2018

Secure SDLC Practices in Smart Contracts Development

Speaker: Pavlo Radchuk
@rdchksec

About me

AppSec Engineer with Masters degree (several of experience)

Smart Contract Audit Team Lead

My team performs 7-10 audits per month

What do my team do

Conducting different researches for new techniques, vulns etc.

Analyzing competitors reports – they are quite different

See all the problems from inside …

Audit Problems

No compliances (e.g. PCI DSS)

No certifications (e.g. OSCP)

No industry accepted standards and guidelines (e.g. OWASP testing guide)

There are some best practices for Ethereum Solidity, but none for EOS, NEO, NEM, etc.

HACKEN

Audit says smart contracts is secure != Secure Smart Contract

KCon

Despite all the drawbacks – an audit is still the best solution for smart contract security

Audits alone are not enough – so what can be done?

What can help with Smart Contracts Security

What do web guys do for security?

Security is achieved by processes

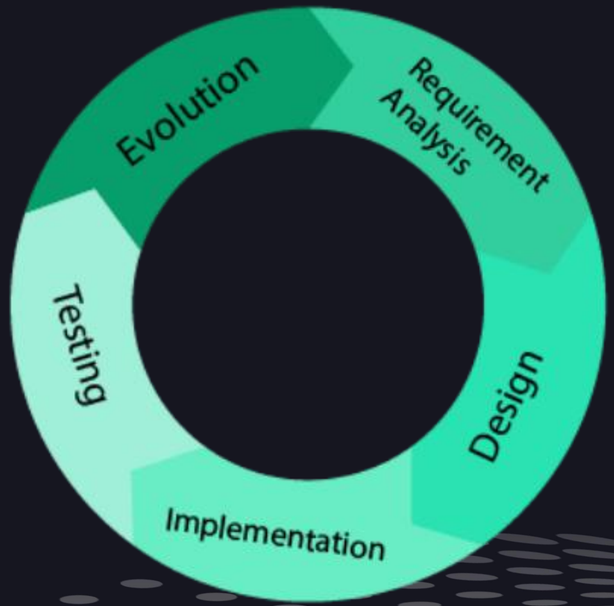
SDLC is a term used in systems engineering, information systems and software engineering to describe a process for planning, creating, testing, and deploying an information system*

Secure SDLC

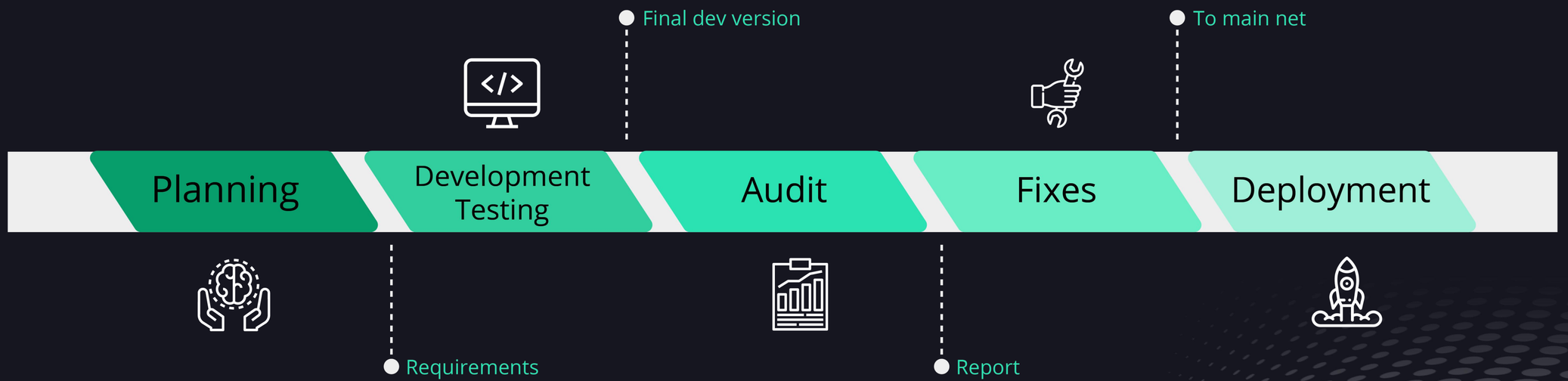
Software Development Lifecycle

* https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf

Classic Web Development Cycle



Typical Smart Contract Development Flow



Smart contracts are immutable after deployment

Web vs Smart Contracts

Web	Smart Contracts
• Some Code Run on Servers • Code can be changed	• Some Code Run on Nodes • If you use proxies – code can be changed (for instance, zos for Solidity)
Development process contains – requirements, programming, testing, deployment, maintenance	
• Existing development guides, pentesting methodologies and compliances	• Some unformalized best practices

How to "buidl" a secure smart contract?

