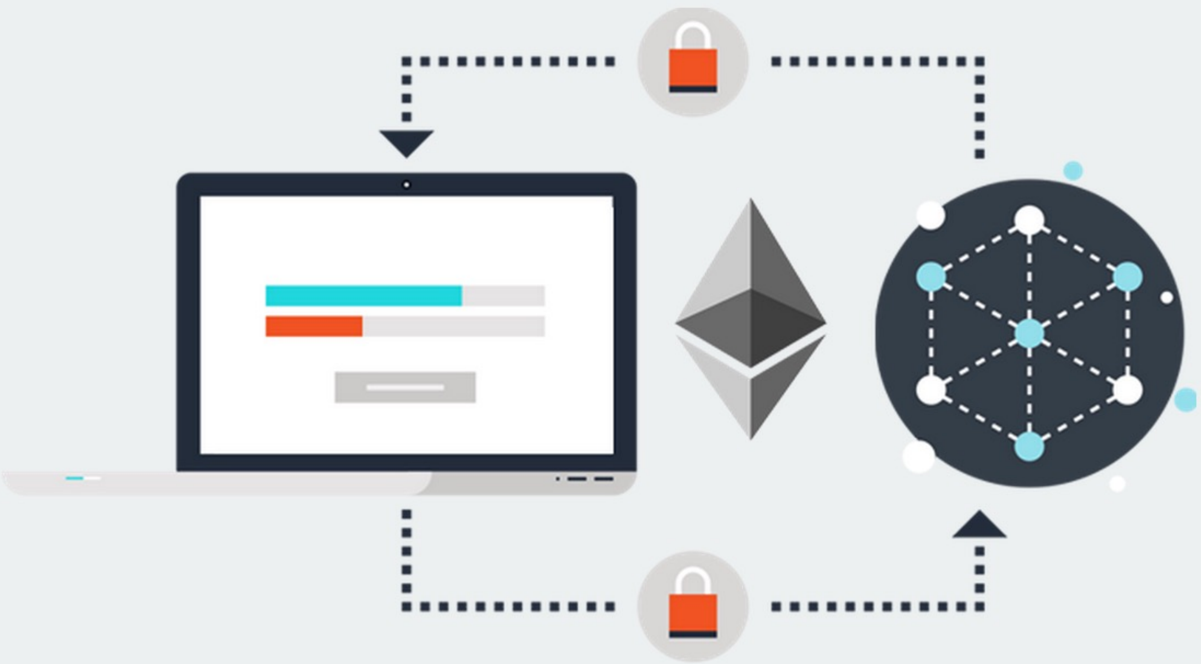
Damian Rusinek

From web apps to smart contracts: tools, vulns, standards and SCSVS



Blockchain Working Group

20th Apr 2022

Introducing Decentralized Applications by analogy to Web Apps

Damian Rusinek

drdr_zz

damianrusinek @ github

securing

Head of Blockchain Security
Security Researcher

Decentralized Apps

WHAT IS IT?

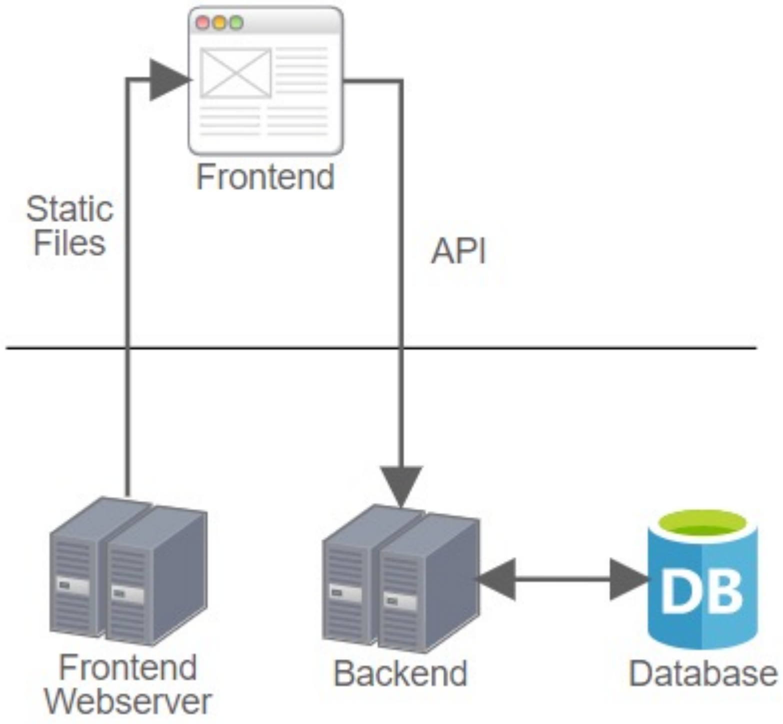
And why are they becoming important?

What is so special about Decentralized Apps?

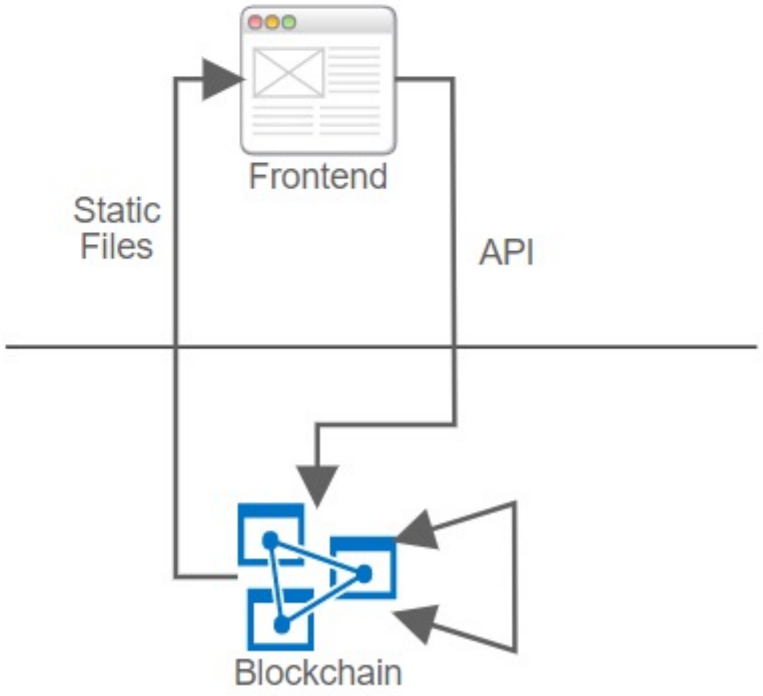
- **Trustlessness:** Use blockchain to store code and data (state).
- No one can turn it off permanently (anyone can bring it to live).
- Everyone can have it (like keeping the database of FB or Reddit locally).

