Security Testing of Solidity Smart Contracts by Deep Dive Cybersecurity

Budapest Blockchain Community Week
Ethereum Hungary **- March 21, 2024**

Agenda

- Comparison of Different Blockchain Audit Services
- Recommendations on How to Prepare For an Audit
- Common Vulnerabilities in the Smart Contracts

Security Testing of Solidity Smart Contracts

Whoami

- Former Cyber Security Specialist Lead @ Deloitte
- Former Lead Offensive Security Engineer @ Halborn
- For the past two years, I have exclusively hacked smart contracts.

István Böhm

Common Vulnerabilities

Disclaimer

The following section contains my personal opinion on the various audit services.

Audit Services

Important Questions

- When can the audit start?
- How flexible is the service?
- What service do you need?
- What type of report do you need?
- What level of confidentially do you need?
- How trustworthy is the vendor/auditor?
- What are the additional services?

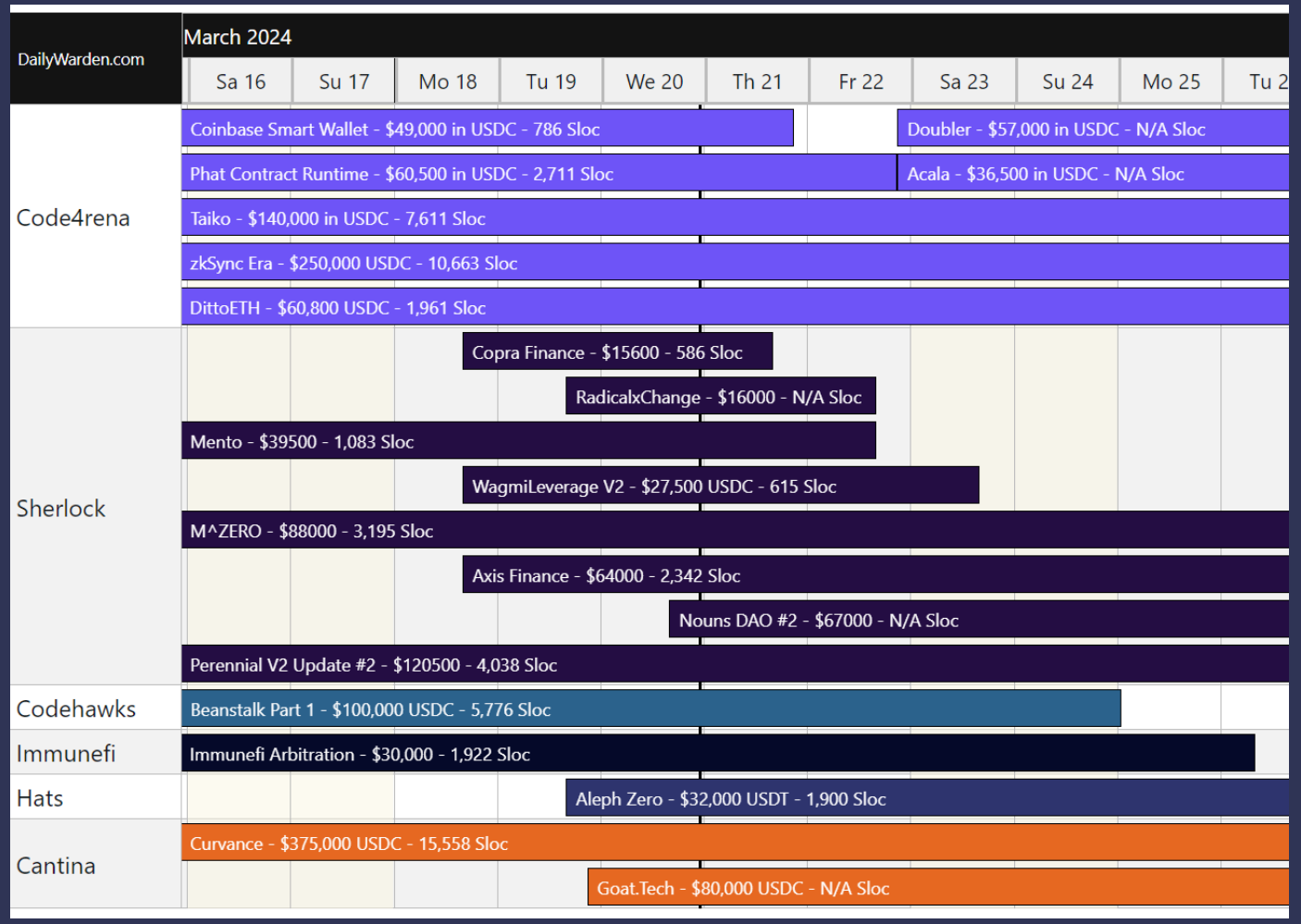
Audit Services

Security Contests

- Many Eyes
- Good Marketing Opportunities

- Expose The Code
- Many Parallel Competition
- Can Be Costly

- Private Contests



Audit Services

Private Audits

- More Flexible
- Less Expensive

- Different Skill Levels
- Authenticity and Reliability Are Very Important Factors

From Peter Steiner's 1993 cartoon

Audit Services

Security Firms

- Reputation and Marketing Opportunities
- Scalability
- Standard Quality

- Less Flexibility
- Higher Price

REKT serves as a public platform for anonymous authors to disclosure blockchain security incidents.

SBF - THE REGULATOR

Friday, October 21, 2022
SBF - Regulation

After aggressively farming many of DeFi's most lucrative opportunities since 2020, SBF is now suggesting his own industry standards, many of which go against the entire concept of decentralisation. Sam says we need "customer protection". But from who?

MORE

MOOLA MARKET - REKT

Wednesday, October 19, 2022
Moola Market - REKT

