Revisiting Smart Contract Vulnerabilities in Hyperledger Fabric

1. Background

- **Hyperledger Fabric**: permissioned enterprise blockchain, allows smart contracts written in general-purpose languages
- **Usage**: contract automation, transparency and immutability
- **Problem**: security vulnerabilities in contracts can be exploited; there is little focus on how contract vulnerabilities can be exploited and mitigated in current research.

2. Research Questions

Q: What are some **security vulnerabilities** in Hyperledger Fabric smart contracts, and what are their countermeasures?

- **How** can the vulnerabilities be **exploited**?
- **What** is the **impact severity**?
- **How** do the **countermeasures** affect the impact severity?

3. Method

Find vulnerabilities → Reproduce and exploit → Estimate impact severity → Evaluate countermeasures

References

[1] P. Lv, Y. Wang, Y. Wang, H. Wang, and Q. Zhou, "Potential risk detection system of Hyperledger Fabric smart contract based on static analysis," EasyChair, Tech. Rep., 2021.

[2] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts." in *Ndss*, 2018, pp. 1–12.

[3] B. Beckert, M. Herda, M. Kirsten, and J. Schiffl, "Formal specification and verification of Hyperledger Fabric chaincode," in *3rd Symposium on Distributed Ledger Technology (SDLT-2018) co-located with ICFEM*, 2018.

4. Vulnerabilities

Global variables

```
1 var totalAssets = 0
2 func CreateAsset(id):
3   totalAssets++
4   asset := Asset{
5       ID:    id,
6       Value: totalAssets / 100}
7   if IdIsAvailable(id):
8     return ctx.PutState(asset)
```

Listing 1: Pseudocode of the global variables vulnerability.

Rich queries

```
1 func UpdateColorByOwner(color, owner):
2   results := ctx.GetQueryResults()
3   for asset := range results {
4       asset.Color = color
5       ctx.PutState(asset)
```

Listing 2: Pseudocode of the rich queries vulnerability.

Pseudorandom number generators (PRNG)

```
1 func InitLedger():
2     lotteryNumber :=
3         rand.Int(ctx.GetTimeStamp())
4     ctx.PutState(lotteryNumber)
```

Listing 3: Pseudocode of the PRNG vulnerability.

	Global variables	PRNG	Rich queries
Base score	8.2	4.3-6.5	5.3
Impact severity	high	medium	medium
Attack complexity	low	low	high
Confidentiality		low-high	
Integrity	low		high
Availability	high		

Table 1: The base scores of the vulnerabilities in the Common Vulnerability Scoring System, with red entries being the highest security risks and green the lowest. Empty entries indicate no risk for the specific metric.

	Global variables	PRNG	Rich queries	Open source
ReviveCC	✓	✓		✓
Chaincode Scanner	✓	✓	✓	
Chaincode Analyzer	✓	✓	unk.	✓
Lv et al. [1]	✓	✓	✓	
ZEUS [2]	unk.	unk.	unk.	
Beckert et al. [3]	unk.	unk.	unk.	

Table 2: Overview of compatible analysis tools. Entries marked "✓" indicate which vulnerabilities the tool detects, and whether it is open source. Entries with "unk." are unknown.

5. Countermeasures

Global variables
- Must be avoided, impact on *availability* cannot be lowered

Rich queries
- Preferably only used in read-only transactions
- Design pattern can lower but not remove impact on *integrity*

PRNG
- *Attack complexity* can be increased by using (de-)centralized oracles

6. Conclusion

- **Countermeasures** can **lower**, but not remove, the **impact** severity
- Global variables can be **replaced** by safer alternatives, but not rich queries and (pseudo-)RNG
- **Developers** need to assess whether the **remaining impact** severity is **acceptable**
- **Static code analysis tools** are **effective** at detection, but availability of tools is **lacking**

author: Cathrine Paulsen, c.r.paulsen@student.tudelft.nl
supervisor: Prof. Dr. Kaitai Liang

CSE3000 Research Project | Jun 29, 2021