Bug Injection on Smart Contracts

Zhou Yu

Beijing University of Posts and Telecommunications

Reference

How Effective are Smart Contract Analysis Tools?
Evaluating Smart Contract Static Analysis Tools Using Bug Injection

https://arxiv.org/abs/2005.11613

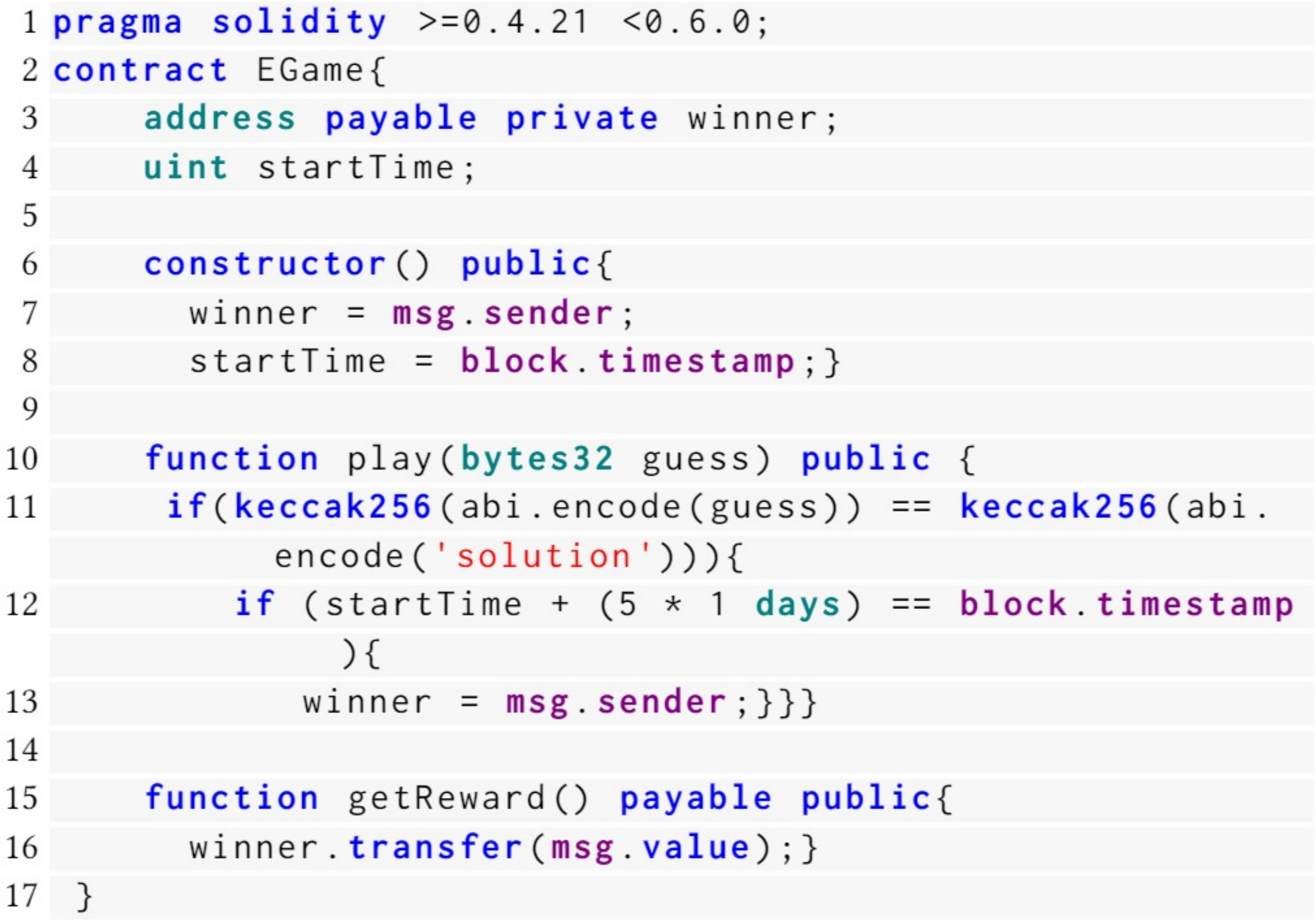
LAVA: Large-scale Automated Vulnerability Addition

https://ieeexplore.ieee.org/abstract/document/7546498

How Effective are Smart Contract Analysis Tools? Evaluating Smart Contract Static Analysis Tools Using Bug Injection

Smart Contracts

- receive and execute transactions autonomously
- immutable
- irreversible



```
pragma solidity >=0.4.21 <0.6.0;
contract EGame{
    address payable private winner;
    uint startTime;

    constructor() public{
      winner = msg.sender;
      startTime = block.timestamp;}

    function play(bytes32 guess) public {
     if(keccak256(abi.encode(guess)) == keccak256(abi.
          encode('solution'))){
        if (startTime + (5 * 1 days) == block.timestamp
           ){
          winner = msg.sender;}}}

    function getReward() payable public{
      winner.transfer(msg.value);}
}
```

A contract written in Solidity.

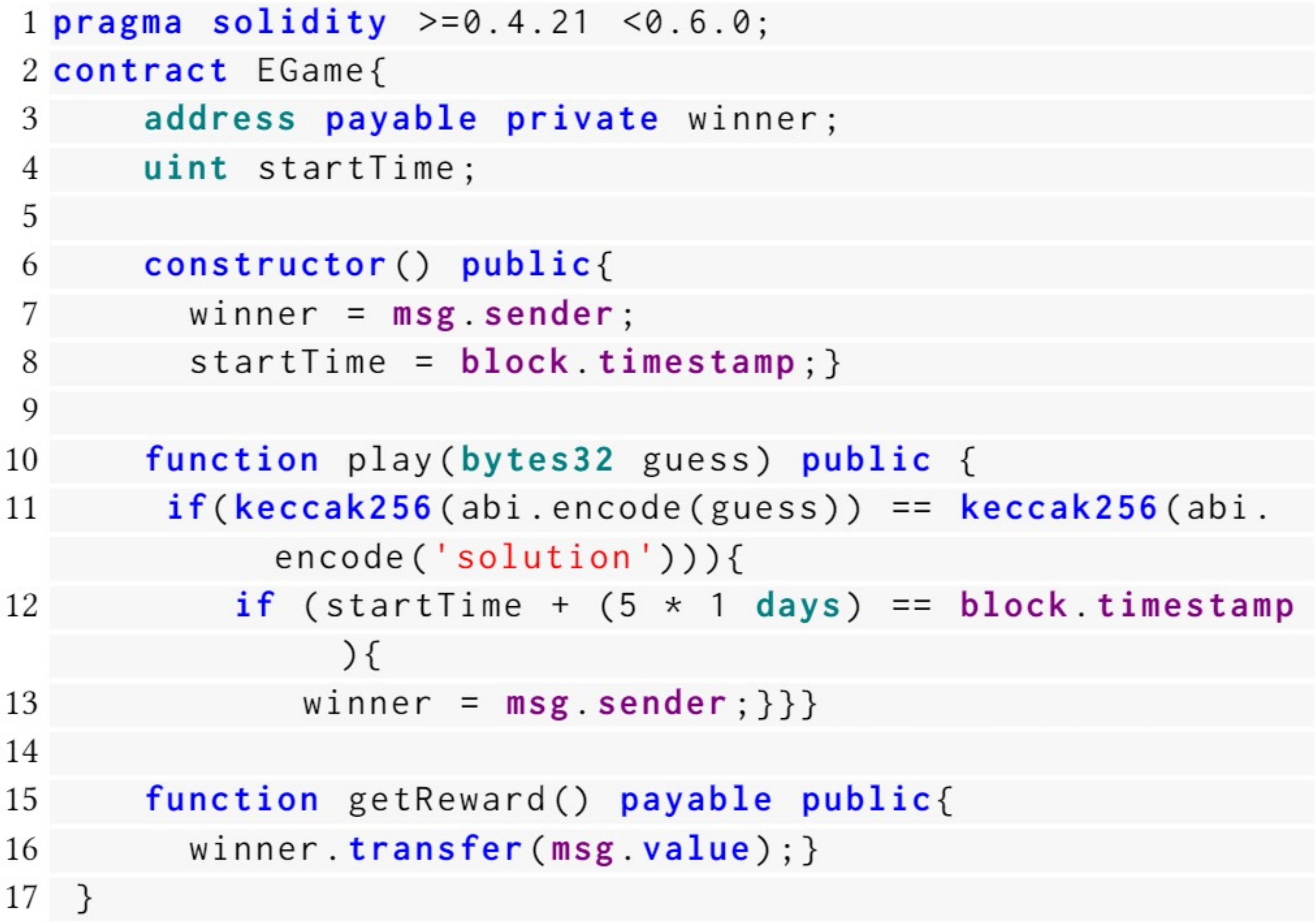
Static Analysis Tools

- Symbolic Execution
  - Oyente, Securify, Mythril, Manticore
- Pattern matching
  - SmartCheck
- Static Single Assignment
  - Slither

- False-negatives (FN)
- False-positives (FP)

Challenges

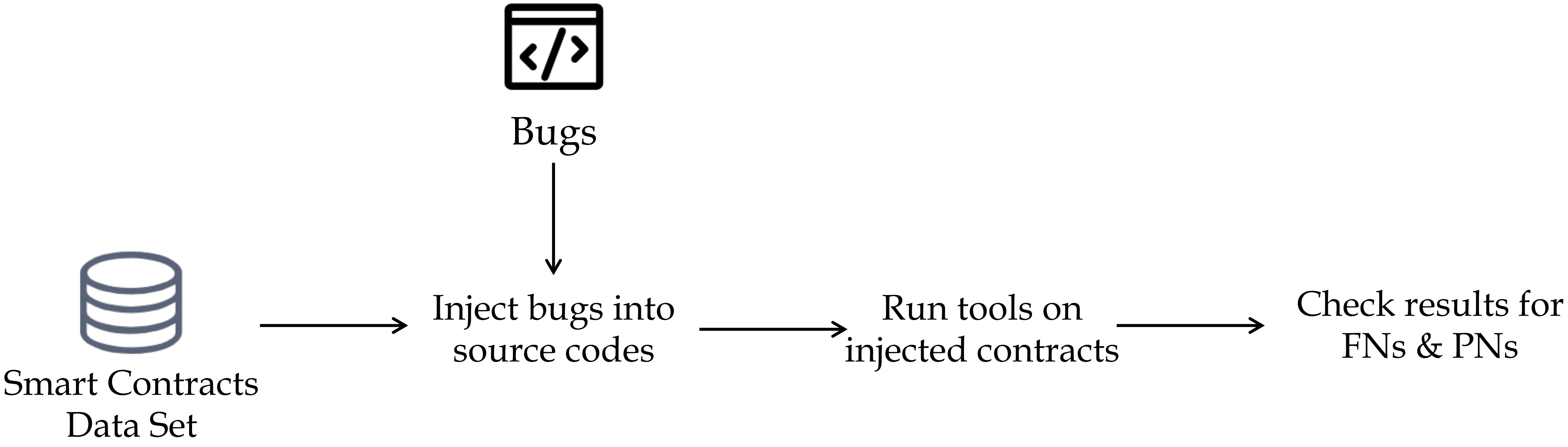
- Bug injection locations
- Semantics dependency



```
pragma solidity >=0.4.21 <0.6.0;
contract EGame{
    address payable private winner;
    uint startTime;

    constructor() public{
      winner = msg.sender;
      startTime = block.timestamp;}

    function play(bytes32 guess) public {
     if(keccak256(abi.encode(guess)) == keccak256(abi.
          encode('solution'))){
        if (startTime + (5 * 1 days) == block.timestamp
             ){
           winner = msg.sender;}}}

    function getReward() payable public{
      winner.transfer(msg.value);}
 }
```