Bear markets offer easy opportunities to market manipulators, who find it easier to move prices when liquidity is low. Lending protocol Moola Market is the latest to fall victim to a "highly profitable trading strategy", and the first CELO protocol on the rekt.news leaderboard (#63).

MORE

DAO MAKER - COMMUNITY INVESTIGATES

Friday, October 14, 2022
DAO Maker

This is a community-led investigation by rekt.news readers. DAO Maker, after getting rekt for $7M and then $4M last year, proposed a compensation plan to the affected users. But it appears the team had a change of heart.

MORE

MANGO MARKETS - REKT

Wednesday, October 12, 2022
Mango Markets - REKT

Solana's flagship margin trading protocol lost 9 figures to a well-funded market manipulator. The attacker managed to spike the price of Mango Markets' native token MNGO and drain their lending pools, leaving the protocol with $115M of bad debt.

MORE

1. **Ronin Network - REKT** *Unaudited*
   $624,000,000 | 03/23/2022
2. **Poly Network - REKT** *Unaudited*
   $611,000,000 | 08/10/2021
3. **BNB Bridge - REKT** *Unaudited*
   $586,000,000 | 10/06/2022
4. **Wormhole - REKT** *Neodyme*
   $326,000,000 | 02/02/2022
5. **BitMart - REKT** *N/A*
   $196,000,000 | 12/04/2021

https://rekt.news

Audit Preparation

Before an Audit

- Use Security Best Practices
- Proper Documentation
- Unit And Integration Tests
- Make Sure You Properly Describe and Present the Protocol to the Vendor For Accurate Scoping
- Code Freeze
- Review the Contracts Before the Audit For Low-Hanging Fruits
- Make Sure the Scope Is Clear

Audit Preparation

During an Audit

- Ask Intermittent Results
- Ask and Give Continuous Feedback and Communicate with the Auditors.
- Make Sure Everything is Clear and There Is No Blockers

Audit Preparation

After Audit

- Read the Report
- Give Feedback and Discuss Problems
- Create Tests for the Identified Findings
- Review the Remediations

Common Vulnerabilities

Disclaimer

The following issues have only been selected as examples to demonstrate common vulnerabilities, and the presentation cannot be considered as a complete mitigation guide or checklist.

