Panda: Security Analysis of Algorand Smart Contracts

Zhiyuan Sun[1,2], Xiapu Luo[1], Yinqian Zhang[2]

[1] The Hong Kong Polytechnic University,

[2] Southern University of Science and Technology

Introduction

What is Algorand?

Algorand is proposed to overcome the blockchain trilemma, or the three fundamental difficulties that blockchain system faces today: security, scalability, and decentralization by adopting a new consensus protocol. As a new permissionless blockchain system, Algorand uses Pure Proof-of-Stake (PPoS) consensus protocol based on Byzantine agreement protocol and is scalable to a number of users, enabling consensus to be reached with low latency.

Outline

1. Intro to Algorand smart contract
2. Vulnerabilities in Algorand smart contract
3. Automated detection tool
4. Evaluation results
5. Case study

Algorand Smart Contract

```python
def clear_state_program():
    return Approve()

def approval_program():
    on_c = Txn.on_completion()
    appID = Txn.application_id()
    on_update = Seq([
        ...
    ])
    on_delete = Reject()
        ... # define on_creation
        ... # define on_call
    program = Cond(
        [appID == Int(0), on_creation],
        [on_c == OnComplete.NoOp, on_call],
        [on_c == OnComplete.UpdateApplication, on_update],
        [on_c == OnComplete.DeleteApplication, on_delete],
        ...
        # handle OptIn and CloseOut
    )
    return program
```

Stateful smart contract (Application)

```python
def smart_signature():
    params_conds = And(
        Txn.type_enum() == TxnType.Payment,
        Txn.fee() == Int(1000),
        Txn.receiver() == Addr(" ... "),
        Txn.amount() == Int(10000)
    )
    safety_conds = And(
        Txn.close_remainder_to() == Global.zero_address(),
        Txn.rekey_to() == Global.zero_address()
    )
    recurring_conds = And( ... )
    program = And(params_conds, safety_conds,
    ↪ recurring_conds)
    return program
```

Smart Signature

Vulnerability discovery

We analyze the semantics of Algorand smart contracts and find 9 generic vulnerabilities which can be categorized into 5 types.

1. Unchecked Transaction Fee
2. Unchecked Transaction Parameters
3. Unexpected Delete and Update Operation
4. Unchecked Transaction Receiver
5. Local State Dependency

Vulnerability discovery

Unchecked Transaction Fee

On Algorand, the sender of the transaction pays the transaction fees. A user can also choose to increase fees to give the transaction a higher priority to be accepted by the blockchain. However, this feature may be exploited for launching attacks. If a smart signature is used as a signature account and does not restrict the transaction fees, then anyone can use this account to send a transaction with huge fees, and this transaction will wipe out all of its balance.

Unchecked Transaction Parameters

There are three important optional parameters in transactions: CloseRemainderTo, CloseAssetTo and RekeyTo. The format of these parameters is the Algorand address. If one of these parameters is set, the transaction will perform some crucial operations.