Bug Injection

- Security Bug → Code Snippet



Bugs

Smart Contracts Data Set

Inject bugs into source codes

Run tools on injected contracts

Check results for FNs & PNs

SolidiFI: Work Flow

* F+M: number of functions and function modifiers

Datasets

Representative 50 smart contracts

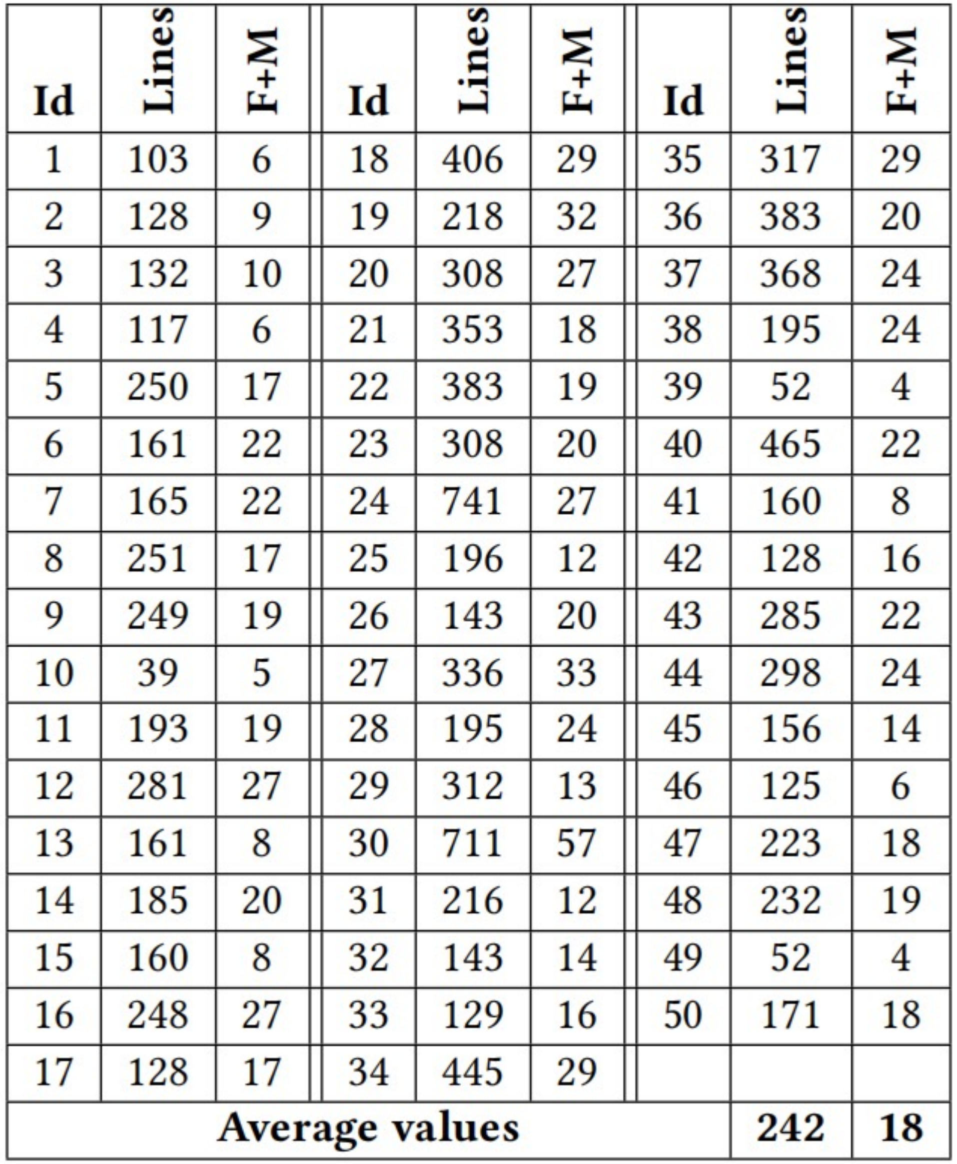
- code size
- compatibility
- functionality

6 types of security bugs

Inject one bug type at a time

9,369 distinct bugs

- data flow/control flow
- design pattern

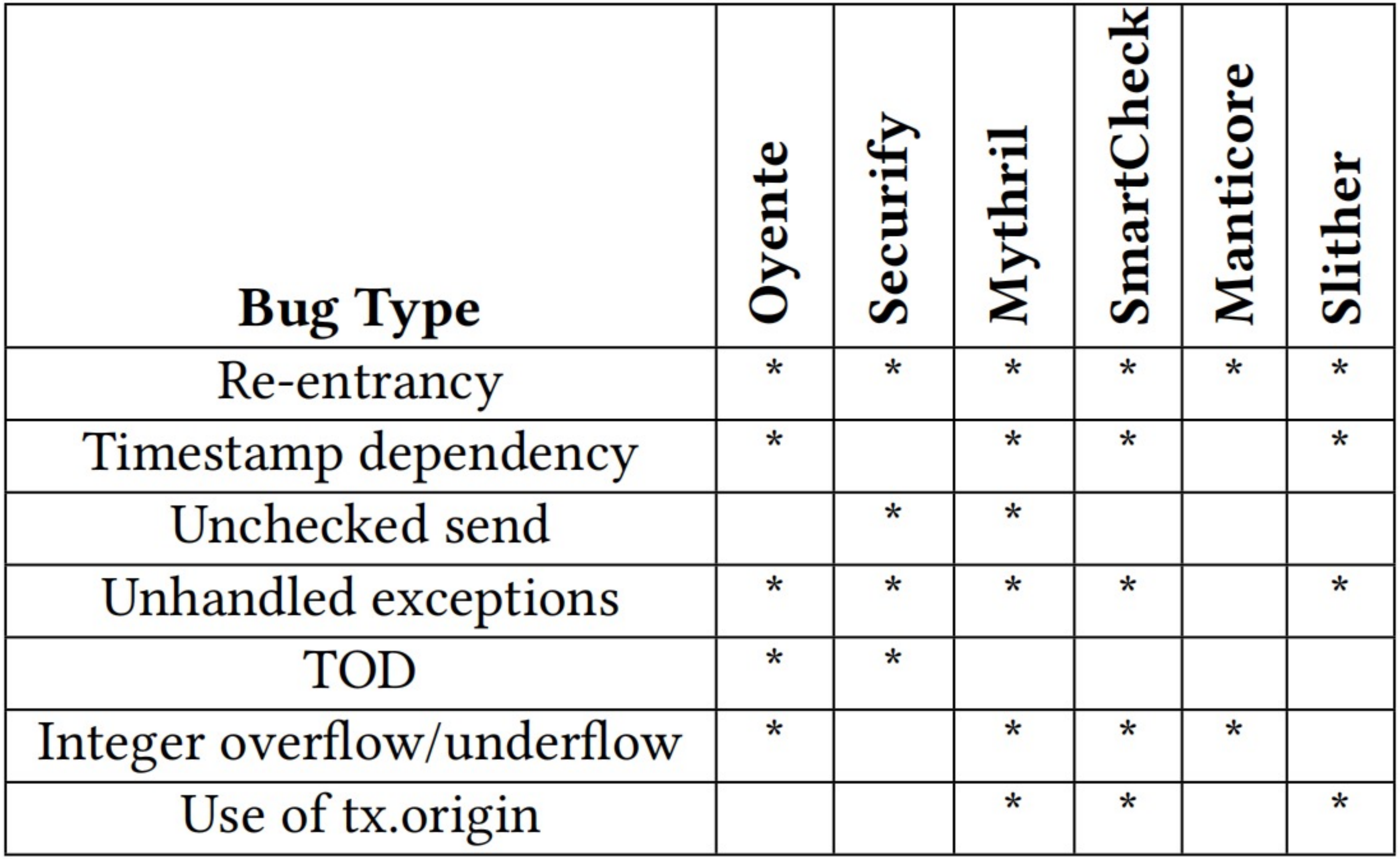


Id	Lines	F+M	Id	Lines	F+M	Id	Lines	F+M
1	103	6	18	406	29	35	317	29
2	128	9	19	218	32	36	383	20
3	132	10	20	308	27	37	368	24
4	117	6	21	353	18	38	195	24
5	250	17	22	383	19	39	52	4
6	161	22	23	308	20	40	465	22
7	165	22	24	741	27	41	160	8
8	251	17	25	196	12	42	128	16
9	249	19	26	143	20	43	285	22
10	39	5	27	336	33	44	298	24
11	193	19	28	195	24	45	156	14
12	281	27	29	312	13	46	125	6
13	161	8	30	711	57	47	223	18
14	185	20	31	216	12	48	232	19
15	160	8	32	143	14	49	52	4
16	248	27	33	129	16	50	171	18
17	128	17	34	445	29			
Average values							**242**	**18**

Bug Injection

BUG TYPE	Full Code Snippet	Code Transformation	Weakening Security Mechanisms
Reentrancy	√		
Timestamp dependency	√		
Unhandled Send	√		
Unhandled exceptions	√		√
Transaction ordering dependency (TOD)	√		
Integer overflow/underflow	√	√	
Use of *tx.origin*	√	√	

Bug Injection



Bug Type	Oyente	Securify	Mythril	SmartCheck	Manticore	Slither
Re-entrancy	*	*	*	*	*	*
Timestamp dependency	*		*	*		*
Unchecked send		*	*			
Unhandled exceptions	*	*	*	*		*
TOD	*	*				
Integer overflow/underflow	*		*	*	*	
Use of tx.origin			*	*		*

Algorithm

Preprocess: compling, AST generation.

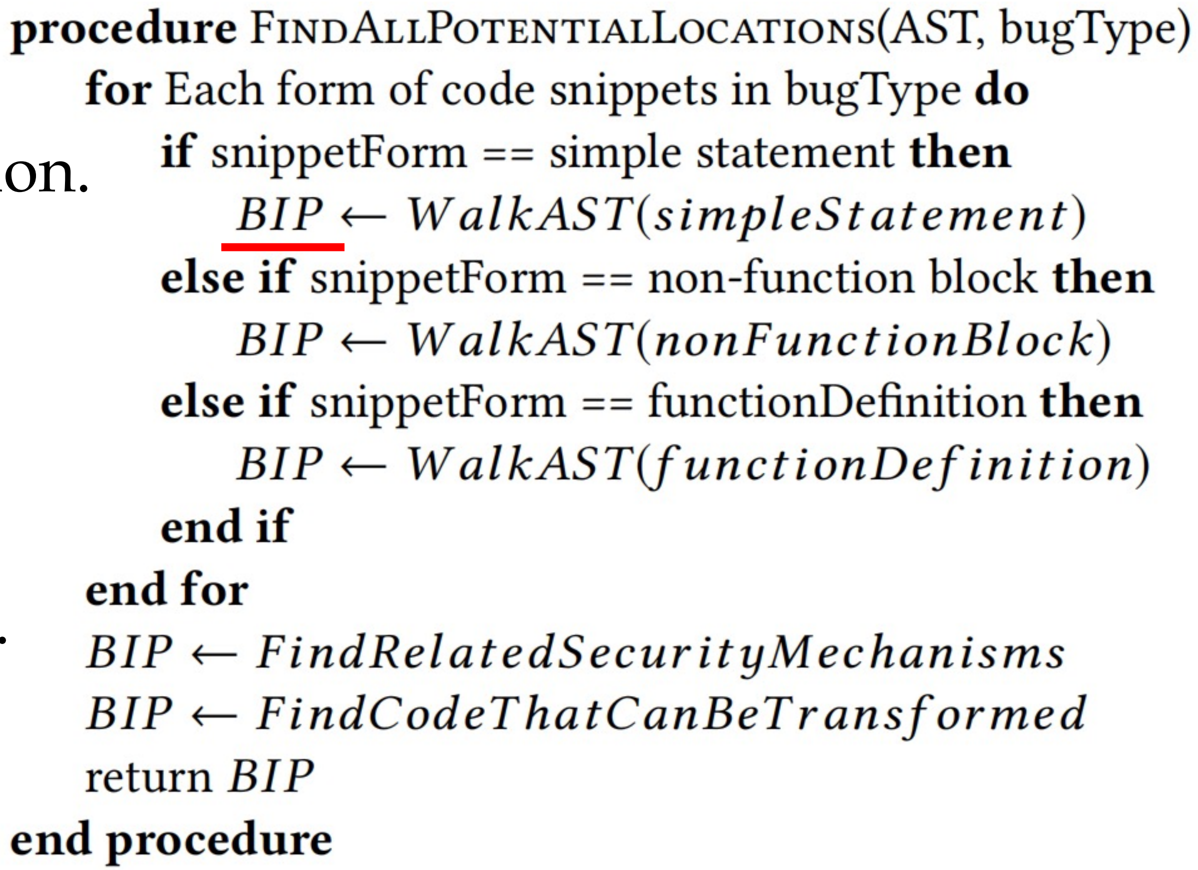
1. Annotatedabstract syntax tree generation.