Common Vulnerabilities

Arbitrary From Parameter

ftrace | funcSig

```
function deposit(
    address from,
    address to,
    uint256 amount
) external {
    IERC20(nativeToken).safeTransferFrom(from,address(this),amount);
    balances[to] += amount;
}
```

Common Vulnerabilities

Arbitrary From Parameter

```
ftrace | funcSig
function deposit(
    address from,
    address to,
    uint256 amount
) external {
    IERC20(nativeToken).safeTransferFrom(from,address(this),amount);
    balances[to] += amount;
}
```

- Tokens can be deposited from users who have approved the protocol for transferring their tokens.

Common Vulnerabilities

Non Standard Tokens

```
function deposit(
    address from,
    address to,
    uint256 amount
) external {
    IERC20(nativeToken).transferFrom(from,address(this),amount);
    balances[to] += amount;
}
```

```
ftrace | funcSig
function setRewardPool(address _token, bytes32 _poolId) external onlyOwner {
    if (_poolId != bytes32(0)) {
        IERC20(_token).approve(address(balancerVault), 0);
        IERC20(_token).approve(address(balancerVault), type(uint256).max);
    }
    extraRewardPools[_token] = _poolId;
}
```

Common Vulnerabilities

Non Standard Tokens

```
function deposit(
    address from,
    address to,
    uint256 amount
) external {
    IERC20(nativeToken).transferFrom(from,address(this),amount);
    balances[to] += amount;
}
```

```
ftrace | funcSig
function setRewardPool(address _token, bytes32 _poolId) external onlyOwner {
    if (_poolId != bytes32(0)) {
        IERC20(_token).approve(address(balancerVault), 0);
        IERC20(_token).approve(address(balancerVault), type(uint256).max);
    }
    extraRewardPools[_token] = _poolId;
}
```

- Transfer Does Not Revert (e.g., ZRX)
- Transfer/Approve Does Not Return Value (e.g., USDT)
- Fee On Transfer Tokens
- Rebase / Inflationary / Deflationary Tokens

Common Vulnerabilities

Frontrunning

```
function sellOrder(address nft, uint256 id, uint256 price) external {
    // transfer NFT to Market from Seller
    // save Order
}

function buyOrder(address nft, uint256 id) external {
    // send payment from Buyer to Seller
    // receive NFT from Market
}
```

Common Vulnerabilities

Frontrunning

```
function sellOrder(address nft, uint256 id, uint256 price) external {
    // transfer NFT to Market from Seller
    // save Order
}

function buyOrder(address nft, uint256 id) external {
    // send payment from Buyer to Seller
    // receive nft from Market
}
```

The Buy Transaction Can Be Frontrunned To Resell The NFT For A Higher Price.