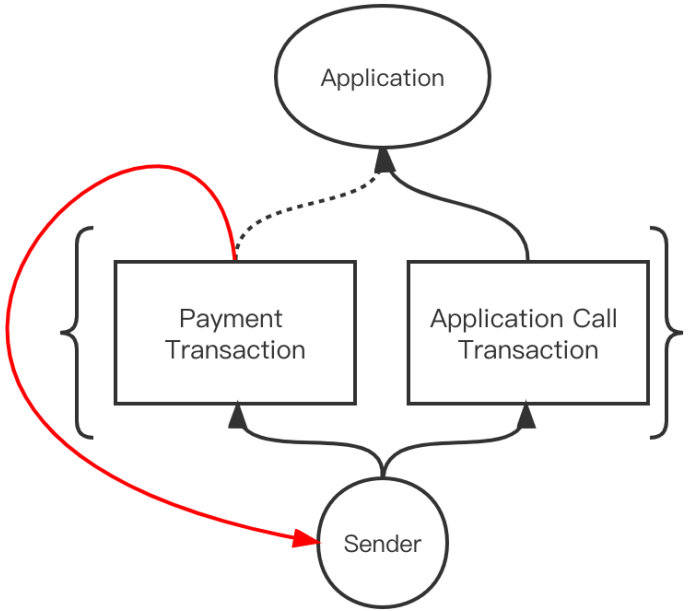
Vulnerability discovery

Unexpected Delete and Update Operation

If an attacker initiates an application update transaction (OnComplete equals to UpdateApplication) and attaches a malicious application in this transaction, then the current application will be replaced by the malicious one after the transaction is recorded in the blockchain.

Note that anyone can send application update transactions and application delete transactions, and whether the transaction is approved depends on the program logic. For example, the program can only allow the application creator to modify the application by comparing the transaction sender's address and the application creator's address. However, things may not always go well, and bad program logic (e.g., a programming mistake) may allow anyone to delete or update applications.

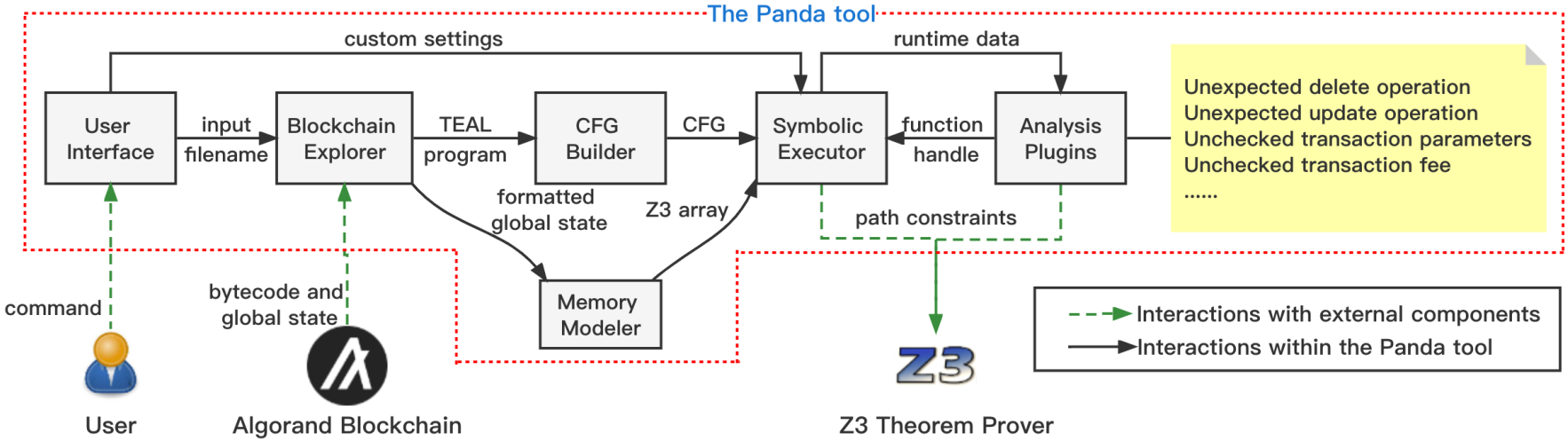
Vulnerability discovery



Unchecked Transaction Receiver

If a smart contract does not check the transaction receiver of the payment transaction or the asset transfer transaction, an attacker can specify the receiver as himself to break the program logic.