   BIP: bug injection profile

   AST-based analysis

2. Bug injection into all marked locations.

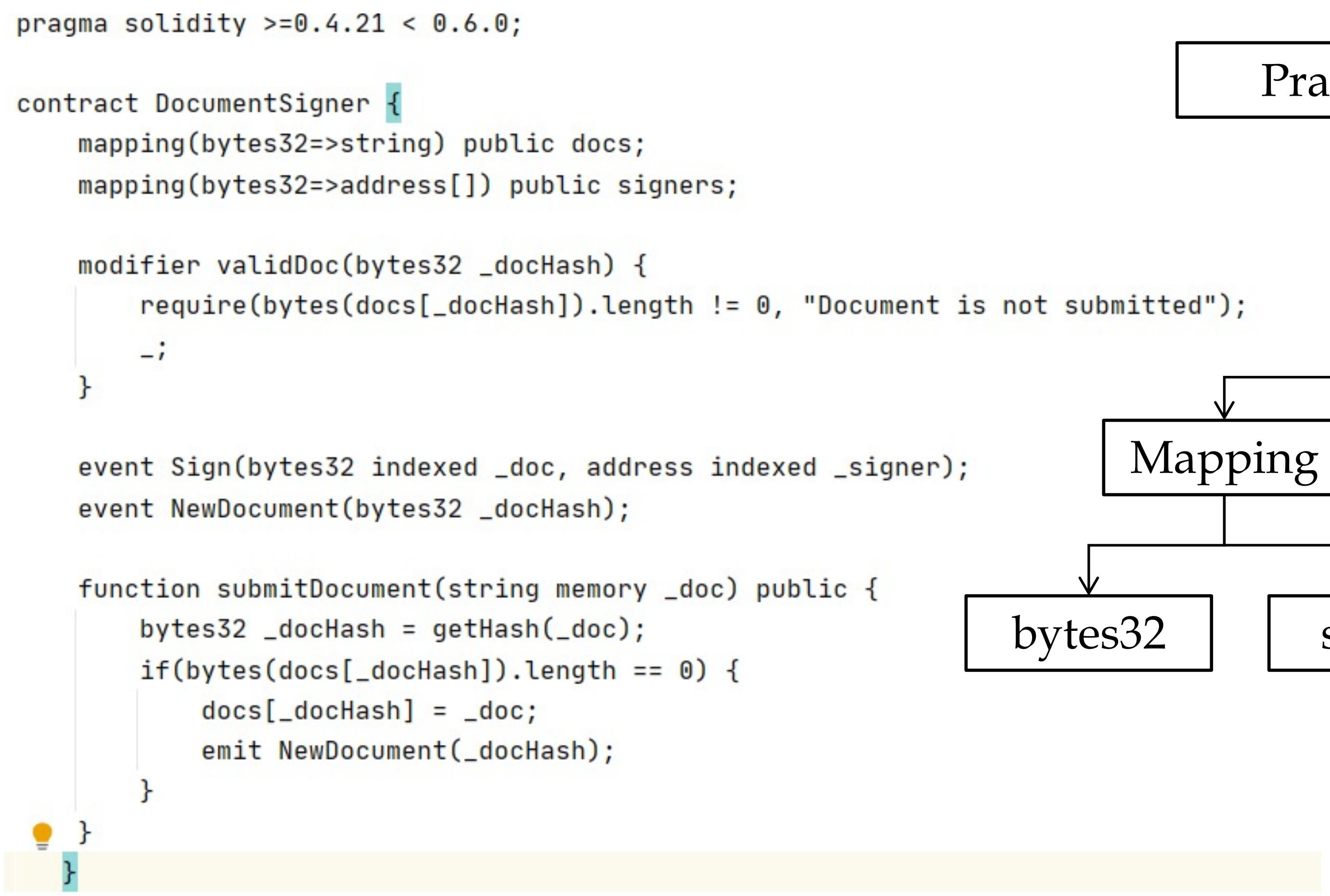
3. Evaluation on static analysis tools.



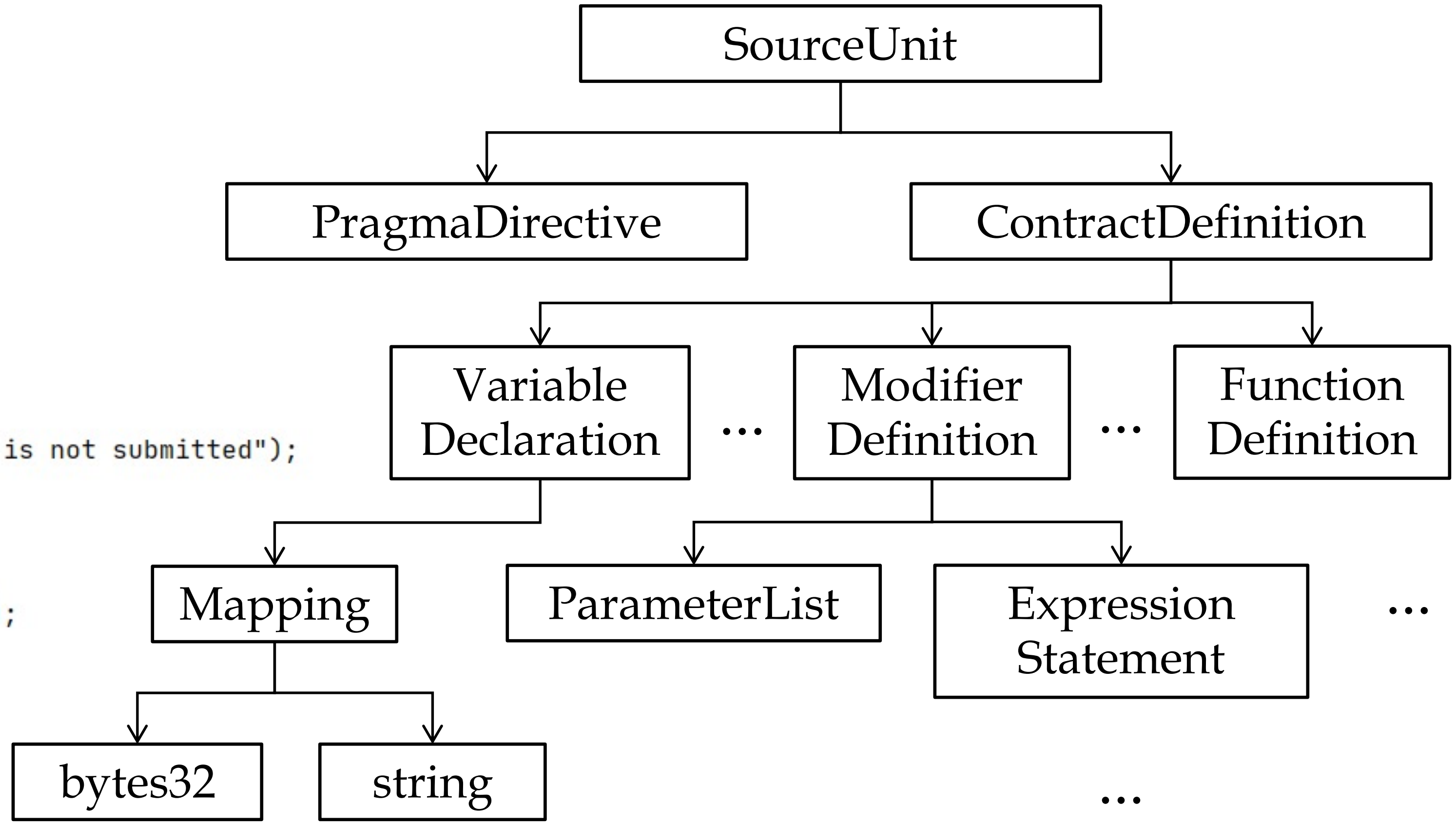
```
procedure FINDALLPOTENTIALLOCATIONS(AST, bugType)
    for Each form of code snippets in bugType do
        if snippetForm == simple statement then
            $BIP \leftarrow WalkAST(simpleStatement)$
        else if snippetForm == non-function block then
            $BIP \leftarrow WalkAST(nonFunctionBlock)$
        else if snippetForm == functionDefinition then
            $BIP \leftarrow WalkAST(functionDefinition)$
        end if
    end for
    $BIP \leftarrow FindRelatedSecurityMechanisms$
    $BIP \leftarrow FindCodeThatCanBeTransformed$
    return $BIP$
end procedure
```

* solc: https://github.com/ethereum/solidity

Algorithm



```
pragma solidity >=0.4.21 < 0.6.0;

contract DocumentSigner {
    mapping(bytes32=>string) public docs;
    mapping(bytes32=>address[]) public signers;

    modifier validDoc(bytes32 _docHash) {
        require(bytes(docs[_docHash]).length != 0, "Document is not submitted");
        _;
    }

    event Sign(bytes32 indexed _doc, address indexed _signer);
    event NewDocument(bytes32 _docHash);

    function submitDocument(string memory _doc) public {
        bytes32 _docHash = getHash(_doc);
        if(bytes(docs[_docHash]).length == 0) {
            docs[_docHash] = _doc;
            emit NewDocument(_docHash);
        }
    }
}
```