Where is the main difference?
Architecture

Web Application

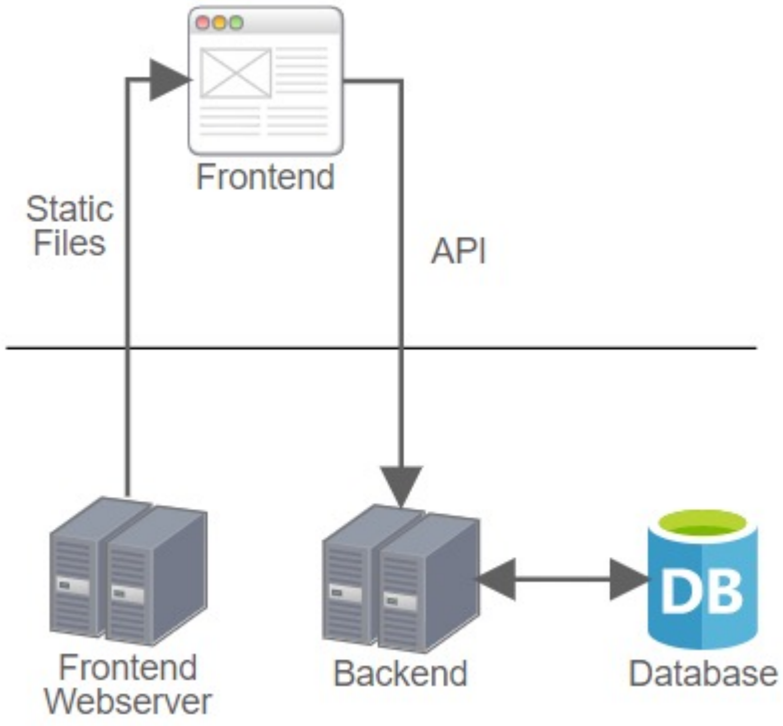


Decentralized Application

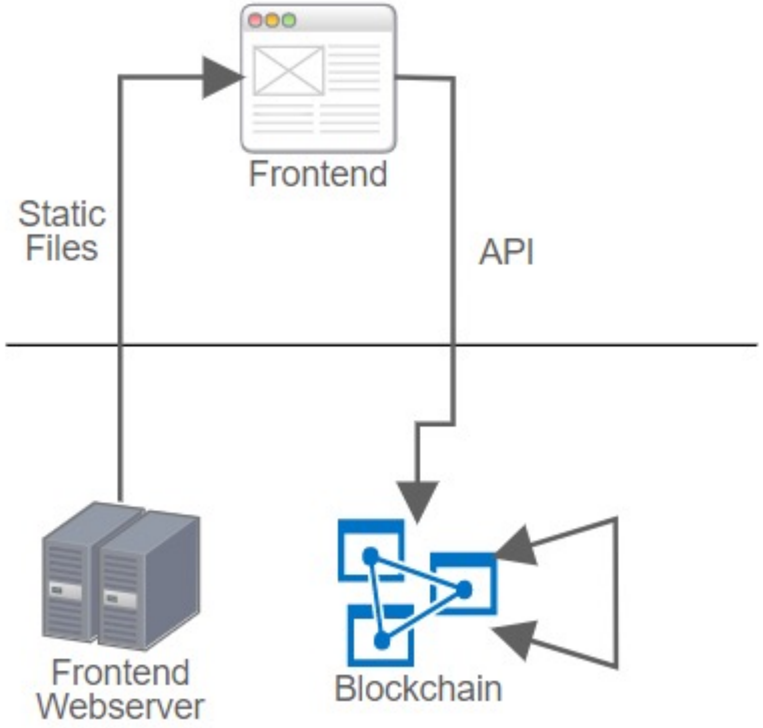


Where is the main difference?
Architecture

Web Application



Hybrid Decentralized Application



Decentralized Apps

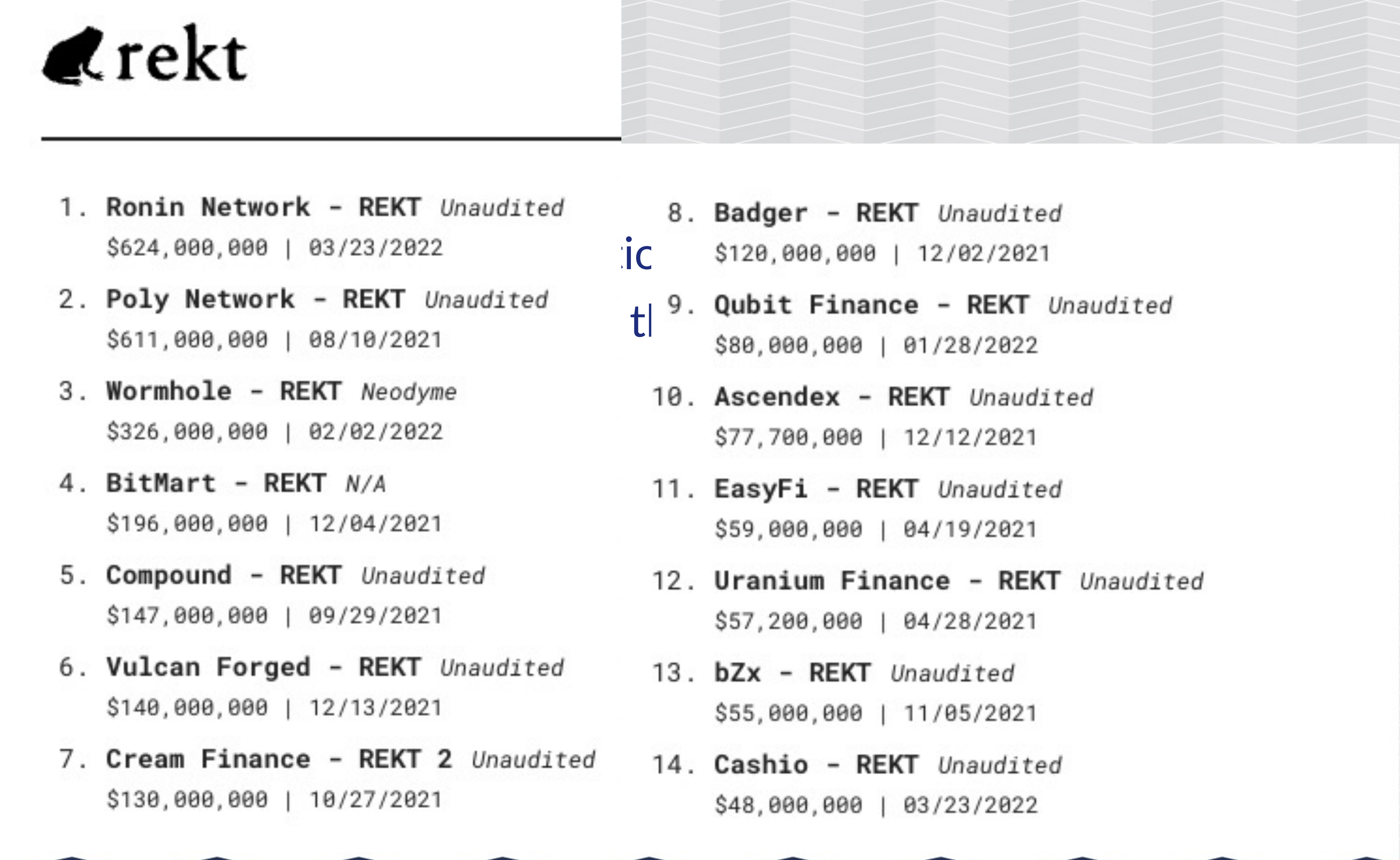
ARE THOSE SECURE?

Are Decentralized Apps secure?

- **Indestructible**: No one can turn it off
- **Cryptographically secure**: All transactions are digitally signed
- **Publicly verifiable**: Anyone can verify the code of smart contracts
- But still….

Are De

- Indest
- Crypt ic
- Public tl
- But st

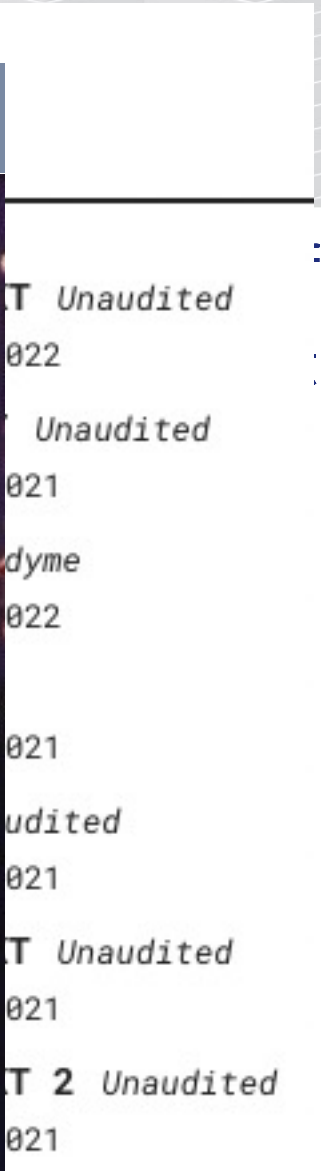


rekt

1. **Ronin Network - REKT** *Unaudited*
   $624,000,000 | 03/23/2022
2. **Poly Network - REKT** *Unaudited*
   $611,000,000 | 08/10/2021
3. **Wormhole - REKT** *Neodyme*
   $326,000,000 | 02/02/2022
4. **BitMart - REKT** *N/A*
   $196,000,000 | 12/04/2021
5. **Compound - REKT** *Unaudited*
   $147,000,000 | 09/29/2021
6. **Vulcan Forged - REKT** *Unaudited*
   $140,000,000 | 12/13/2021
7. **Cream Finance - REKT 2** *Unaudited*
   $130,000,000 | 10/27/2021
8. **Badger - REKT** *Unaudited*
   $120,000,000 | 12/02/2021
9. **Qubit Finance - REKT** *Unaudited*
   $80,000,000 | 01/28/2022
10. **Ascendex - REKT** *Unaudited*
    $77,700,000 | 12/12/2021
11. **EasyFi - REKT** *Unaudited*
    $59,000,000 | 04/19/2021
12. **Uranium Finance - REKT** *Unaudited*
    $57,200,000 | 04/28/2021
13. **bZx - REKT** *Unaudited*
    $55,000,000 | 11/05/2021
14. **Cashio - REKT** *Unaudited*
    $48,000,000 | 03/23/2022