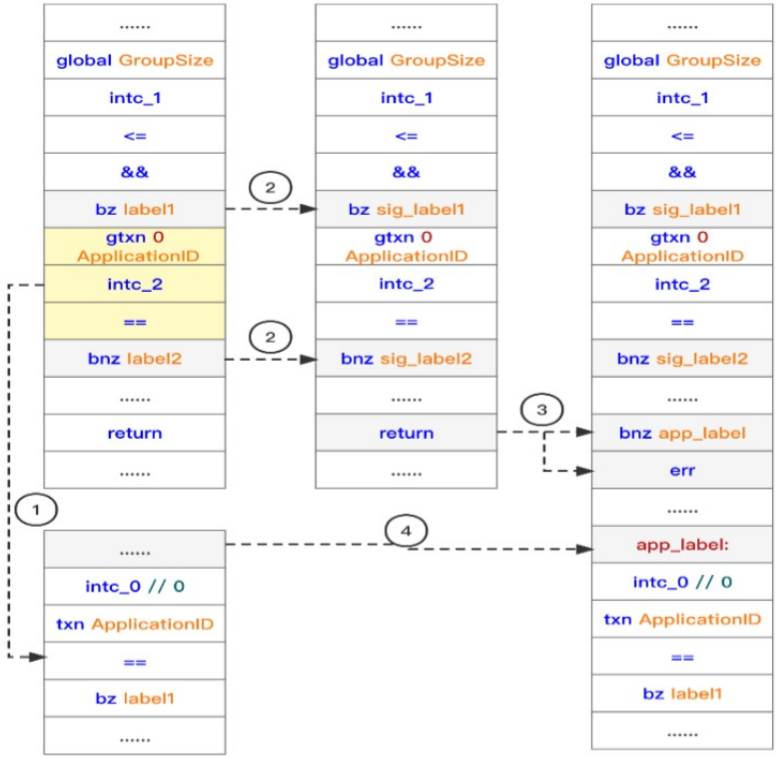
The Panda Tool



The figure above depicts the workflow and architecture of Panda, which consists of six major components, i.e. User Interface, Blockchain Explorer, CFG Builder, Memory Modeler, Symbolic Executor and Analysis Plugins.

The project is open sourced at: https://github.com/Sun-C0ffee/Panda

Difficlities



Handling Smart Signatures with Validators

The left graph illustrates the merging process of a smart signature and a validator. It is consist of four steps.

1. Identify the application ID and fetch its bytecode from the blockchain.

2. Rename jump labels to avoid name conflicts.

3. Substitute return instructions to bnz instructions which jump to the application entry.

4. Concatenate the smart signature and the application into a new smart signature.

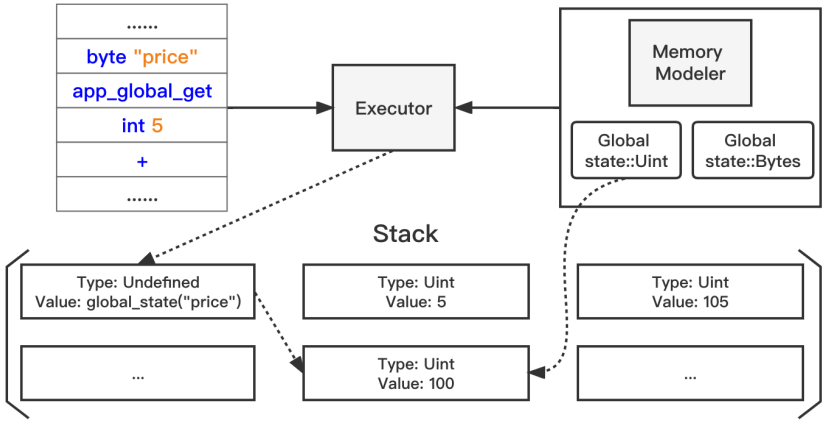
Difficlities

Since Algorand has two different data types (the Uint type and the Bytes type), we need to adopt some new techniques to deal with data type-related issues.

Runtime Type Checking
Most of the opcodes in Algorand distinguish the two data types explicitly. Thus, We use a Python dictionary to store data type and value separately.

Asynchronous Type Binding
To determine the type of the symbolic variables, we propose a new technique named asynchronous type binding. The right picture illustrates the specific process.

Recognizing Data Types



Detection Rules

To accurately express the vulnerability detection rules, we define the following 4 predicates:

P(constraints) is true if the path constraint set is solvable after adding the new path constraints.

