**Example Legends**

\_ 🡪 Any expression

Uop 🡪 Unary decrement, and increment operations (++, --)

Bop 🡪 Binary Arithmetic operations (+,-,%,/)

**Protector function**

Check(Current Block Number) 🡪 Check the execution flow integrity.

Record(Current Block Number) 🡪 Save current block number that is will be executed right next.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Basic rules – to ensure execution flow integrity** | | | | |
| Block Start | | **Expression** | **Rewritten code** | **Comment** |
| \_; | Check(cur\_block\_no);  \_; | Inject Check(); before the user code. |
| Block End | | **Expression** | **Rewritten code** | **Comment** |
| \_; (except for break, return, call) | \_;  Record(cur\_block\_no); | Inject Record(); after the user code. |
| break or return or call; | Record(cur\_block\_no);  break or return or call; | Inject Record(); before the user code where the end of code is break or return or call(). |
| **Extended rules – to ensure value range and integer overflow** | | | | |
| - | **Statement** | | **Rewritten code** | **Comment** |
| a = x +(Bop) y; | | If (CheckIntAdd (x,y))  a = IntAdd(x,y); // SafeMath  else revert(); | This rule supports other arithmetic operations (+,/, %, and so on). |
| a = x +(Bop) y +(Bop) z; | | if (CheckIntAdd (x,y))  if (CheckIntAdd(IntAdd(x,y),z))  a = IntAdd(IntAdd(x,y),z);  else revert();  else revert(); | How to handle this case? |
| a --(Uop); | | If (CheckDecrementOp(a))  a--;  else revert(); | This rule also supports prefix increment operation(++). |
| for (int x = 0; x < y; x++)  {  \_;  } | | for (int x = 0; x < y; x++)  {  if (checkGasLimit(y, cur\_block\_no))  {  \_;  }  else revert();  } | This rule checks that the approximate gas limit to execute this loop is safe or not. For this our tool pre-calculates the gas fee for each basic block. |
| if (isCheck)  {  a = x[5] +(Bop) y;  }  else  {  break;  } | | if (isCheck)  {  if (checkIntAdd(x[5],y))  a = IntAdd(x[5],y);  else revert();  }  else  {  Record(cur\_block\_no);  break;  } |  |
| a = x.call() +(Bop) \_; | | tmp = x.call(); //return int256  if (CheckIntAdd (tmp, \_))  {  a = IntAdd(x, \_);  }  else revert(); | This rule separates the assignment statement to external call() and some operations. |
| a = x.call(y.call()) +(Bop) z; | | tmp = x.call(y.call());  if (CheckIntAdd (tmp, z))  a = IntAdd(tmp, z);  else revert(); |  |