Ar

- U
- C
- P
- B

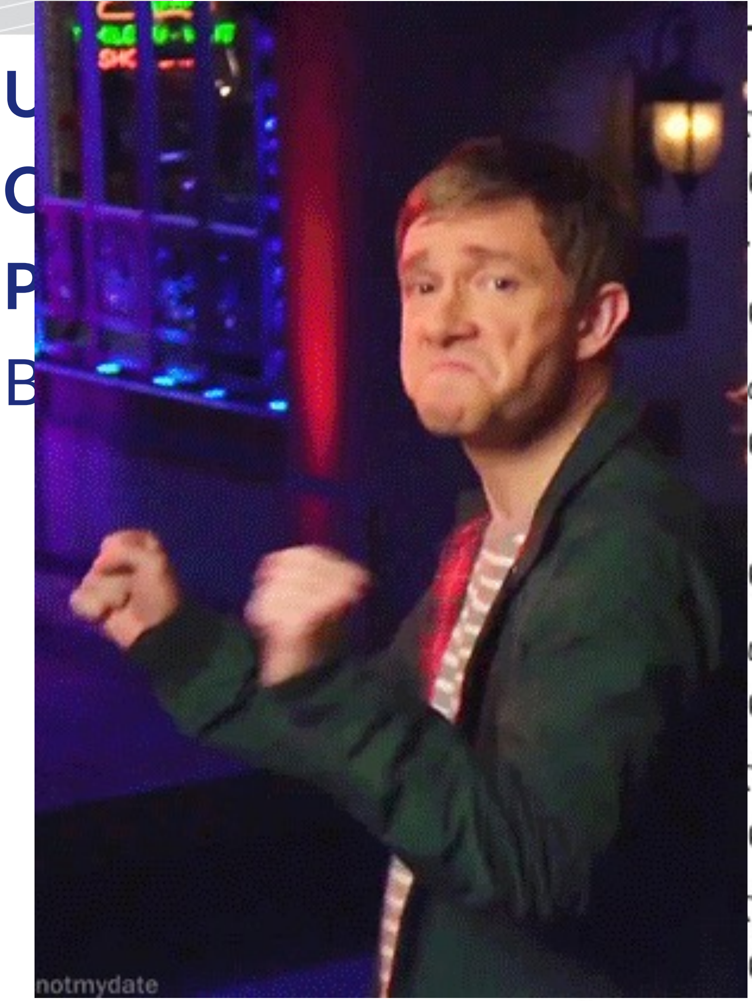


T Unaudited
022
Unaudited
021
dyme
022
021
udited
021
T Unaudited
021
T 2 Unaudited
021

ic

tl

Expectations



Reality

From web apps to smart contracts

WE NEED SECURITY!

Security needs

Technical

- Build secure applications.
  - Omit the insecure patterns.
- Find ane remediate the security bugs (vulnerabilities).

Business

- Make sure that the application is secure.
- The status: List of green and red points.

Security Projects & Standards

Web Apps

- Most common vulnerabilities?
  - **OWASP Top 10**
- The end-to-end security checklist to perform an audit?
  - **OWASP ASVS Application Security Verification Standard**

Decentralized Apps

- Most common vulnerabilities?
  - **DASP Top 10** (https://dasp.co)
- The end-to-end security checklist to perform an audit?

Smart Contracts
Security Verification Standard

SCSVS

- Smart Contracts Security Verification Standard

drdr_zz wh01s7

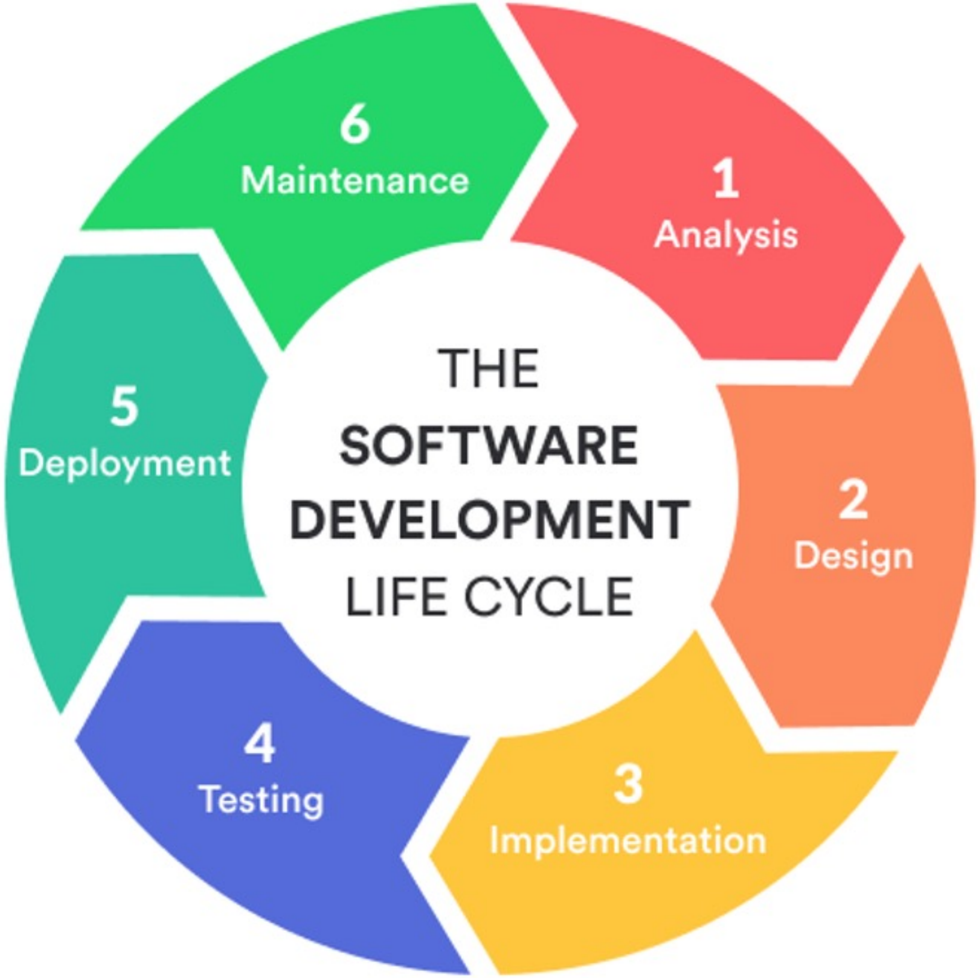
SCSVS - Objectives

Smart Contracts
Security Verification Standard

- Objectives:
  - A checklist for architects, developers and security reviewers.
- Technical needs
  - Help to mitigate known vulnerabilities by design.
  - Help to develop high quality code of the smart contracts.
- Business needs
  - Provide a clear and reliable assessment of how secure the smart contract is in relation to the percentage of SCSVS coverage.
- 14 categories of security requirements.
- Format similar to ASVS.