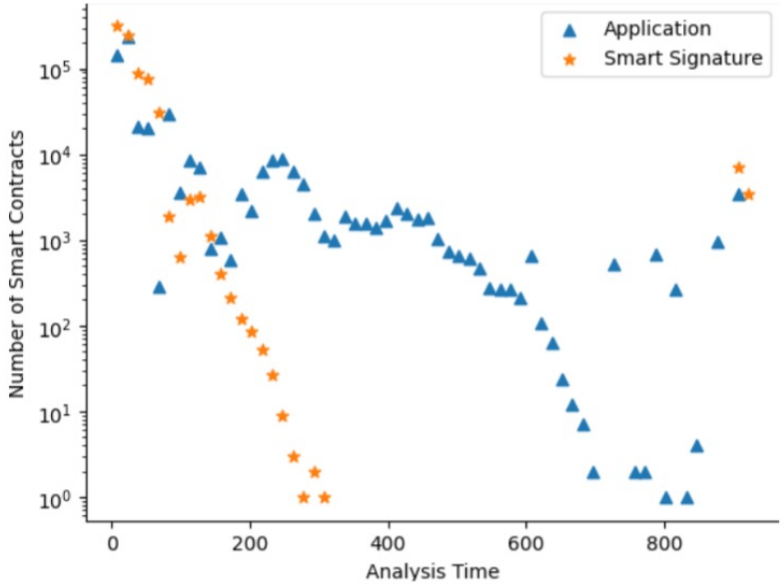
Q(variables) holds if none of the variables in the parameter set (i.e. variables) are contained in the current path constraint.

R(opcodes) holds if at least one opcode in the parameter set is used in the current execution trace.

I(txn, type) checks the implicit type of the transaction and returns true if the type of the transaction is the same as that specified in the second parameter

Index & Mode & Vulnerability Type	Detection Rules
(1) [SIG] Unchecked transaction fee	$^{1}\mathbf{Q}(\{currentTxn.Fee\}) \wedge (^{2}\mathbf{P}(\{GroupSize > \lvert groupTxns\rvert\}) \vee \exists txn \in groupTxns,$ $^{3}\mathbf{Q}(\{txn.Fee\}) \wedge {}^{4}\mathbf{P}(\{txn.Sender = LogicAddr, txn.Index = currentTxn.Index\}) \wedge$ $^{5}\bigwedge_{t \in groupTxns - \{txn\}} \mathbf{P}(\{t.Sender = RandomAddr \vee t.Sender = ZeroAddr\}))$
(2) [SIG] Unchecked RekeyTo	$^{1}Version >= 2 \wedge \mathbf{Q}(\{currentTxn.RekeyTo\}) \wedge$ $^{2}\mathbf{P}(\{currentTxn.CloseRemainderTo = ZeroAddr, currentTxn.AssetCloseTo = ZeroAddr\}) \wedge$ $(^{3}\mathbf{P}(\{GroupSize > \lvert groupTxns\rvert\}) \vee \exists txn \in groupTxns, {}^{4}\mathbf{Q}(\{txn.RekeyTo\}) \wedge$ $^{5}\mathbf{P}(\{txn.Sender = LogicAddr, txn.CloseRemainderTo = ZeroAddr,$ $txn.AssetCloseTo = ZeroAddr, txn.Index = currentTxn.Index\}) \wedge$ $^{6}\bigwedge_{t \in groupTxns - \{txn\}} \mathbf{P}(\{t.Sender = RandomAddr \vee t.Sender = ZeroAddr\}))$
(3) [SIG] Unchecked CloseRemainderTo	$^{1}\mathbf{Q}(\{currentTxn.CloseRemainderTo\}) \wedge$ $^{2}\mathbf{P}(\{currentTxn.TypeEnum = 1, currentTxn.Type = \text{"pay"}\}) \wedge$ $^{3}\mathbf{I}(currentTxn, Payment) \wedge (^{4}\mathbf{P}(GroupSize > \lvert groupTxns\rvert) \vee$ $\exists txn \in groupTxns, {}^{5}\mathbf{P}(\{txn.TypeEnum = 1, txn.Type = \text{"pay"}, txn.Sender = LogicAddr,$ $txn.Index = currentTxn.Index\}) \wedge {}^{6}\mathbf{I}(txn, Payment) \wedge {}^{7}\mathbf{Q}(\{txn.CloseRemainderTo\}) \wedge$ $^{8}\bigwedge_{t \in groupTxns - \{txn\}} \mathbf{P}(\{t.Sender = RandomAddr \vee t.Sender = ZeroAddr\}))$