Process

SDLC Practices

1. Threat Assessment
2. Security Requirements
3. Developer Education
4. Private Key Management
5. QA Testing
6. Security Testing
7. Compliance

1. Threat Assessment

Understanding threats:

What ifs:

- What if the only copy of private key is lost
- What if Ethereum gets hacked/DoSed etc. – can you fully rely on a third party?
- What if your token/wallet/etc. gets hacked

You need to understand the risks and accept/mitigate/transfer them

2. Security Requirements

One of the most widespread bugs – absence of security modifiers

All Security modifiers should be defined

Particularly, all function with all modifiers predefined and documented

```
1  pragma solidity ^0.4.24;
2  contract Unprotected{
3      address private owner;
4      modifier onlyowner {
5          require(msg.sender==owner);
6          _;
7      }
8      constructor()public
9      { owner = msg.sender; }
10     function changeOwner(address _newOwner) public
11     {
12         owner = _newOwner;
13     }
14     function changeOwner_fixed(address _newOwner) public onlyowner
15     {
16         owner = _newOwner;
17     }
18 }
19
20
```

https://github.com/trailofbits/not-so-smart-contracts/blob/master/unprotected_function/Unprotected.sol

3. Developer Education

Developers should know common vulnerabilities/attacks:

Examples for Solidity:

- Reentrancy
- Unchecked math
- Timestamp Dependence
- Unchecked external call

```
function getRate() public view returns(uint256) {
    uint256 rate;
    if (now < (startTime + 10 days)) {
        rate = priceEth.mul(100).div(tokenPrice).mul(bonus1.add(100)).div(100);
    } if (now > (startTime + 10 days) && now < (startTime + 20 days)) {
        rate = priceEth.mul(100).div(tokenPrice).mul(bonus2.add(100)).div(100);
    } else {
        rate = priceEth.mul(100).div(tokenPrice).mul(bonus3.add(100)).div(100);
    }
    return rate;
}


// in buy function
uint256 tokens = weiAmount.mul(getRate());

```

4. Private Key Management

Contract management architecture –operator and other management accounts; Multisig wallets

How and where PKs are stored and used?

5. QA Testing

Unit and other QA tests

How fixes should be done

- Fixes during development
- Proxies and operators for deployed contracts

6. Security Testing

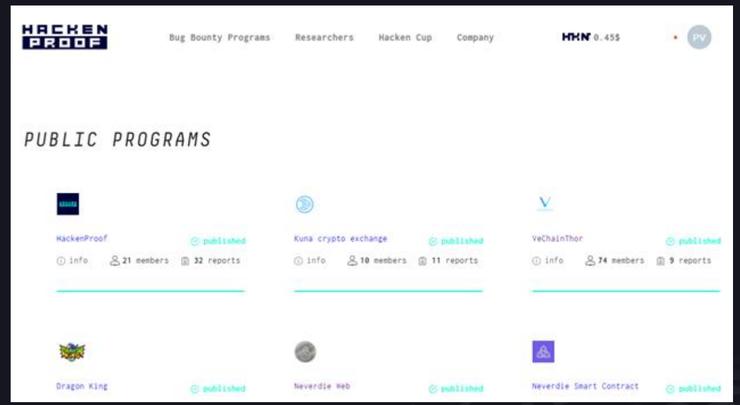
Testing against security requirements

Audit

One more audit

Bug Bounty

*https://blog.hackenproof.com/industry-news/smart-contracts-bug-hunting/



7. Compliance

Legal compliance – KYC for anti money laundering etc.

Listing requirements – security audits

Technical compliance – security requirements (like PCI DSS)

Conclusion

Audits are a must, but not enough

Security needs a process

Developers need our help

Conclusion

Web security SDLC practices are applicable for Smart Contracts

We develop best practices/recommendations

Contact me if you want to participate

Contacts

Speaker: Pavlo Radchuk
Twitter: @rdchksec
WeChat: @rdchksec
Email: p.radchuk@hacken.io