Software Development Life Cycle

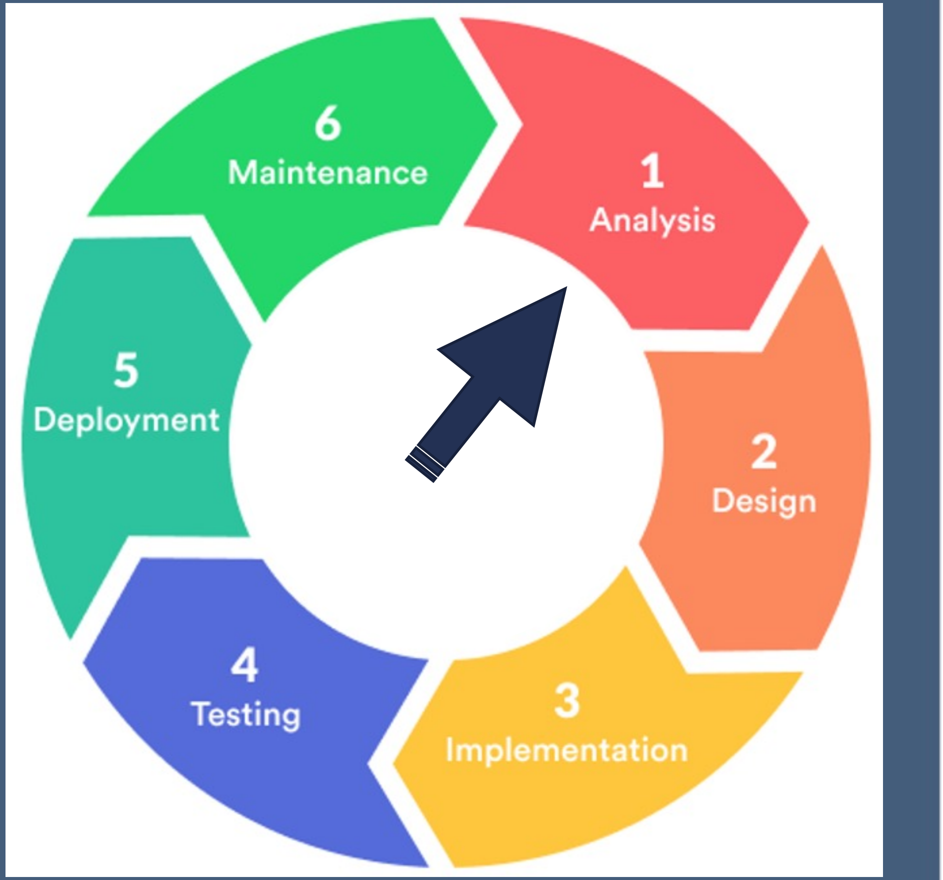
SCSVS covers all stages of SDLC process.



From web apps to smart contracts

SDLC

- Analysis & Requirements



SDLC – Analysis & Requirements

Similiarities

- Threat modelling

Smart Contracts
Security Verification Standard

1.1 Verify that the every introduced design change is preceded by an earlier threat modelling.

Smart Contracts
Security Verification Standard

1.2 Verify that the documentation clearly and precisely defines all trust boundaries in the contract (trusted relations with other contracts and significant data flows).

SDLC – Analysis & Requirements

Differences – Sensitive data

Web Apps

- Stored in protected database

Decentralized Apps

- Stored on public blockchain
  - Forever
  - Anyone can read

Smart Contracts Security Verification Standard

3.1 Verify that any data saved in the contracts is not considered safe or private (even private variables).

Smart Contracts Security Verification Standard

3.2 Verify that no confidential data is stored in the blockchain (passwords, personal data, token etc.).

SDLC – Analysis & Requirements

Differences – Randomness and oracles

Web Apps

- A matter of a function call

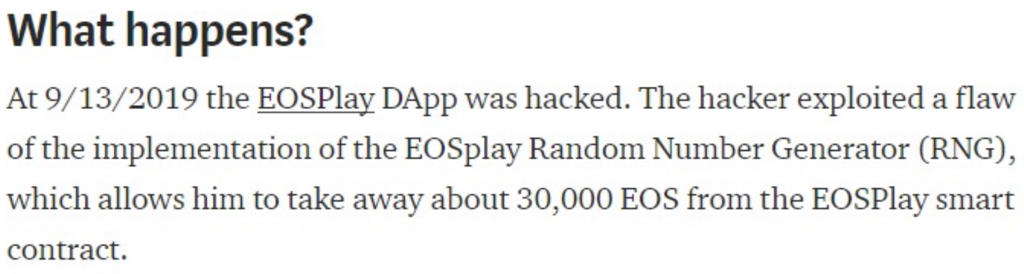
Decentralized Apps

- Not trivially achieved in the decentralized computer
- No local parameters can be used
- but…
- ETH2.0 going to change that a little bit.

SDLC – Analysis & Requirements

Differences – Randomness

- EOSPlay hack
  - 30k EOS stolen (~120k USD)



What happens?

At 9/13/2019 the EOSPlay DApp was hacked. The hacker exploited a flaw of the implementation of the EOSplay Random Number Generator (RNG), which allows him to take away about 30,000 EOS from the EOSPlay smart contract.

- DiceGame
  - my finding presented on EthCC

SCAN ME

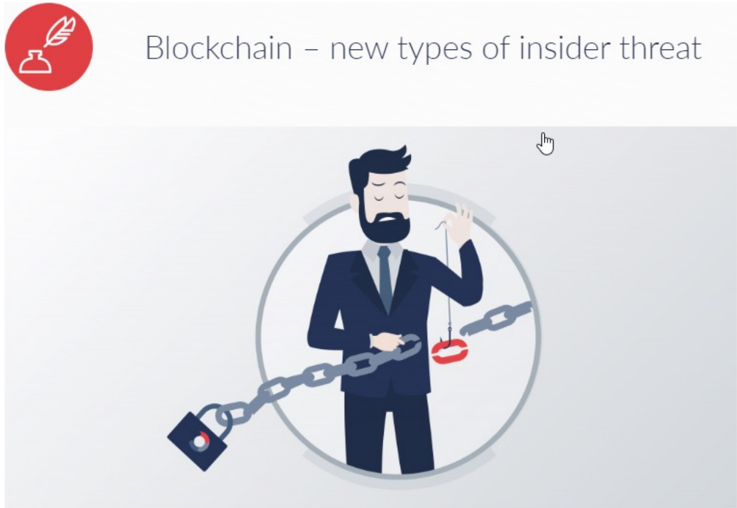
Smart Contracts Security Verification Standard

7.5 Verify that the contract does not generate pseudorandom numbers trivially basing on the information from blockchain (e.g. seeding with the block number).

SDLC – Requirements & Analysis

New threat actors for Decentralized Apps

- Miners/Validators
  - Validate transactions and add new blocks



Blockchain – new types of insider threat

SDLC – Requirements & Analysis

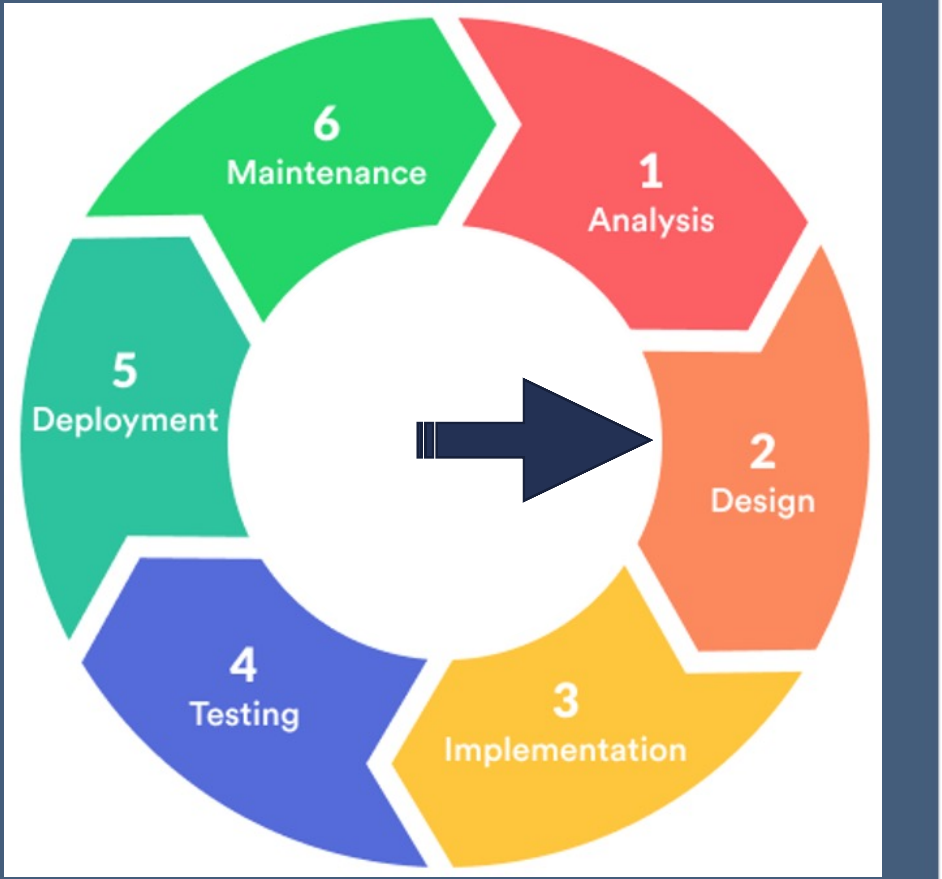
New threat actors for Decentralized Apps

Smart Contracts Security Verification Standard	8.1 Verify that the contract logic implementation corresponds to the documentation.
Smart Contracts Security Verification Standard	8.3 Verify that the contract has business limits and correctly enforces it.
Smart Contracts Security Verification Standard	9.3 Verify that the contract logic does not disincentivize users to use contracts (e.g. the cost of transaction is higher than the profit).