Performance of Panda



The analysis time of applications and smart signatures.
A point is taken at an interval of 15 seconds.

The left figure reports the analysis time by running Panda. The median and average analysis times for applications are 15 seconds and 67 seconds, while the results for smart signatures are 19 seconds and 35 seconds, respectively.

Evaluation

	on-chain	
Type	Vulnerable (%*)	Unique (%*)
Arbitrary update	1,420 (1.43%)	147 (1.04%)
Arbitrary delete	2,590 (2.61%)	167 (1.18%)
Force clear state	1,360 (1.37%)	141 (0.99%)
Unchecked payment receiver	710 (0.72%)	48 (0.34%)
Unchecked asset receiver	123 (0.12%)	60 (0.42%)
Total	4,008 (4.04%)	364 (2.57%)

Evaluation results for on-chain applications

We used Panda to conduct a vulnerability assessment on all smart contracts on the Algorand blockchain and found 80,515 (10.38%) vulnerable smart signatures and 150,676 (27.73%) vulnerable applications. Of the vulnerable applications, 4,008 (4.04%) are still on the blockchain and have not been deleted.

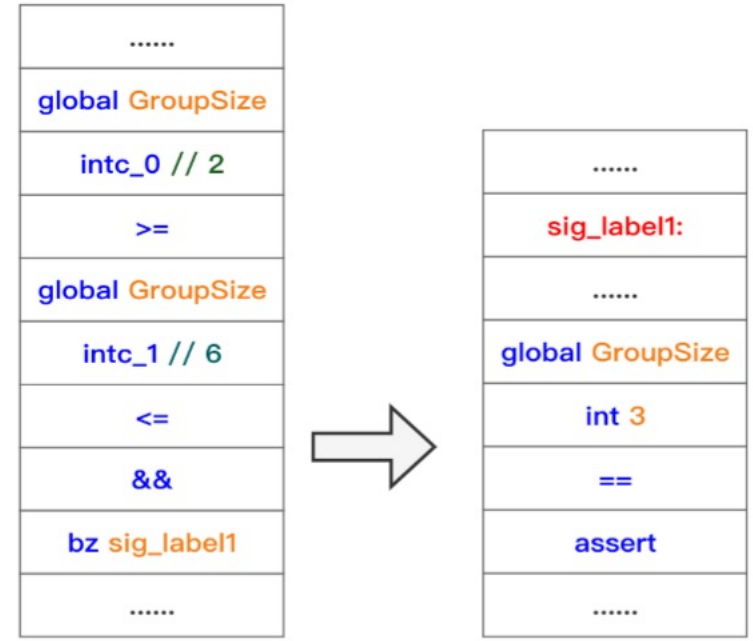
	off-chain	
Type	Vulnerable (%*)	Unique (%*)
Arbitrary update	91,246 (16.79%)	454 (0.69%)
Arbitrary delete	97,908 (18.02%)	437 (0.67%)
Force clear state	10,749 (1.98%)	441 (0.67%)
Unchecked payment receiver	1,570 (0.29%)	98 (0.15%)
Unchecked asset receiver	43,066 (7.93%)	141 (0.21%)
Total	150,676 (27.73%)	987 (1.50%)

