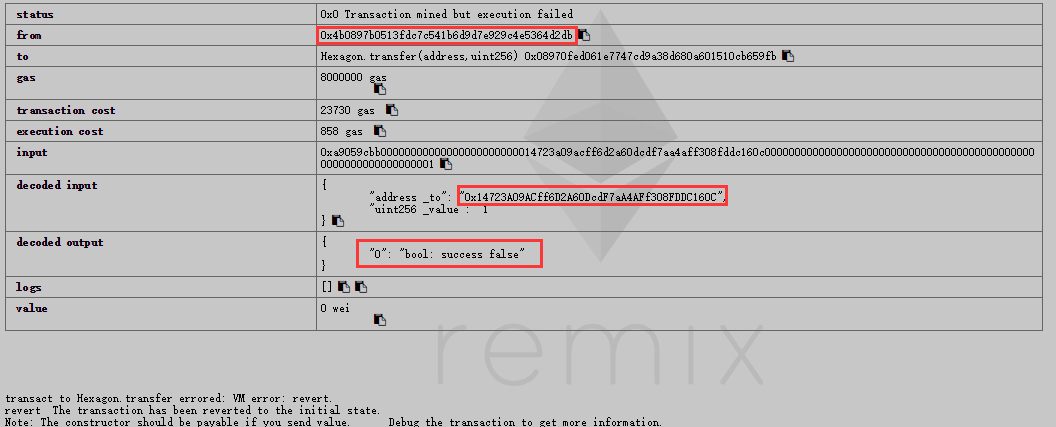


Figure 15: test with normal revert situation

Because the test address has no balance on contract,as the transaction initiator,it can not pass the check in line 81 from figure 12 as

(balanceOf[\_from])=0 < 3=(\_value+burnPerTransaction),resulting in the transaction revert.Then following how to attack.

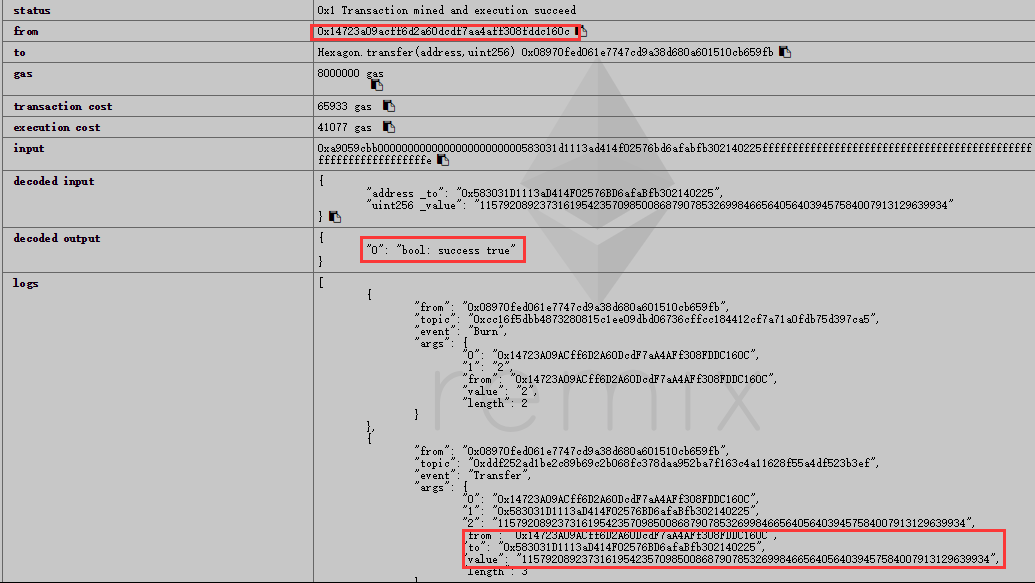


Figure 16: test with overflow

As shown in figure 16,the sender has no balance on contract but make the transaction success.

After the input data \_value+burnPerTransaction,overflow occur and the result become 0,so it can pass the check and send non-existing tremendous balance to any address freely without any consumption of token.