From web apps to smart contracts

SDLC

- Design



SDLC – Design

Similiarities

- Least privilege rule
- Access control
    - Public and known to everyone
    - Centralized and simple

Smart Contracts
Security Verification Standard

2.3 Verify that the creator of the contract complies with the rule of least privilege and his rights strictly follow the documentation.

Smart Contracts
Security Verification Standard

2.11 Verify that all user and data attributes used by access controls are kept in trusted contract and cannot be manipulated by other contracts unless specifically authorized.

SDLC – Design

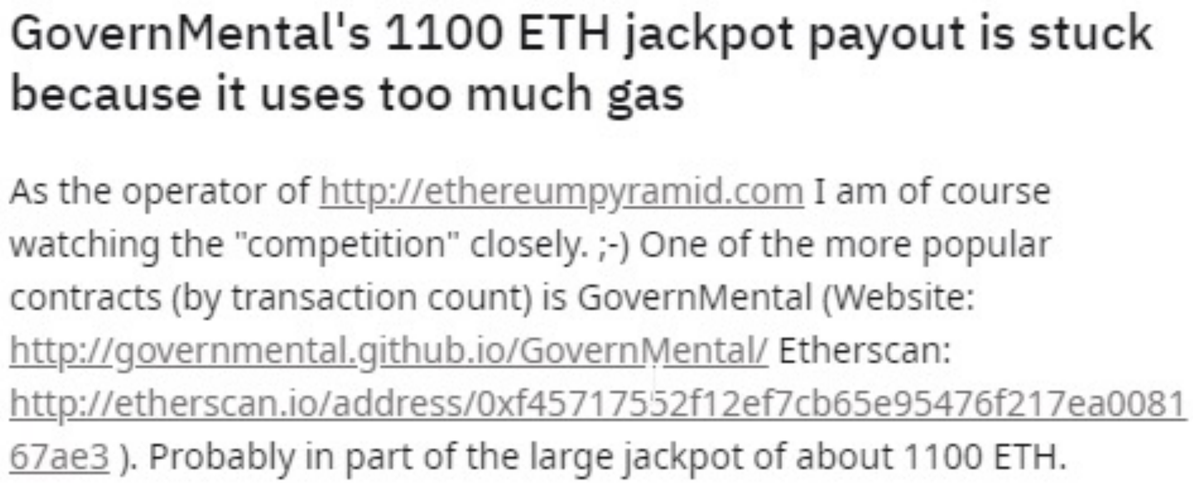
Differences – Loops

Web Apps	Decentralized Apps
• Infinite loops -> DoS	• Unbound loops -> DoS

SDLC – Design

Differences – Loops

- GovernMentals
  - A ponzi scheme
  - Iteration over a huge array
  - 1100 ETH frozen
  - https://bit.ly/2kVXwaj



GovernMental's 1100 ETH jackpot payout is stuck because it uses too much gas

As the operator of http://ethereumpyramid.com I am of course watching the "competition" closely. ;-) One of the more popular contracts (by transaction count) is GovernMental (Website: http://governmental.github.io/GovernMental/ Etherscan: http://etherscan.io/address/0xf45717552f12ef7cb65e95476f217ea008167ae3). Probably in part of the large jackpot of about 1100 ETH.

Smart Contracts
Security Verification Standard

7.3 Verify that the contract does not iterate over unbound loops.

Smart Contracts
Security Verification Standard

8.8 Verify that the contract does not send funds automatically, but it lets users withdraw funds on their own in separate transaction instead.

SDLC – Design

Decreasing the risk

- Decentralized Applications keep cryptocurrencies
- The higher the amount the bigger the incentive for hackers

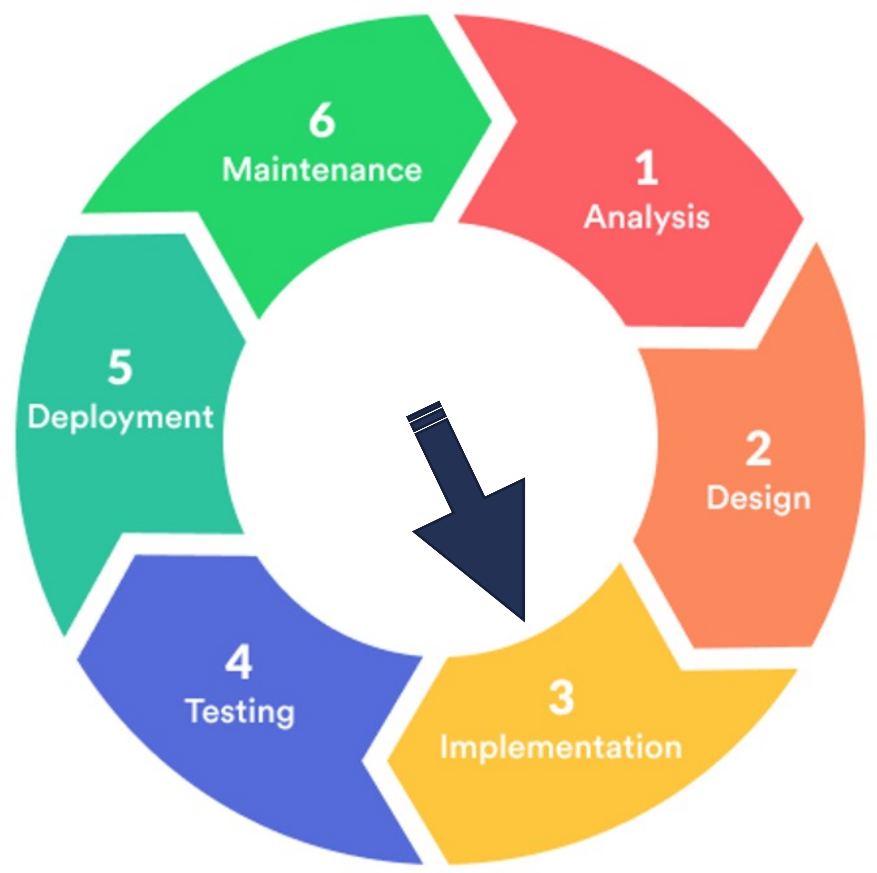
Smart Contracts
Security Verification Standard

1.9 Verify that the amount of cryptocurrencies kept on contract is controlled and at the minimal acceptable level.

From web apps to smart contracts

SDLC

- Implementation



SDLC – Implementation

- Great tools

remix Ethereum Studio Hardhat Foundry

  - Perform basic security analysis

- But we still make bugs.
- Sounds familiar? ☺

SDLC – Implementation

Similarities – Arithmetic bugs

Web Apps

- Not that common

Decentralized Apps

- Overflows and underflows
- …yep, still after 0.8 with *unchecked*

SDLC – Implementation

Similarities – Arithmetic bugs

- Multiple ERC20 Smart Contracts
  - Allow to transfer more than decillions (10^60) of tokens
  - https://bit.ly/2lWa9ma
  - https://bit.ly/2ksNEF1

SDLC – Implementation

Similarities – Arithmetic bugs

- Tellor
  - Not trivial
    - Required staking
  - Reported
  - No funds stolen

- my finding presented on EthCC

SCAN ME

SDLC – Implementation

Similarities – Arithmetic bugs

Smart Contracts Security Verification Standard

5.1 Verify that the values and math operations are resistant to integer overflows. Use SafeMath library for arithmetic operations before solidity 0.8.*.

Smart Contracts Security Verification Standard

5.2 Verify that the unchecked code snippets from Solidity 0.8.* do not introduce integer under/overflows.

Smart Contracts Security Verification Standard

5.3 Verify that the extreme values (e.g. maximum and minimum values of the variable type) are considered and does change the logic flow of the contract.