Evaluation results for off-chain applications

	Whether contain Validator	
	YES	NO
Type	Vulnerable (%*)	Vulnerable (%*)
Unchecked transaction fees	15,539 (3.03%)	23,251 (8.83%)
Unchecked rekey_to	751 (0.15%)	8,713 (3.31%)
Unchecked close_remainder_to	42,084 (8.21%)	4,509 (1.71%)
Unchecked asset_close_to	900 (0.18%)	3,206 (1.22%)
Total	57,120 (11.14%)	23,395 (8.89%)

Evaluation results of smart signatures

Case Study (Unchecked group size)

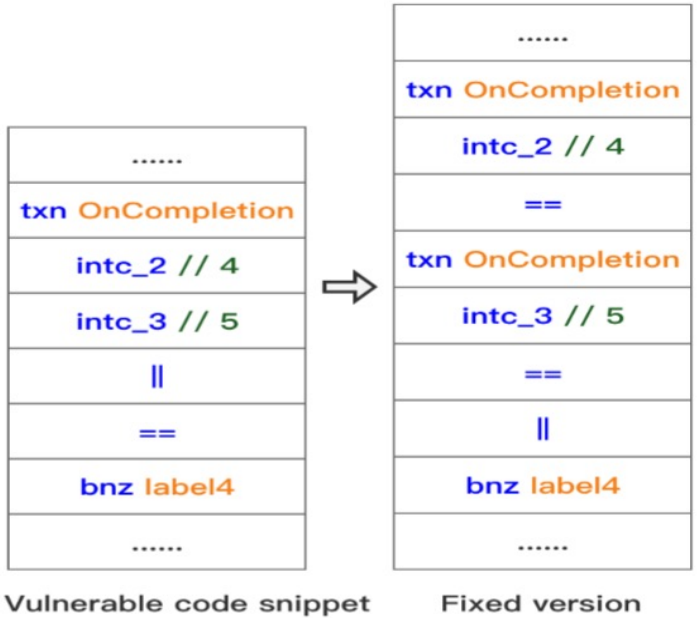


Vulnerable code snippet

Fixed version

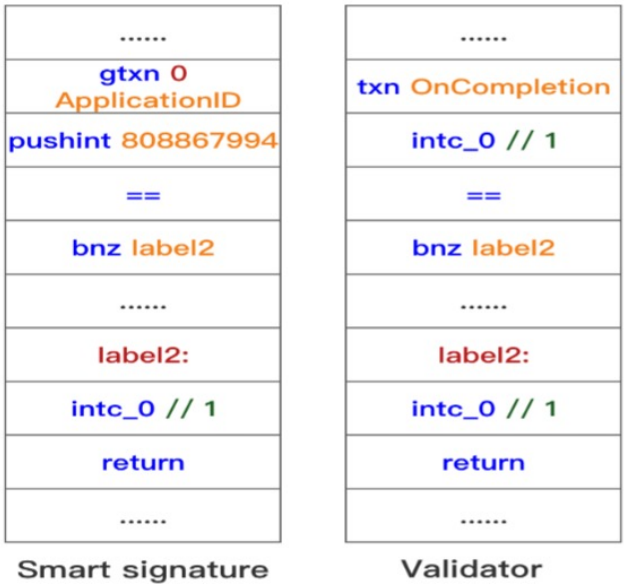
Lessons learned: We have to specify the group size explicitly and check all the parameters of each of these transactions in smart signatures or in the Validator.

Case Study (Unexpected Delete and Update Operation)



Impact: This vulnerable example has a duplicate of 333 on the blockchain.

Case Study (Validator can be bypassed)



Impact: Panda has reported a large number of smart signatures with this vulnerability pattern which includes more than 40,000 vulnerable escrow accounts of ALGOxNFT (the total trade volume exceeds 2 million Algos) and a vulnerable liquidity pool of FXDX with a deposit of more than 500,000 Algos. We reported these vulnerabilities to the corresponding developers and helped them to fix the vulnerabilities. **We also received a bug bounty of 10,000 Algos from ALGOxNFT.**

Thanks for listening!