De.Fi

SECURITY BIBLE

How to Outplay 90% DeFi Users
That Don't Know the Basics of DeFi Security

www.de.fi

CONTENTS

A BRIEF HISTORY OF DEFI SCAMS AND HACKS

The Decentralized Finance or DeFi industry was developing slowly until the middle of 2020, when some significant innovations occurred and new capital flowed into the industry.

As momentum built, this attracted the attention of millions of people around the world who then dived into a whole range of DeFi investment opportunities. Unfortunately, so did a lot of scammers, who started exploiting smart contract vulnerabilities in order to drain off this wave of new capital for themselves. This led to tremendous losses for DeFi users.

In most cases, malicious actors created their own projects, gathered an audience and lured everyone into thinking their investments would result in gains of 10x or more. They developed smart contracts containing functions that acted as backdoors, allowing them to manipulate terms and steal user funds.

$154 million was lost in 2020 as a result of security vulnerabilities in smart contracts. This includes losses from malicious projects that had not been audited by independent third parties and legitimate projects that were exploited by hackers.

PRINCIPLES FOR ASSESSING DEFI PROJECTS

It's very difficult to pinpoint exactly what distinguishes a trustworthy project's smart contracts from those of a malicious one because scams are set up in different ways at different times. Context is key.

However, it is definitely possible to follow some general security principles that will help you to analyze smart contracts, assess a project's intentions and identify risks.

Your analysis should cover two main areas - smart contracts analysis and project analysis. Within these main areas, there are a range of things you need to look at, including everything from suspicious functions within smart contracts to potential token inflation and even what the project's social media presence can tell you about the team's motives.

All of this is covered in this handbook and you don't need to be a technical expert to read it. All you need is an interest in understanding how and why malicious actors would use certain techniques to scam investors, as well as a willingness to weigh these factors in context to decide how safe an investment opportunity really is.

SMART CONTRACT ANALYSIS

Every DeFi project is based on a complex system of smart contracts. By identifying the smart contracts that exist and then analysing some of the functions within them, you can build up a much clearer understanding of whether the project can be trusted.

Ownership

When a smart contract is owned by an externally owned account (EOA), you need to understand what functions the EOA can call. That's because it may be able to call functions that directly affect the security of user funds or the project's investment terms, such as token minting, ownership transfer, adding new minters or changing fees and reward rates.

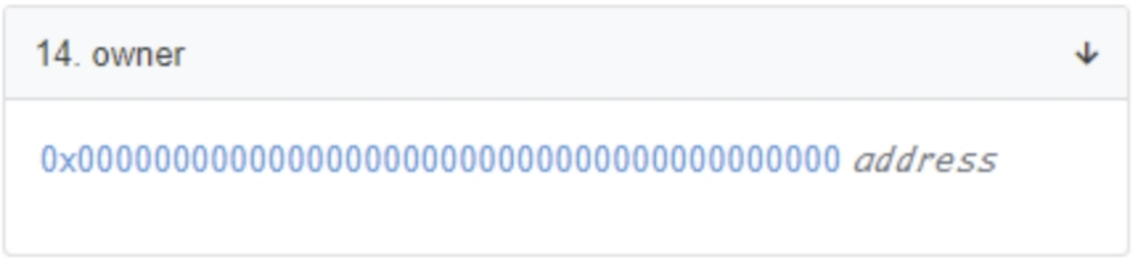
If an EOA does control a smart contract, you would need to trust this third party to act in your interests, which totally contradicts the concept of Decentralized Finance. You should be wary of projects where this is the case.

An example of where the contract owner can mint any number of tokens to his EOA might look like this:

```
function mint(address _address, uint256 _value) public{
    require(msg.sender == owner, "Only owner could mint");
    balances[_address] = balances[_address].add(_value);
    emit Transfer(0, _address, _value);
}
```

It is usually best to avoid centralized projects where a smart contract owner can impose their direct influence on key functions.

As a general rule, you should choose decentralized smart contracts that ensure smart contract owner addresses are burned (as indicated in the example below) during contract logic initializations, after all deployment procedures are finished.



The mint function

As with any function, the mint function doesn't represent any danger by itself. However, depending on the context in which it is used, it could pose the following risks:

- Infinite minting
- New minters created
- Token inflation

All of these situations can result in token distribution promises being broken and are explained in more detail below.

Infinite minting

Infinite minting represents a serious risk because it can be used by malicious actors to mint tokens and send them to a specific address. These minted tokens might then be sold off, leading to a crash in the token's value and liquidity being sucked out of all token pairs. If you are the victim of this, you will be left with tokens that are worthless and cannot be sold.