SDLC – Implementation

Differences – Recursive calls

Web Apps

- Must be explicitly included in the logic

Decentralized Apps

- Executing some logic multiple times in one call
- The DAO hack
  - Recursive withdrawals
  - 3.6 mln ETH stolen
  - https://bit.ly/2hBQjKq

Smart Contracts Security Verification Standard

4.5 Verify that re-entrancy attack is mitigated by blocking recursive calls from other contracts. Follow CEI pattern.

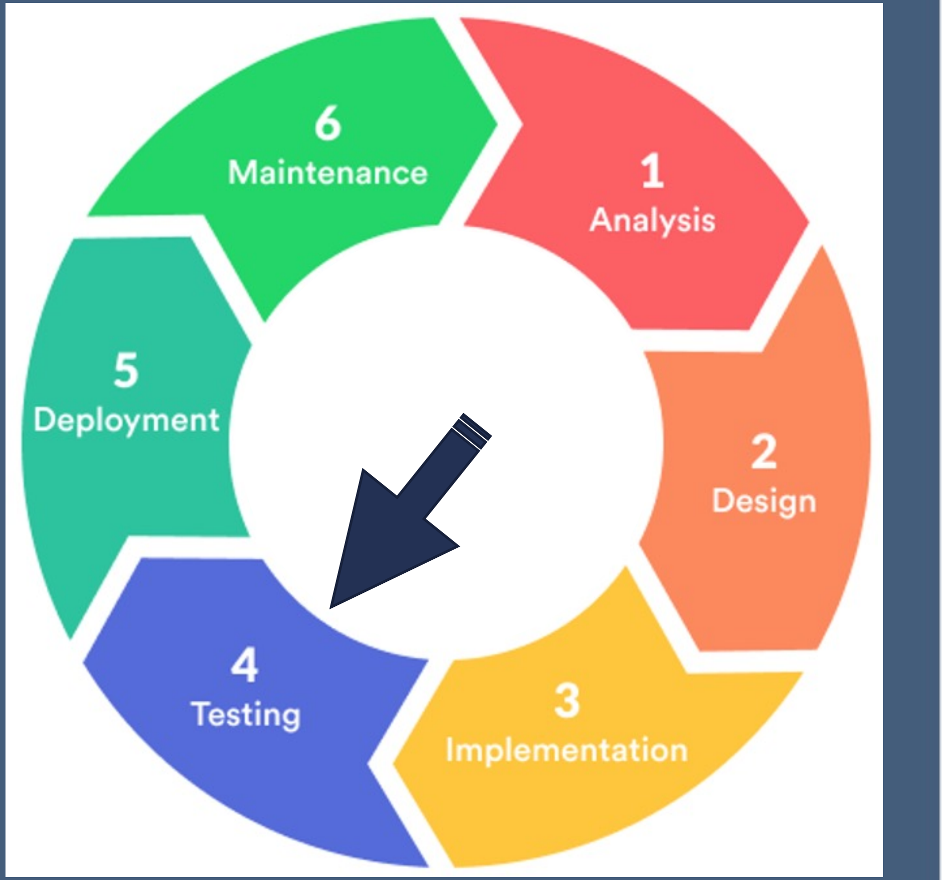
Smart Contracts Security Verification Standard

4.6 Verify that the result of low-level function calls (e.g. send, delegatecall, call) from another contracts is checked.

From web apps to smart contracts

SDLC

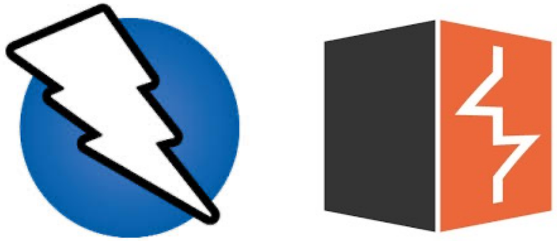
- Testing



SDLC – Testing

Similarities – Great tools for automatic scans

Web Apps



Decentralized Apps

DILIGENCE Scribble

Solhint

https://bit.ly/2mpaL3U

SLITHER

MythX

https://mythx.io/

Smart Contracts
Security Verification Standard

1.12 Verify that code analysis tools are in use that can detect potentially malicious code.

SDLC – Analysis & Requirements

Similiarities – Ensuring the testing takes place

Smart Contracts Security Verification Standard

12.1 Verify that all functions of verified contract are covered with tests in the development phase.

Smart Contracts Security Verification Standard

12.2 Verify that the implementation of verified contract has been checked for security vulnerabilities using static and dynamic analysis.

Smart Contracts Security Verification Standard

12.3 Verify that the specification of smart contract has been formally verified.

Smart Contracts Security Verification Standard

12.4 Verify that the specification and the result of formal verification is included in the documentation.

- including manual security tests

Smart Contracts Security Verification Standard

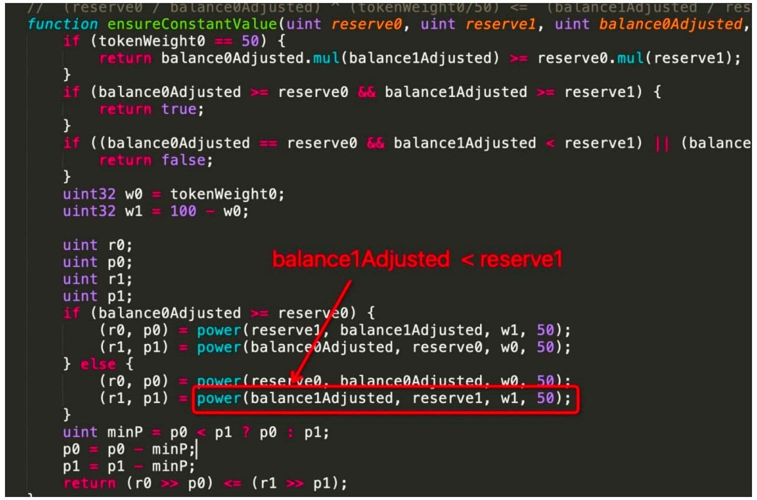
1.3 Verify that the SCSVS, security requirements or policy is available to all developers and testers.

SDLC – Analysis & Requirements

Similiarities – Business logic errors

- Hard to find using automated scans
- Value DeFi
    - Incorrect assumptions
    - 10m$ lost
    - „improper use of a complex exponentiation power() function”

https://rekt.news/value-rekt3/



```
// (reserve0 / balance0Adjusted) ^ (tokenWeight0/50) <= (balance1Adjusted / res
function ensureConstantValue(uint reserve0, uint reserve1, uint balance0Adjusted,
    if (tokenWeight0 == 50) {
        return balance0Adjusted.mul(balance1Adjusted) >= reserve0.mul(reserve1);
    }
    if (balance0Adjusted >= reserve0 && balance1Adjusted >= reserve1) {
        return true;
    }
    if ((balance0Adjusted == reserve0 && balance1Adjusted < reserve1) || (balance
        return false;
    }
    uint32 w0 = tokenWeight0;
    uint32 w1 = 100 - w0;

    uint r0;
    uint p0;
    uint r1;
    uint p1;
    if (balance0Adjusted >= reserve0) {
        (r0, p0) = power(reserve1, balance1Adjusted, w1, 50);
        (r1, p1) = power(balance0Adjusted, reserve0, w0, 50);
    } else {
        (r0, p0) = power(reserve0, balance0Adjusted, w0, 50);
        (r1, p1) = power(balance1Adjusted, reserve1, w1, 50);
    }
    uint minP = p0 < p1 ? p0 : p1;
    p0 = p0 - minP;
    p1 = p1 - minP;
    return (r0 >> p0) <= (r1 >> p1);
```

balance1Adjusted < reserve1

Smart Contracts Security Verification Standard

1.11 Verify that the business logic in contracts is consistent. Important changes in the logic should be allowed for all or none of the contracts.