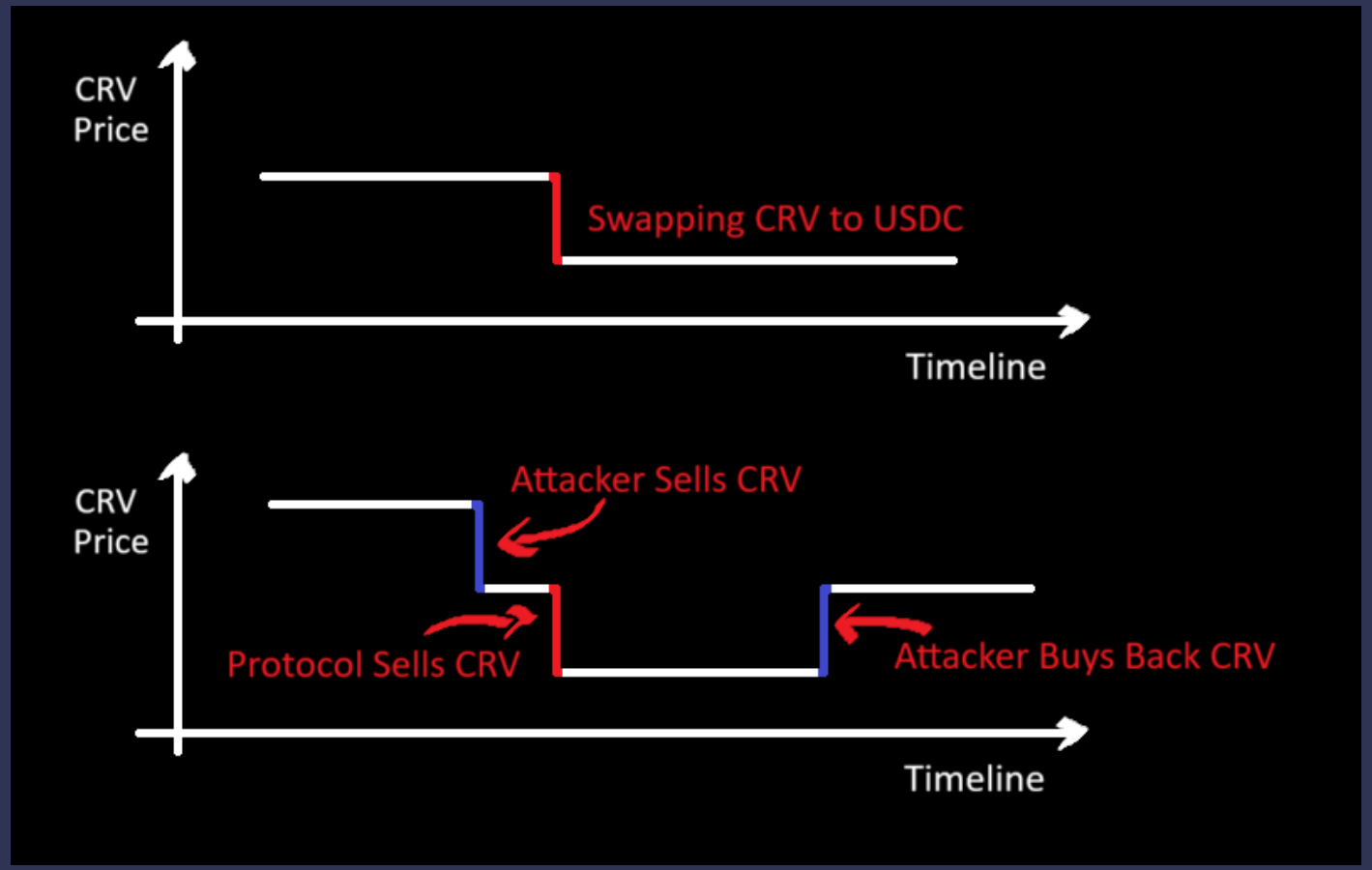
Common Vulnerabilities

Sandwich Attacks

```
ftrace | funcSig
function swapRewards() external {
    // swap curve rewards to underlying tokens using uniswap
}
```

Common Vulnerabilities

Sandwich Attacks



Sometimes, Can Be Exploited With A Flashloan

Potential Solutions:

- Controlling The Expected Return Amount
- Using TWAP (time-weighted average prices) Oracles

Common Vulnerabilities

Missing Authorization Checks

```
ftrace | funcSig
function swapForRewards(address _token, uint256 _amountIn) external returns (bool) {

}
```

```
function initialize(address _owner) external {
    if (owner != address(0)) revert CannotInitialize();
    owner = _owner;
    ...
}
```