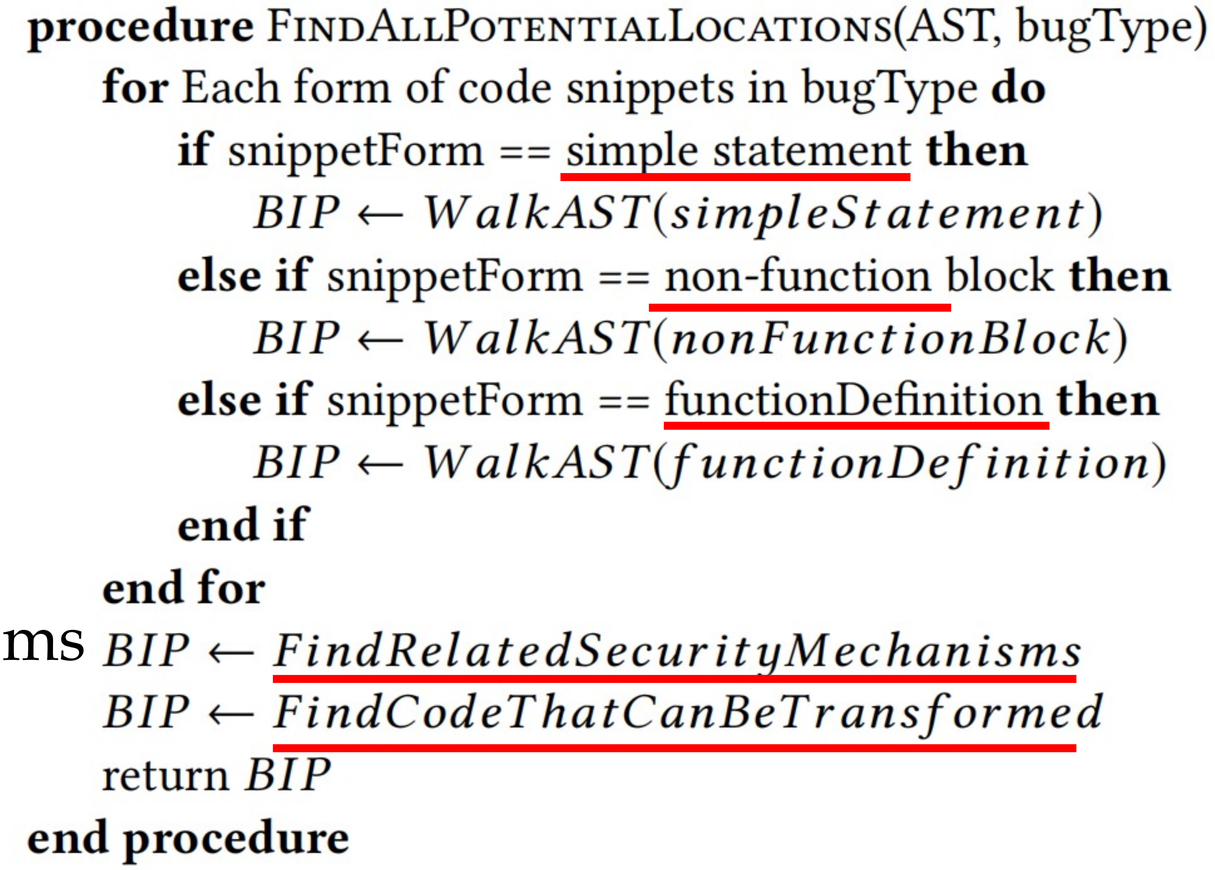
Source code → AST

Algorithm

1. Annotatedabstract syntax tree generation.

- Inject full code snippets

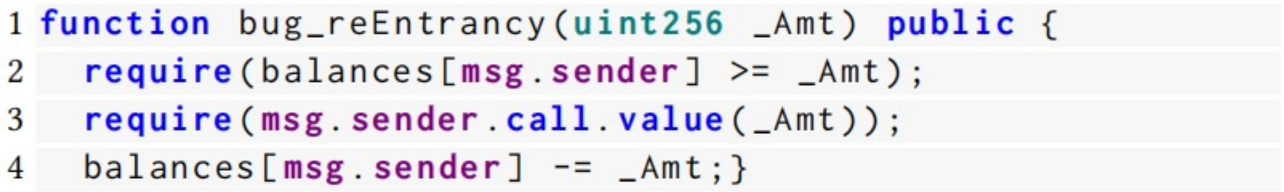
- Weakening security mechanisms
- Transfrom codes



```
procedure FINDALLPOTENTIALLOCATIONS(AST, bugType)
    for Each form of code snippets in bugType do
        if snippetForm == simple statement then
            $BIP \leftarrow WalkAST(simpleStatement)$
        else if snippetForm == non-function block then
            $BIP \leftarrow WalkAST(nonFunctionBlock)$
        else if snippetForm == functionDefinition then
            $BIP \leftarrow WalkAST(functionDefinition)$
        end if
    end for
    $BIP \leftarrow FindRelatedSecurityMechanisms$
    $BIP \leftarrow FindCodeThatCanBeTransformed$
    return $BIP$
end procedure
```

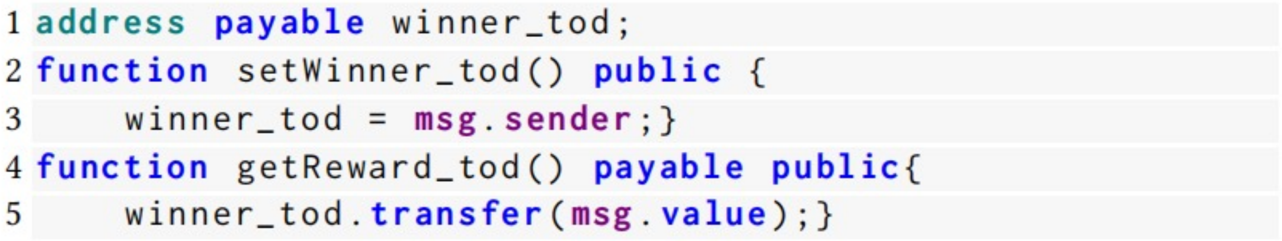
Bug Injection

- Full code snippet



```
function bug_reEntrancy(uint256 _Amt) public {
  require(balances[msg.sender] >= _Amt);
  require(msg.sender.call.value(_Amt));
  balances[msg.sender] -= _Amt;}
```

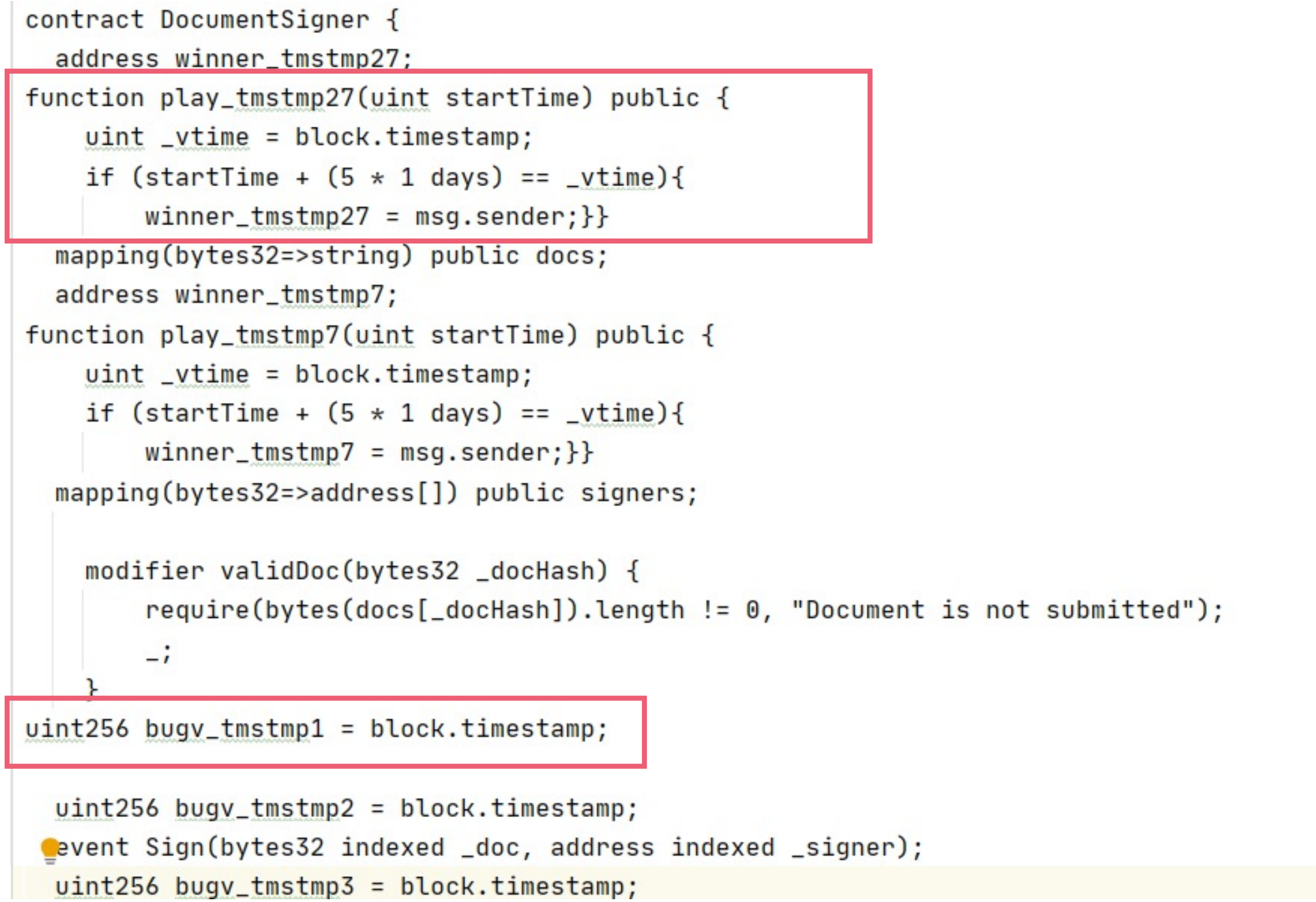
Re-entrancy example



```
address payable winner_tod;
function setWinner_tod() public {
    winner_tod = msg.sender;}
function getReward_tod() payable public{
    winner_tod.transfer(msg.value);}
```

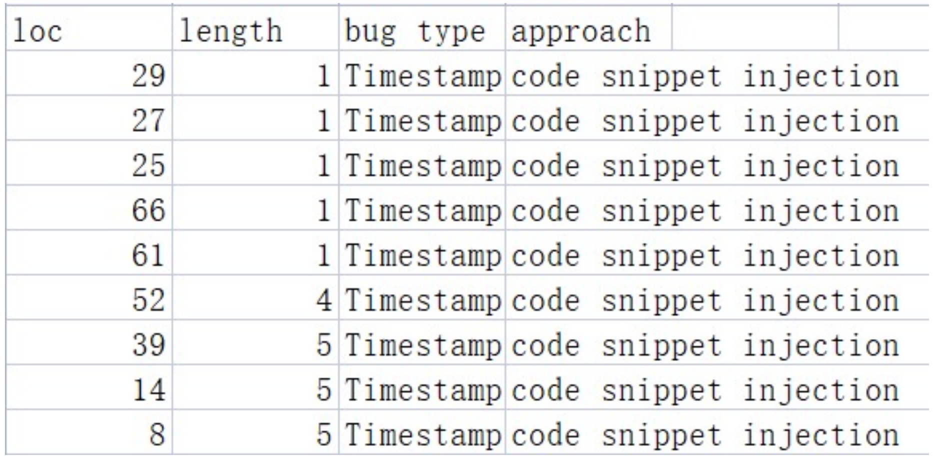
Transaction ordering dependency example

Algorithm



```
contract DocumentSigner {
  address winner_tmstmp27;
function play_tmstmp27(uint startTime) public {
    uint _vtime = block.timestamp;
    if (startTime + (5 * 1 days) == _vtime){
        winner_tmstmp27 = msg.sender;}}
  mapping(bytes32=>string) public docs;
  address winner_tmstmp7;
function play_tmstmp7(uint startTime) public {
    uint _vtime = block.timestamp;
    if (startTime + (5 * 1 days) == _vtime){
        winner_tmstmp7 = msg.sender;}}
  mapping(bytes32=>address[]) public signers;

    modifier validDoc(bytes32 _docHash) {
        require(bytes(docs[_docHash]).length != 0, "Document is not submitted");
        _;
    }
uint256 bugv_tmstmp1 = block.timestamp;

  uint256 bugv_tmstmp2 = block.timestamp;
  event Sign(bytes32 indexed _doc, address indexed _signer);
  uint256 bugv_tmstmp3 = block.timestamp;
```

Buggy Contracts

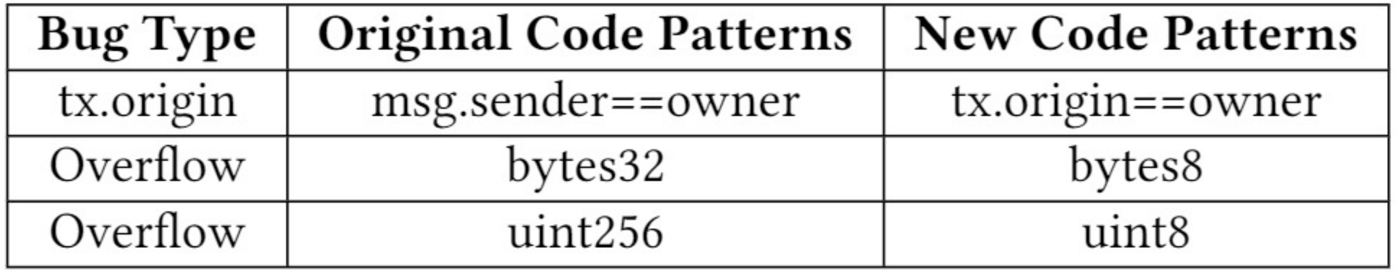


loc	length	bug type	approach
29	1	Timestamp	code snippet injection
27	1	Timestamp	code snippet injection
25	1	Timestamp	code snippet injection
66	1	Timestamp	code snippet injection
61	1	Timestamp	code snippet injection
52	4	Timestamp	code snippet injection
39	5	Timestamp	code snippet injection
14	5	Timestamp	code snippet injection
8	5	Timestamp	code snippet injection