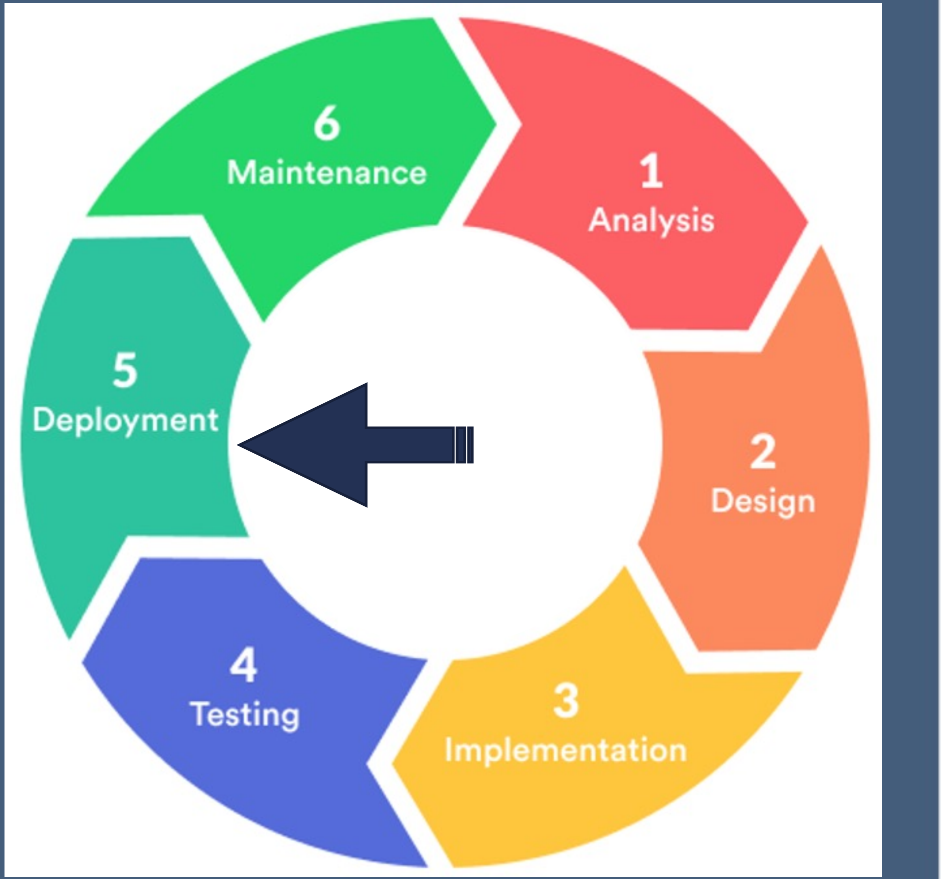
Smart Contracts Security Verification Standard

8.2 Verify that the business logic flows of smart contracts proceed in a sequential step order and it is not possible to skip any part of it or to do it in a different order than designed.

From web apps to smart contracts

SDLC

- Deployment



SDLC – Deployment

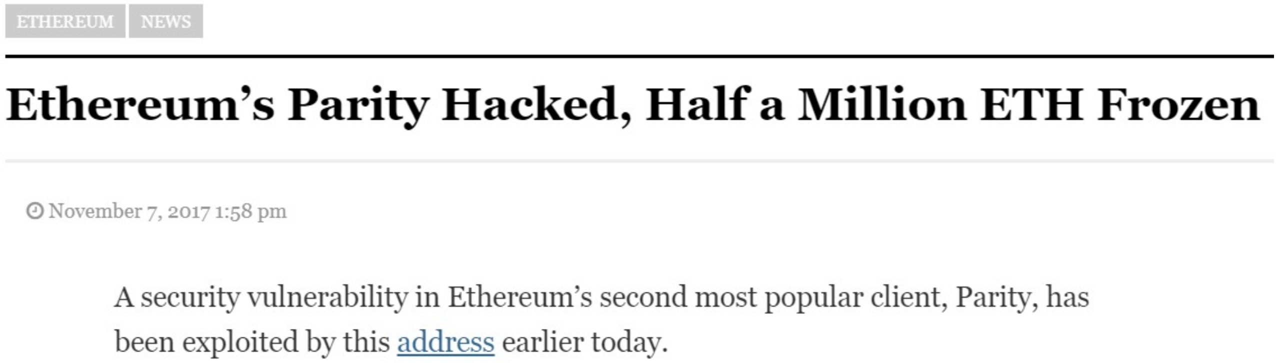
Differences – Initialization stage

Web Apps	Decentralized Apps
• Setting up configurations and integrations	• Setting up configurations and integrations
• Performed once during deployment	• What if one can (re-)initialize the contract?

SDLC – Deployment

Differences – Initialization stage

- Parity Wallet hack:
  - Kill contract shared by hundreds of other contracts
  - 500k ETH frozen
  - https://bit.ly/2kIBYhA
  - https://bit.ly/2kpfKkm



ETHEREUM NEWS

Ethereum's Parity Hacked, Half a Million ETH Frozen

November 7, 2017 1:58 pm

A security vulnerability in Ethereum's second most popular client, Parity, has been exploited by this address earlier today.

SDLC – Deployment

Differences – Initialization stage

Smart Contracts Security Verification Standard

11.7 Verify that all storage variables are initialised.

Smart Contracts Security Verification Standard

2.8 Verify that the initialization functions are marked internal and cannot be executed twice.

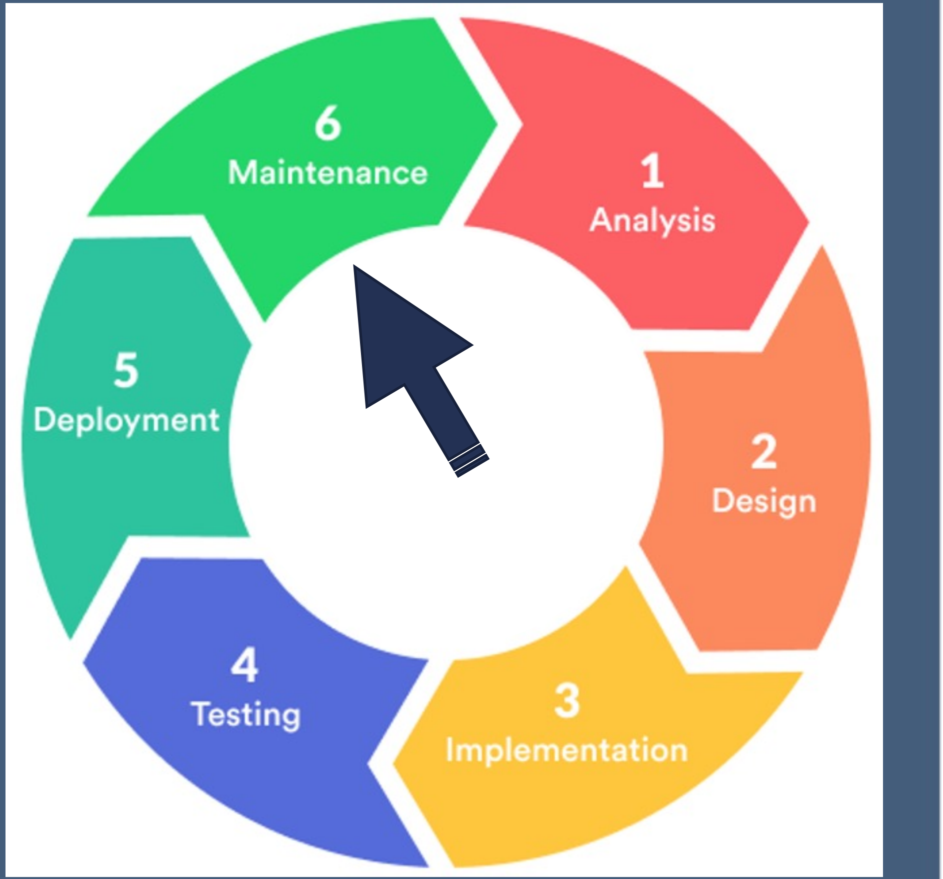
Smart Contracts Security Verification Standard

9.1 Verify that the self-destruct functionality is used only if necessary.

From web apps to smart contracts

SDLC

- Maintenance



SDLC – Analysis & Requirements

Differences – Security Alert and Fix

Web Apps

- Application goes down
- The bug is fixed (patch)
- Application redeployed

Decentralized Apps

- ~~Smart contract goes down~~
- The bug is fixed (patch)
- Smart contract deployed again

Smart Contracts Security Verification Standard

1.7 Verify that there exists a mechanism that can temporarily stop the sensitive functionalities of the contract in case of a new attack. This mechanism should not block access to the assets (e.g. tokens) for the owners.

Smart Contracts Security Verification Standard

1.4 Verify that there exists an upgrade process for the contract which allows to deploy the security fixes or it is clearly stated that the contract is not upgradeable.

Security Projects & Standards

Web Apps

- Most common vulnerabilities?
  - **OWASP Top 10**
- The end to end security checklist to perform an audit?
  - **OWASP ASVS (Application Security Verification Standard)**

Decentralized Apps

- Most common vulnerabilities?
  - **DASP Top 10** (https://dasp.co)
- The end to end security checklist to perform an audit?

SCSVS

SCSVS meets your security needs

Technical

- Build secure applications.
    - Omit the insecure patterns.
- Find ane remediate the security bugs (vulnerabilities).

Business

- Make sure that the application is secure.
- The status: List of green and red points.

Go for SCSVS!