Common Vulnerabilities

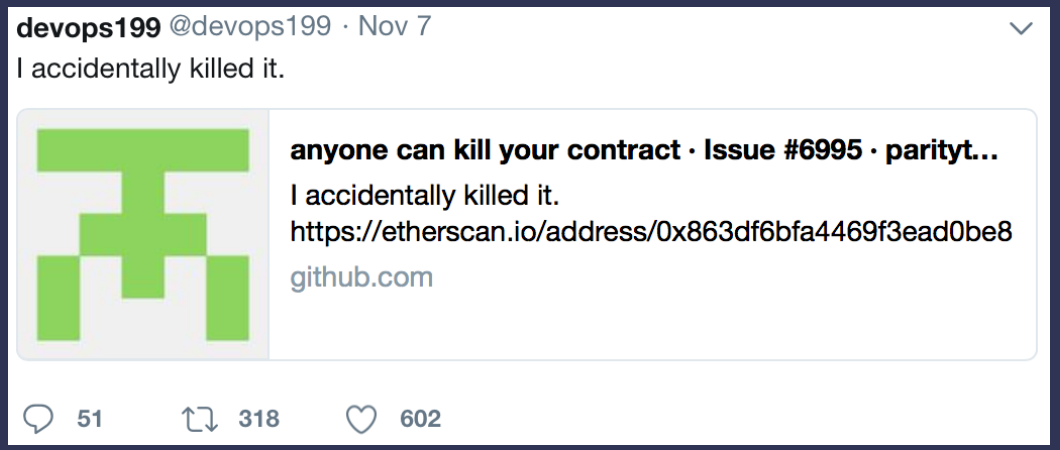
Missing Authorization Checks

```
ftrace | funcSig
function swapForRewards(address _token, uint256 _amountIn) external returns (bool) {

}
```

```
function initialize(address _owner) external {
    if (owner != address(0)) revert CannotInitialize();
    owner = _owner;
    ...
}
```

- Lack Of Documentation
- Missing / Improper Authorization
- Initialize Functions Are Not Correctly Protected



Devops199 „accidentally” evaporated $300 million

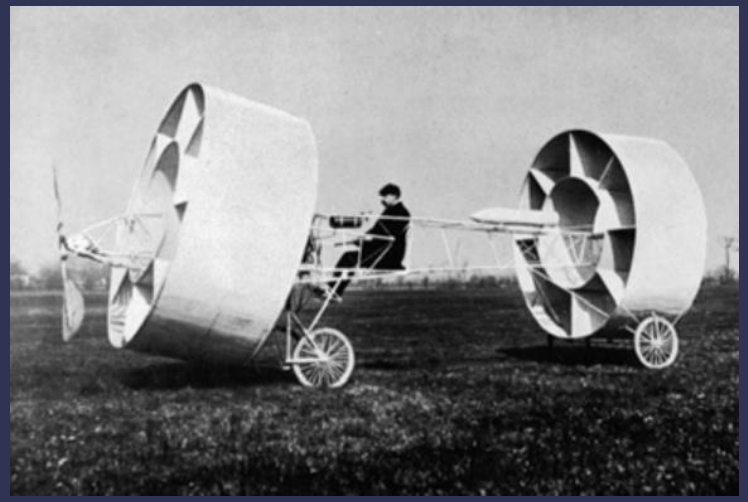
Common Vulnerabilities

Improper Implementation

```
ftrace | funcSig
function permit(address owner⬆, address spender⬆, uint256 value⬆, uint256 deadline⬆)
    public
    virtual
{
    if (owner⬆ == address(0)) revert AddressZero();
    if (block.timestamp > deadline⬆) revert SignatureExpired();

    allowance[owner⬆][spender⬆] = value⬆;
    nonces[owner⬆]++;
}
```

```
function unlock(address _to, uint256 _amount, uint8 _v,bytes32 _r, bytes32 _s) external {
    bytes32 hashData = keccak256(_to, _amount);
    address addr = ecrecover(hashData, _v, _r, _s);
    require(_isValidator(addr));
    to.transfer(_amount);
}
```



Common Vulnerabilities

Improper Implementation

```
ftrace | funcSig
function permit(address owner⬆, address spender⬆, uint256 value⬆, uint256 deadline⬆)
    public
    virtual
{
    if (owner⬆ == address(0)) revert AddressZero();
    if (block.timestamp > deadline⬆) revert SignatureExpired();

    allowance[owner⬆][spender⬆] = value⬆;
    nonces[owner⬆]++;
}
```

```
function unlock(address _to, uint256 _amount, uint8 _v,bytes32 _r, bytes32 _s) external {
    bytes32 hashData = keccak256(_to, _amount);
    address addr = ecrecover(hashData, _v, _r, _s);
    require(_isValidator(addr));
    to.transfer(_amount);
}
```