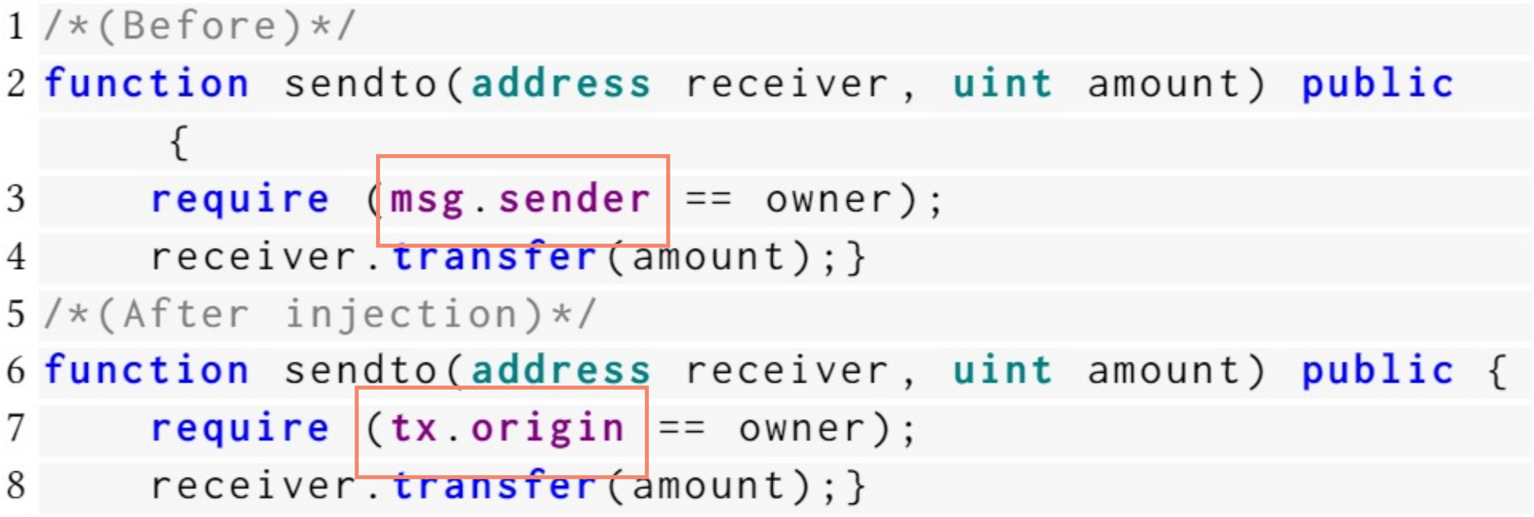
Corresponding Logs

Bug Injection

- Code transformation



Bug Type	Original Code Patterns	New Code Patterns
tx.origin	msg.sender==owner	tx.origin==owner
Overflow	bytes32	bytes8
Overflow	uint256	uint8

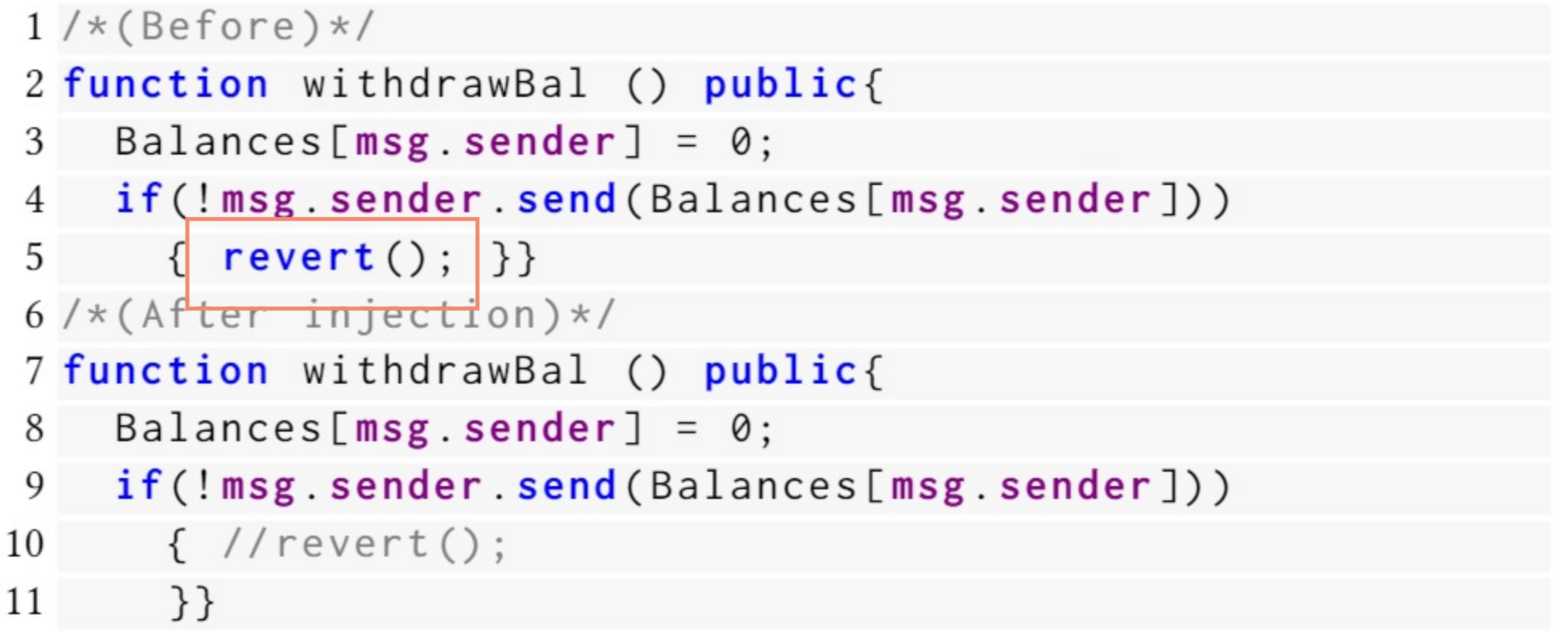


```
/*(Before)*/
function sendto(address receiver, uint amount) public {
    require (msg.sender == owner);
    receiver.transfer(amount);}
/*(After injection)*/
function sendto(address receiver, uint amount) public {
    require (tx.origin == owner);
    receiver.transfer(amount);}
```

Example: Use of tx.origin

Bug Injection

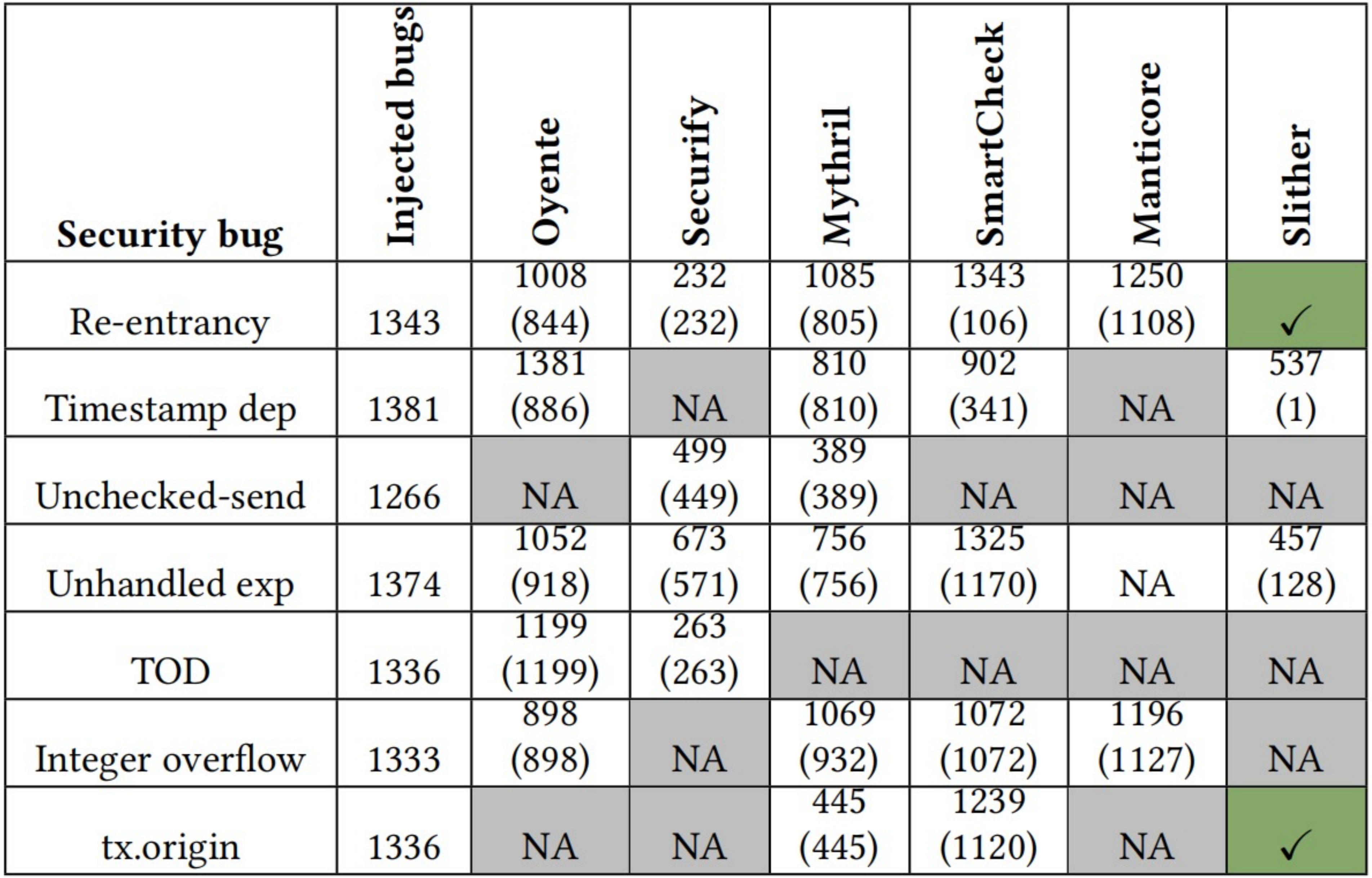
- Weakening security mechanisms



```
/*(Before)*/
function withdrawBal () public{
  Balances[msg.sender] = 0;
  if(!msg.sender.send(Balances[msg.sender]))
    { revert(); }}
/*(After injection)*/
function withdrawBal () public{
  Balances[msg.sender] = 0;
  if(!msg.sender.send(Balances[msg.sender]))
    { //revert();
    }}
```

Example: Unhandled exception

Evaluation on Tools: False-negatives

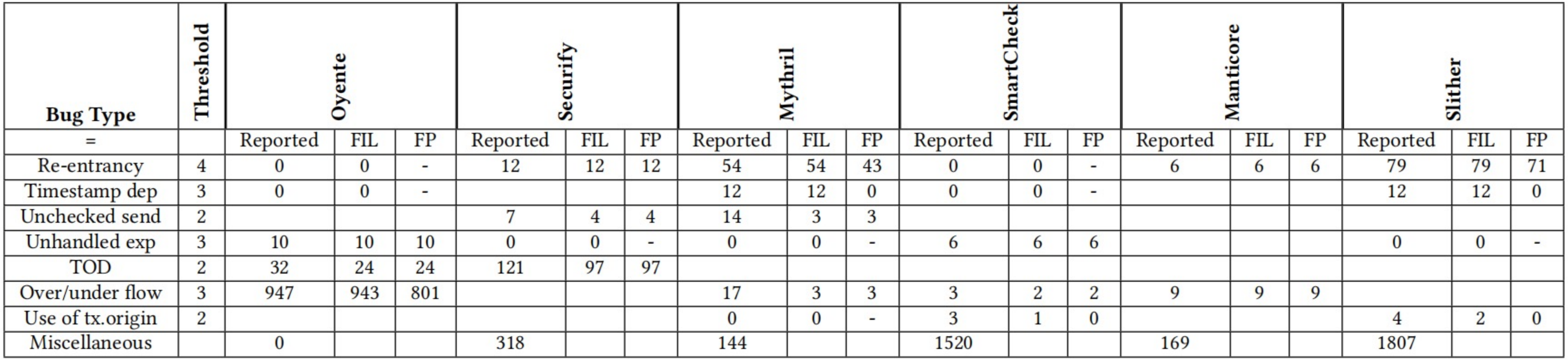


Security bug	Injected bugs	Oyente	Securify	Mythril	SmartCheck	Manticore	Slither
Re-entrancy	1343	1008 (844)	232 (232)	1085 (805)	1343 (106)	1250 (1108)	✓
Timestamp dep	1381	1381 (886)	NA	810 (810)	902 (341)	NA	537 (1)
Unchecked-send	1266	NA	499 (449)	389 (389)	NA	NA	NA
Unhandled exp	1374	1052 (918)	673 (571)	756 (756)	1325 (1170)	NA	457 (128)
TOD	1336	1199 (1199)	263 (263)	NA	NA	NA	NA
Integer overflow	1333	898 (898)	NA	1069 (932)	1072 (1072)	1196 (1127)	NA
tx.origin	1336	NA	NA	445 (445)	1239 (1120)	NA	✓

Evaluation on Tools: False-positives

For each smart contract:
manually examine only those bugs that are not reported by the majority of the other tools.