Smart Contracts
Security Verification Standard

SCSVS 2.0

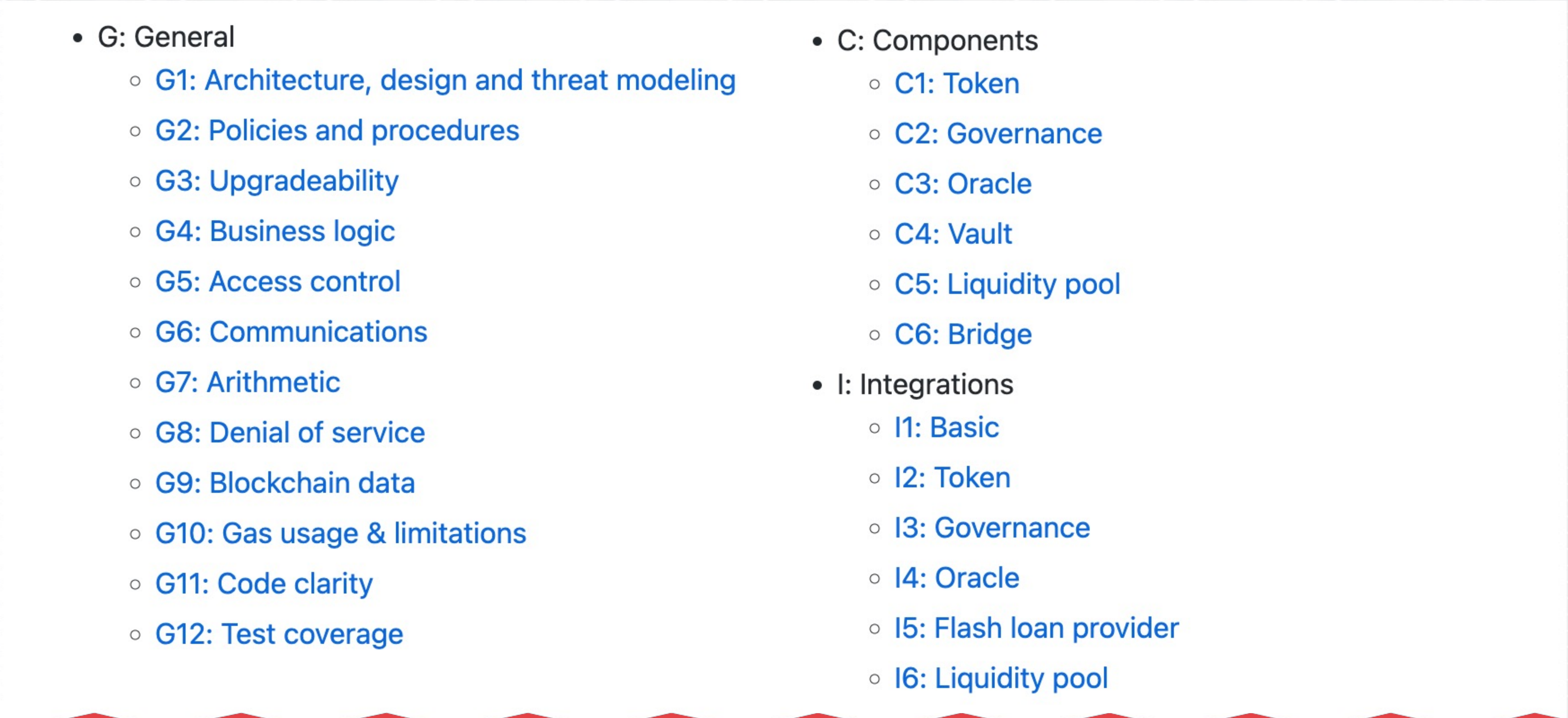
- The Future

drdr_zz wh01s7

SCSVS 2.0

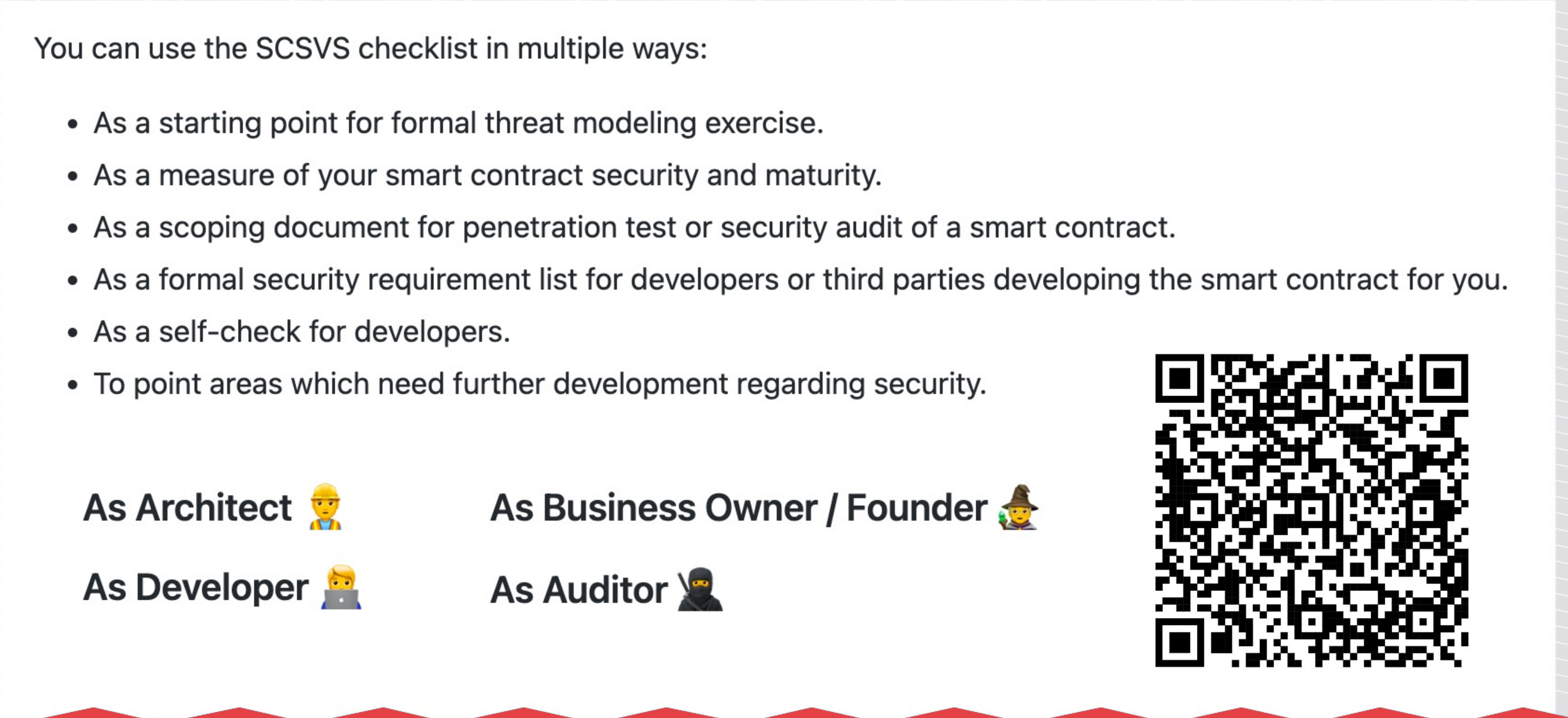
COMPOSABILITY

SCSVS 2.0 - categories



- G: General
  - G1: Architecture, design and threat modeling
  - G2: Policies and procedures
  - G3: Upgradeability
  - G4: Business logic
  - G5: Access control
  - G6: Communications
  - G7: Arithmetic
  - G8: Denial of service
  - G9: Blockchain data
  - G10: Gas usage & limitations
  - G11: Code clarity
  - G12: Test coverage
- C: Components
  - C1: Token
  - C2: Governance
  - C3: Oracle
  - C4: Vault
  - C5: Liquidity pool
  - C6: Bridge
- I: Integrations
  - I1: Basic
  - I2: Token
  - I3: Governance
  - I4: Oracle
  - I5: Flash loan provider
  - I6: Liquidity pool

SCSVS 2.0 – how to use



You can use the SCSVS checklist in multiple ways:

- As a starting point for formal threat modeling exercise.
- As a measure of your smart contract security and maturity.
- As a scoping document for penetration test or security audit of a smart contract.
- As a formal security requirement list for developers or third parties developing the smart contract for you.
- As a self-check for developers.
- To point areas which need further development regarding security.

As Architect 👷

As Business Owner / Founder 🧙

As Developer 👨‍💻

As Auditor 🥷

securing

Want to develop secure smart contracts?

Want a security audit of smart contract?

Go for SCSVS!

Ok, Thank you! drdr_zz

Damian.Rusinek@securing.pl