- Missing/Improper Signature Validation
- Missing Replay Attack Protection (nonce)
- Lack Of Signature Expiration/Invalidation Logic
- Domain Separator Cannot Be Regenerated
- Signature Malleability
- Lack Of Zero Address Check For Recovered Signature (ecrecover)

Common Vulnerabilities

Improper Integration

```
/**
 * @dev oracle to get the price in USD with 18 decimals
 * @param token the token address
 * @return the price in USD with 18 decimals
 */
ftrace | funcSig
function getUSDPrice(address token⬆) public view returns (uint256 priceInUSD) {
    uint256 decimals = ERC20(token⬆).decimals();

    AggregatorV3Interface priceFeed = IPriceFeed(priceFeedAddress).getPriceFeedFromAsset(token⬆);
    if (address(priceFeed) == address(0)) revert PriceFeedNotFound();
    (, int256 priceInUSDInt,,,) = priceFeed.latestRoundData();
    if (decimals < 18) {
        return uint256(priceInUSDInt) * 10 ** (18 - decimals - 2);
    }
    return uint256(priceInUSDInt) * 1e10;
}
```

Common Vulnerabilities

Improper Integration

```
/**
 * @dev oracle to get the price in USD with 18 decimals
 * @param token the token address
 * @return the price in USD with 18 decimals
 */
ftrace | funcSig
function getUSDPrice(address token⬆) public view returns (uint256 priceInUSD) {
    uint256 decimals = ERC20(token⬆).decimals();

    AggregatorV3Interface priceFeed = IPriceFeed(priceFeedAddress).getPriceFeedFromAsset(token⬆);
    if (address(priceFeed) == address(0)) revert PriceFeedNotFound();
    (, int256 priceInUSDInt,,,) = priceFeed.latestRoundData();
    if (decimals < 18) {
        return uint256(priceInUSDInt) * 10 ** (18 - decimals - 2);
    }
    return uint256(priceInUSDInt) * 1e10;
}
```

Not Checking For Down L2 Sequencer (Arbitrum)

Assuming Oracle Price Precision

Not Checking For Stale Prices

Only Handling 6 Or 18 Token Decimals

Common Vulnerabilities

Insecure External Calls

```
} else if (action == Constants.OPERATION_CALL) {
  (bytes memory returnData, uint8 returnValues) = _call(values[i], datas[i], value1, value2);

  if (returnValues == 1) {
    (value1) = abi.decode(returnData, (uint256));
  } else if (returnValues == 2) {
    (value1, value2) = abi.decode(returnData, (uint256, uint256));
  }
}
```

The protocol have an extra „call” functionality, which decodes input data to extract the callee address and call data and performs a call to the specified address with the assembled call data.

Common Vulnerabilities

Insecure External Calls

```
} else if (action == Constants.OPERATION_CALL) {
  (bytes memory returnData, uint8 returnValues) = _call(values[i], datas[i], value1, value2);

  if (returnValues == 1) {
    (value1) = abi.decode(returnData, (uint256));
  } else if (returnValues == 2) {
    (value1, value2) = abi.decode(returnData, (uint256, uint256));
  }
}
```

- Vulnerability allows to call any non blacklisted contracts with arbitrary data.
- The transaction is executed by the protocol.

Example exploit: call the transferFrom function in order to transfer tokens that were approved by the users to the attacker.

Thank You

Useful resources

- Common Vulnerabilities in Smart Contracts:
  https://github.com/istvanbohm/solidity-vulnerabilities

- Testing Smart Contracts with Brownie:
  https://github.com/istvanbohm/hacktivity2022

- Reading Reports From Audit Firms and Competitions.

- Reading About Blockchain Security Incidents
  https://rekt.news/