For each tool:
randomly selected 20 bugs of each bug category that were not excluded by the majority approach.

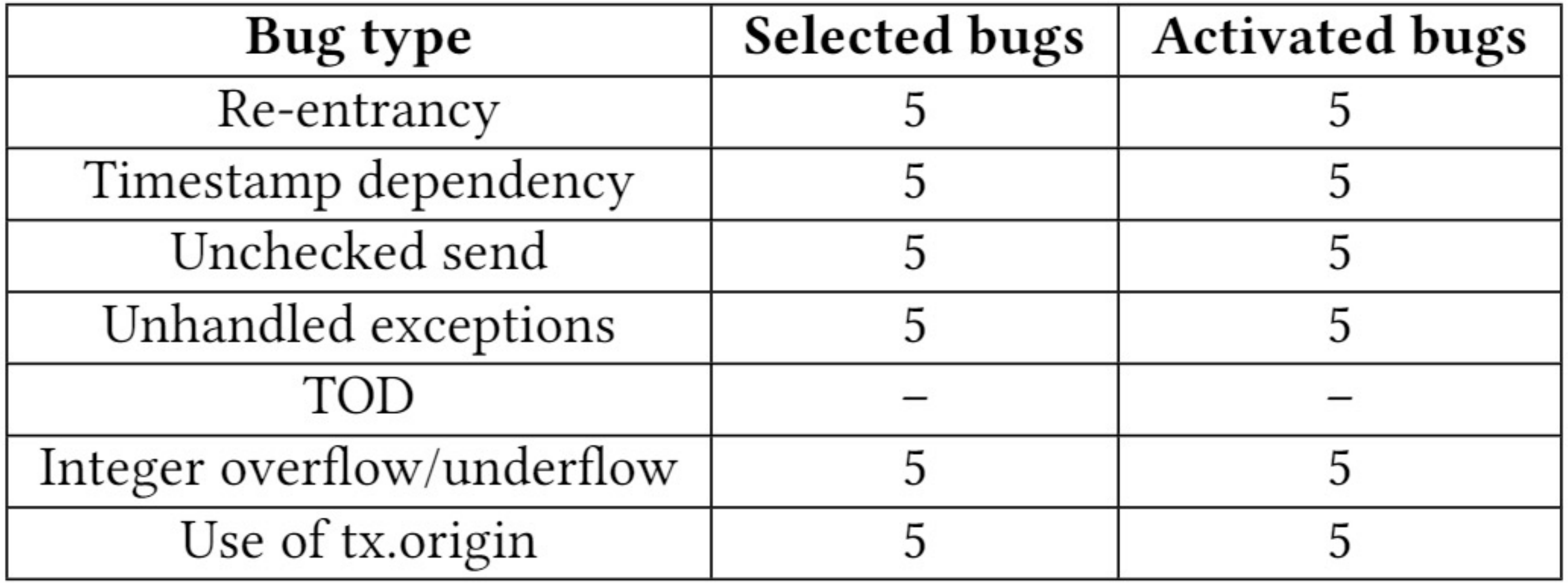


Bug Type	Threshold	Oyente			Securify			Mythril			SmartCheck			Manticore			Slither		
=		Reported	FIL	FP	Reported	FIL	FP	Reported	FIL	FP	Reported	FIL	FP	Reported	FIL	FP	Reported	FIL	FP
Re-entrancy	4	0	0	-	12	12	12	54	54	43	0	0	-	6	6	6	79	79	71
Timestamp dep	3	0	0	-				12	12	0	0	0	-				12	12	0
Unchecked send	2				7	4	4	14	3	3									
Unhandled exp	3	10	10	10	0	0	-	0	0	-	6	6	6				0	0	-
TOD	2	32	24	24	121	97	97												
Over/under flow	3	947	943	801				17	3	3	3	2	2	9	9	9			
Use of tx.origin	2							0	0	-	3	1	0				4	2	0
Miscellaneous		0			318			144			1520			169			1807		

INFURA. https://infura.io
MetaMask. https://metamask.io

Exploitability

- Run buggy contracts on Ethereum nodes



Bug type	Selected bugs	Activated bugs
Re-entrancy	5	5
Timestamp dependency	5	5
Unchecked send	5	5
Unhandled exceptions	5	5
TOD	–	–
Integer overflow/underflow	5	5
Use of tx.origin	5	5

Experiment results

* "–": not performed considering the cost

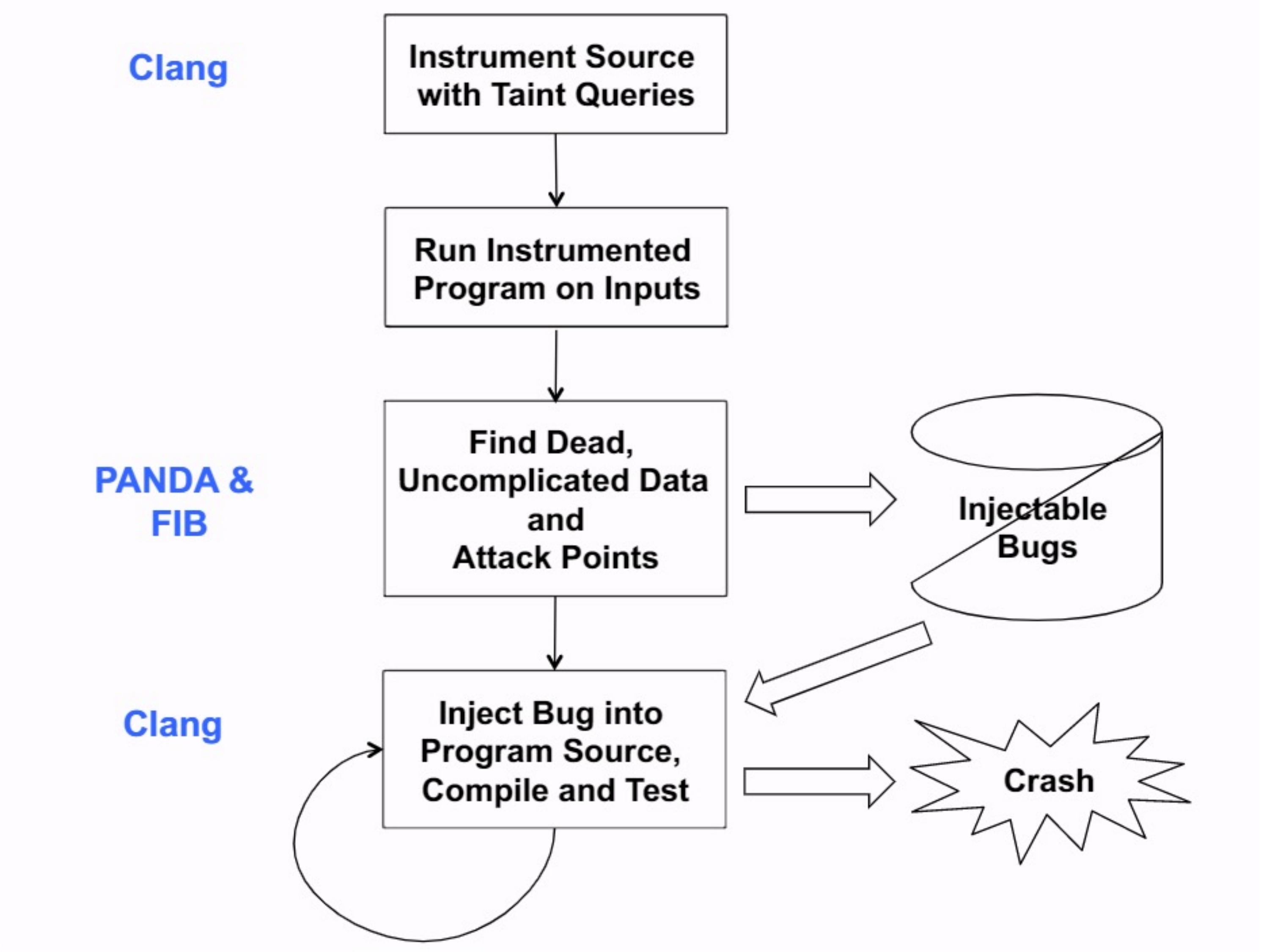
LAVA: Large-scale Automated Vulnerability Addition

Vulnerability corpora sources

SOURCE	**COST**	**REALISM**	**YIELD**
Accident	**Free**	**High**	**Tiny**
Search	**$$$**	**Meg-high**	**Low**
Injection	**$$**	**Med**	**Low-med**
Synthesis	*$*	**Low**	*High*

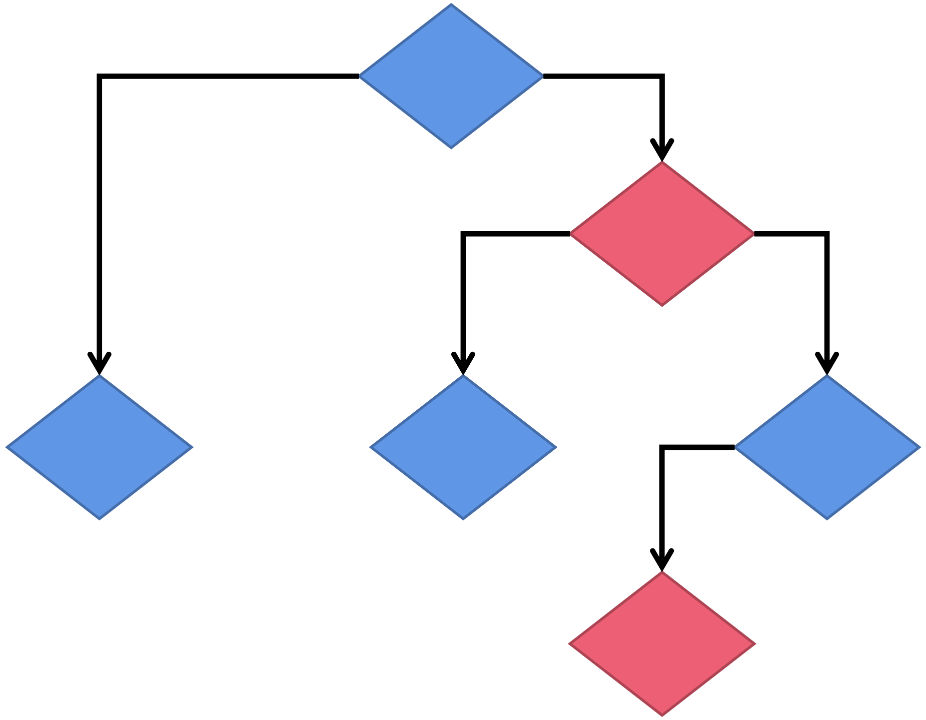
LAVA architecture

* PANDA: dynamic analysis platform


Clang
Instrument Source with Taint Queries
Run Instrumented Program on Inputs
PANDA & FIB
Find Dead, Uncomplicated Data and Attack Points
Injectable Bugs
Clang
Inject Bug into Program Source, Compile and Test
Crash