## 3.3 Fix

Basically the same way as fix on proxyoverflow.

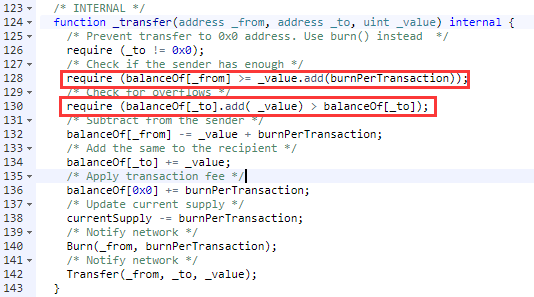


Figure 17:modification on \_transfer function

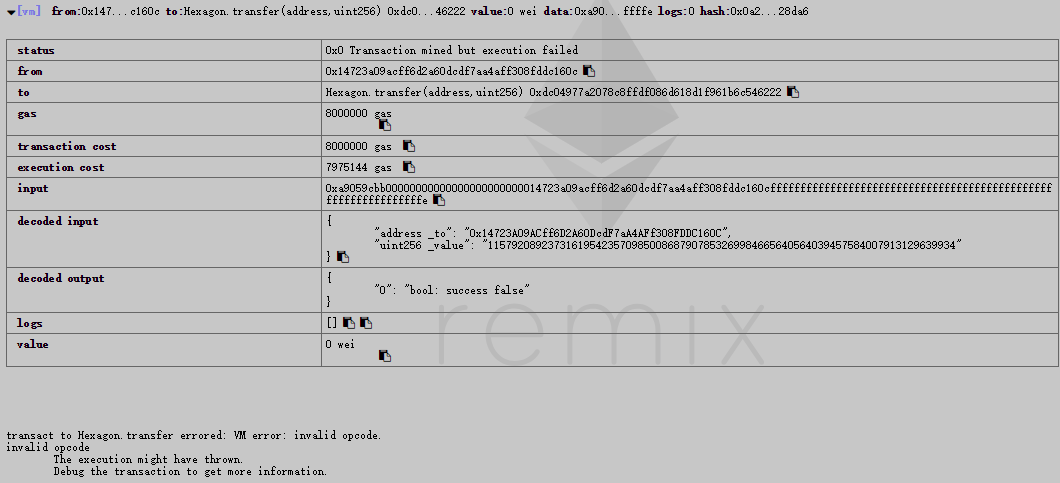


Figure 18: test with safemath

After using safemath,in line 128,the add method will check whether there is overflow and use assert to report.Additional,about this contract,it’s supposed to implement standard ERC20 function in this case.There are common rule for this:define a Token contract as top father class to provide interface for ERC token,then achieve the detail of standard function in ERC20 contract inherit from Token.At last,finish your own contract inherit from ERC 20with customize code to call standard as you like.

## 4.Summary

Seem the current compiler can not perfectly fix the integer overflow and underflow,since such flow vulnerability already caused great damage on block chain .But if import safemath into compiler as default detector for all uint or int type,that would case great gas to run it.Execution of Ethereum smart contracts is limited by the gas limit on each block which is currently around 6m. This only allows for between one thousand and one million instructions in each block (every 15 seconds). Checking for overflow a million times each second is not expensive on processors produced in this millennium. I have to disagree with you on this answer.It would be bad for overall EVM performance and by extension bad for tx rate, node performance.To prevent flow vulnerability mostly depend on contract developer themselves However,quite a lot contract developers can not realize the severity yet,forget to check the vulnerability then allow malicious attack,leading many time of revert and waste of space on chain.No absolutely way to prevent this

Vulnerability,only try best to make code normalization.

As for flow,these step may be useful:

1. after declare the version of pragma solidity,import the safemath library.
2. After define contract,using safemath for uint256 at first.
3. Use uint256 but not other bits of uint as far as possible.
4. Check every arithmetic operation handle with outside data like the input parameter,then use method add,sub,mul,div from safemath instead of +,-,\*,/.
5. Test your contract,especially the transfer or withdraw function.