Taint-based measures

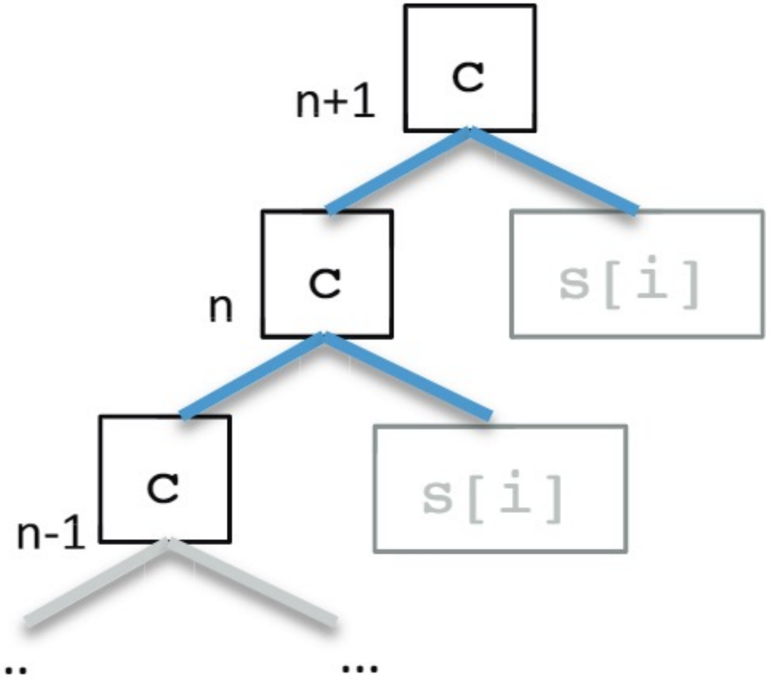
DUA: Dead, Uncomplicated and Available data.



Liveness

combine with

DUA

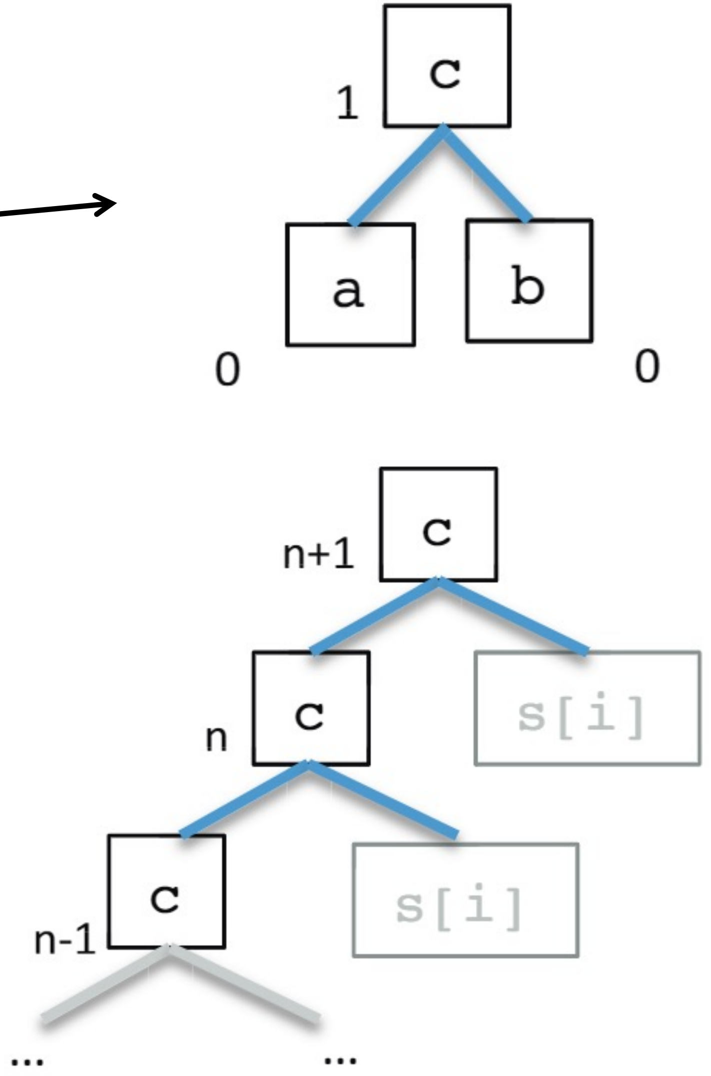


Taint Compute Number

Taint-based measures: TCN

```
void foo(int a, int b, char *s, char *d, int n) {
     int c = a+b;
     if (a != 0xdeadbeef)
          return;
     for (int i=0; i<n; i++)
          c+=s[i];
     memcpy(d,s,n+c); // Original source
     // BUG: memcpy(d+(b==0x6c617661)*b,s,n+c);
}
```

Source code



Taint Compute Number(c)

TCN: the depth of the tree of computation required to obtain a quantity from input byte

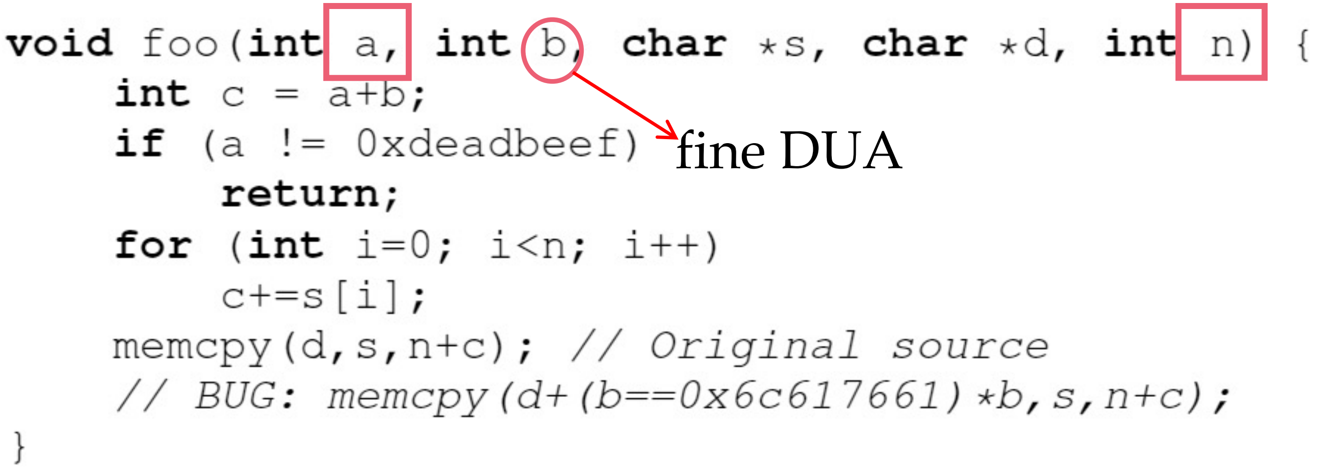
If TCN is 0, the quantity is a direct copy of input bytes

DUA: computationally close to the input

Taint-based measures: liveness

liveness: 1

liveness: n

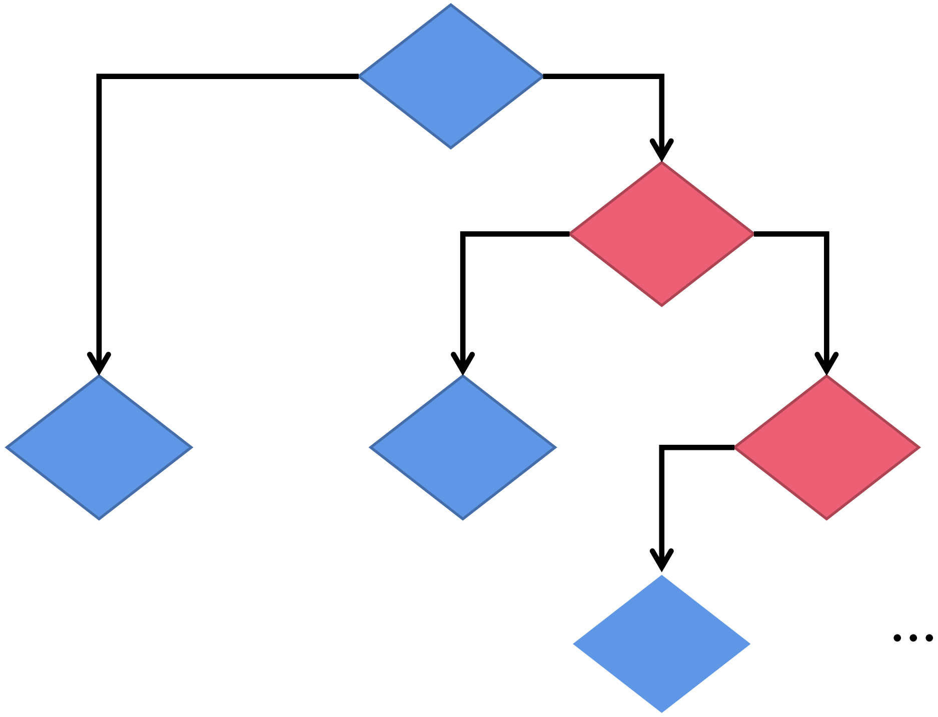


```
void foo(int a, int b, char *s, char *d, int n) {
    int c = a+b;
    if (a != 0xdeadbeef)
        return;
    for (int i=0; i<n; i++)
        c+=s[i];
    memcpy(d,s,n+c); // Original source
    // BUG: memcpy(d+(b==0x6c617661)*b,s,n+c);
}
```

fine DUA

Liveness: number of branches a byte in the input has been used to decide

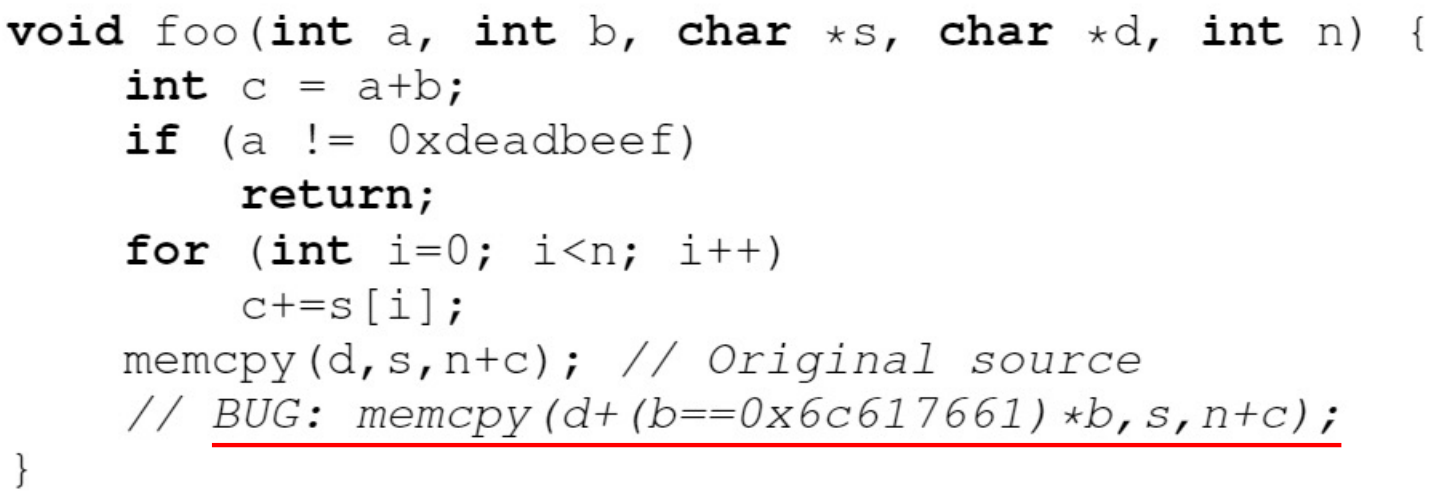
If a particular input byte label was never found in a taint label set associated with any byte used to decide a branch, it will have liveness of 0.



Liveness(n)

Attack point selection

- Make use of DUA to inject a bug
- Temporally after an appearance of a DUA



```
void foo(int a, int b, char *s, char *d, int n) {
    int c = a+b;
    if (a != 0xdeadbeef)
        return;
    for (int i=0; i<n; i++)
        c+=s[i];
    memcpy(d,s,n+c); // Original source
    // BUG: memcpy(d+(b==0x6c617661)*b,s,n+c);
}
```

Running example

Data-flow bug injection

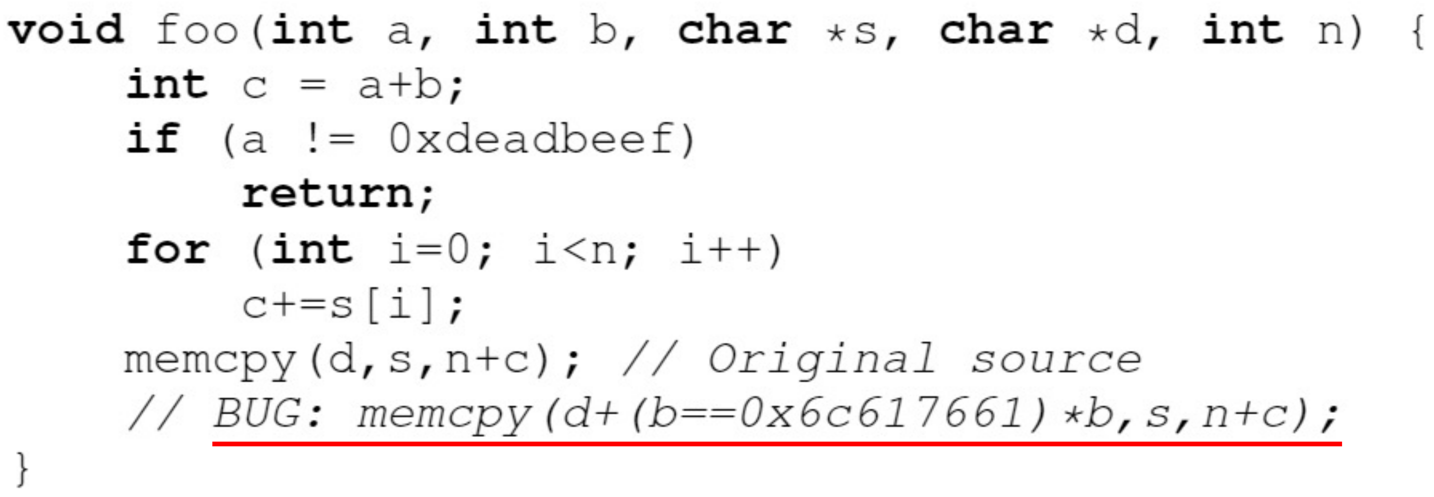
- Introduce a dataflow relationship between DUA and attack point.

One specific input

In this example,
out of bounds write
is trigerred only when
bytes 4..7 of the input
exactly match 0x6c617661.

If statement

...



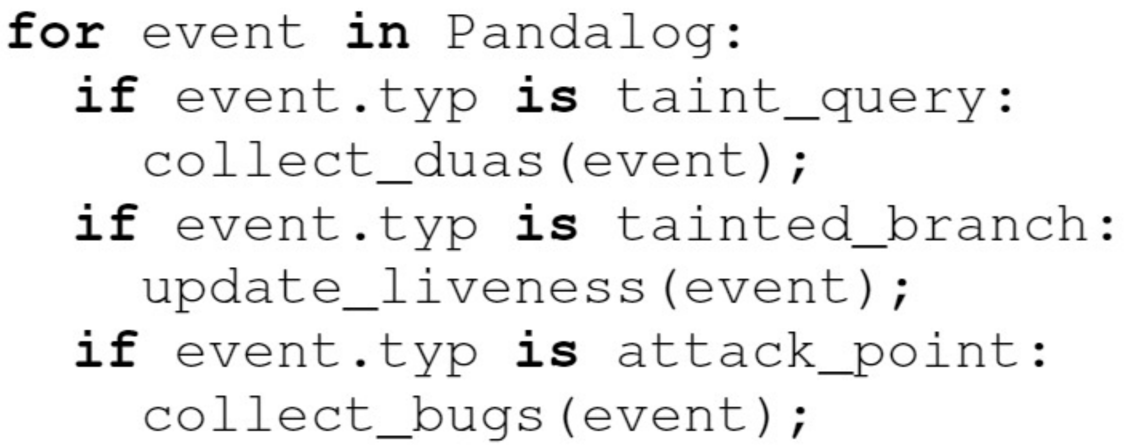
```
void foo(int a, int b, char *s, char *d, int n) {
    int c = a+b;
    if (a != 0xdeadbeef)
        return;
    for (int i=0; i<n; i++)
        c+=s[i];
    memcpy(d,s,n+c); // Original source
    // BUG: memcpy(d+(b==0x6c617661)*b,s,n+c);
}
```

Running example.

Finding DUA/attack point pairs

- Taint queries generation
- Run programs with a variety of inputs
- Choose inputs to maximize code coverage

- Mining pandalog
- Find injectable bugs

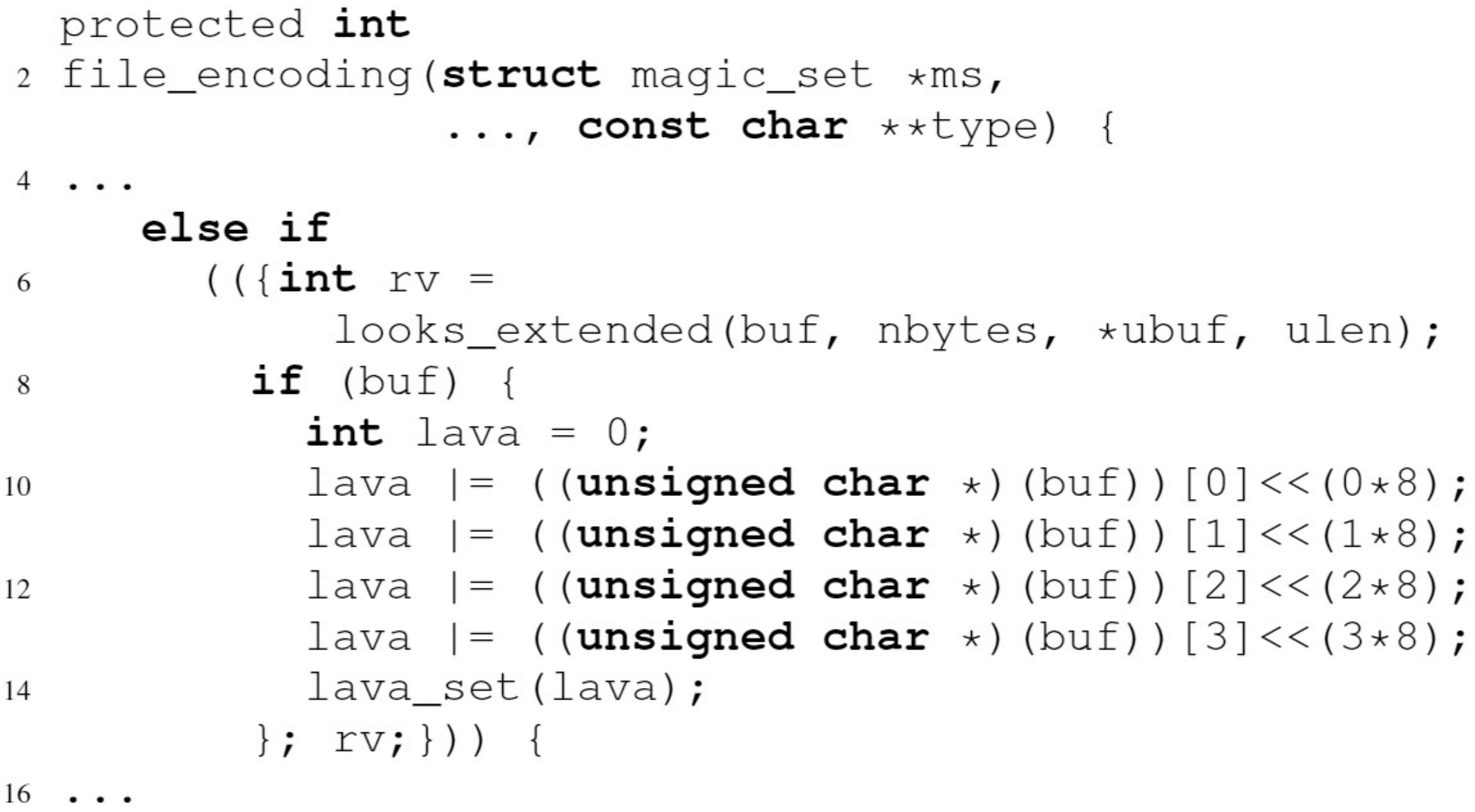


```
for event in Pandalog:
  if event.typ is taint_query:
    collect_duas(event);
  if event.typ is tainted_branch:
    update_liveness(event);
  if event.typ is attack_point:
    collect_bugs(event);
```

Find injectable bugs

Injecting the bugs

- For each DUA/ATP pair: generate the C code which uses the DUA to trigger the bug



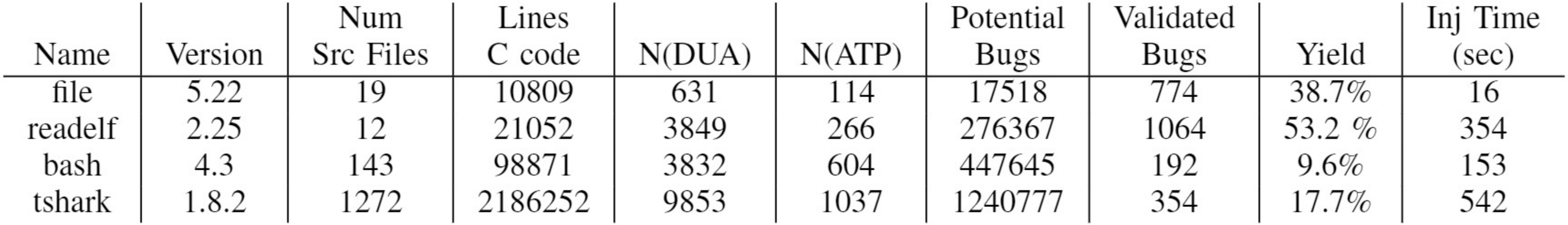
```
protected int
file_encoding(struct magic_set *ms,
              ..., const char **type) {
...
   else if
     (({int rv =
           looks_extended(buf, nbytes, *ubuf, ulen);
       if (buf) {
         int lava = 0;
         lava |= ((unsigned char *)(buf))[0]<<(0*8);
         lava |= ((unsigned char *)(buf))[1]<<(1*8);
         lava |= ((unsigned char *)(buf))[2]<<(2*8);
         lava |= ((unsigned char *)(buf))[3]<<(3*8);
         lava_set(lava);
       }; rv;})) {
...
```

Injected bugs

Injecting effectiveness

- 4 open source programs
- Validated inject bugs: 10~50%
- 2,000+ bugs* injected

* particular (DUA,attackpoint) pair



Name	Version	Num Src Files	Lines C code	N(DUA)	N(ATP)	Potential Bugs	Validated Bugs	Yield	Inj Time (sec)
file	5.22	19	10809	631	114	17518	774	38.7%	16
readelf	2.25	12	21052	3849	266	276367	1064	53.2 %	354
bash	4.3	143	98871	3832	604	447645	192	9.6%	153
tshark	1.8.2	1272	2186252	9853	1037	1240777	354	17.7%	542

For each target, the author chooses 2,000 potential bugs at random to validate.