New minters

If an EOA can become a new minter or if an EOA can set any EOA (including itself) as a receiver of minted tokens, funds are endangered because tokens can be stolen and sold.

Here is an example of the addMinter function:

```
function _mint(address account, uint256 amount) internal onlyMinter {
    require(account != address(0), "ERC20: mint to the zero address");

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);

    emit Transfer(address(0), account, amount;
}
function _addMinter(address newMinter) external onlyOwner {
    minters[newMinter] = true; //or minters.push(newMinter);
}
```

Token inflation

Token inflation is a distinct possibility if there is no maximum token supply or if tokens can be minted without limitation or if token minting is not balanced with a token burning system.

In the example below, you can see how minting new tokens is

limited when the total token supply reaches a hard cap:

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount;

    _totalSupply = _totalSupply.add(amount);
    require(_totalSupply <= hardCap, 'hard cap reached!');
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}
```

The migrate function

This function may represent a serious risk to yield farmers because a migration can be used by scammers to move funds from a contract to a regular EOA address or to a separate centralized contract. This would put all users at risk of losing their funds.

Below, you can see a code example where the migration functionality is enabled:

```
function setMigrator(IMigratorChef _migrator) public onlyOwner {
    migrator = _migrator;
}

// Migrate lp token to another lp contract. Can be called by anyone.
function migrate(uint256 _pid) public {
    require(address(migrator) != address(0), "migrate: no migrator");
    PoolInfo storage pool = poolInfo[_pid];
    IERC20 lpToken = pool.lpToken;
    uint256 bal = lpToken.balanceOf(address(this));
    lpToken.safeApprove(address(migrator), bal);
    IERC20 newLpToken = migrator.migrate(lpToken);
    require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
    pool.lpToken = newLpToken;
}
```

Funds Lockup Period

Projects can introduce a funds lock period to prevent anyone, including users and their team members, from unstaking their tokens or selling them on an exchange for a certain period of time. If a funds lock period is implemented, make sure it satisfies your investment plans, as you won't have access to your funds until the period is over.

The pause function

This function allows its creator to pause a smart contract, even if you have funds staked in it. You will not be able to access your funds until the pause is over.

This might be risky if, for example, a smart contract vulnerability is revealed and funds need to be transferred to a safer place. It might also be an issue if the price of staked tokens fall on the open market and users need to unstake tokens in order to sell, as the pause function would mean they couldn't access their funds.

Moreover, scam projects can introduce the pause function with other conditions that allow them to transfer funds from the contract to their wallets.

An example of the pause function can be seen below:

```
function withdrawEther(address payable _dest) external onlyOwner whenNotPaused {
  uint256 _balance = address(this).balance;
  emit TokenWithdraw(ETH_TOKEN_PLACHOLDER, _balance, _dest);
  _dest.sendValue(_balance);
}

/**
 * @notice Pause contract
 */
function pause() external onlyOwner whenNotPaused {
  _pause();
}
```

```
/**
 * @notice Unpause contract
 */
function unpause() external onlyOwner whenPaused {
  _unpause();
}
```

Suspicious Functions

A smart contract may include suspicious functions that have been intentionally designed by the dev team to cheat users.

Poorly designed functions that can bring unexpected results should also be considered suspicious. Technical skills are required to review code and decide if any suspicious functions should be considered dangerous but some examples to look out for are given below.

Timelocks

The timelock contract or timelock conditions are added to certain functions in order to delay a process or transaction being performed on a blockchain. There are a few reasons why contract deployers use them and one of the main ones is controlling token sales on the open market.

A project might want to allow its developers to hold a big share of tokens that have been granted as a reward. It might also want to let investors purchase a large amount of tokens at an advantageous price at the start of a project's life. Both these situations pose a risk to the token price if these holders can sell their tokens at any time. Timelocks are applied to solve this problem, as they restrict parties from selling tokens before a certain deadline.

Timelocks can be used as a safety measure that is imposed on

smart contract functions in order to affect user funds, rewards, depositing or withdrawing terms and other conditions. However, timelocks don't guarantee user safety, as they only give users time to react to important changes.

Yield farmers should use special Telegram bots to track changes and, if required, perform the necessary procedures to secure their funds before changes take place. This may include unstaking funds or withdrawing tokens to a safe place.

Token location

Always check where funds deposited by users and user rewards are stored. They could be held by one EOA that is owned by the project dev team or within a suspicious smart contract that gives the team complete control over the funds.

When tokens are under centralized control such as this, the dev team can move them anywhere and, in the worst case scenario, steal them.

Code verification

You should check whether the analyzed smart contract code has been verified by Etherscan, as this reflects whether the contract bytecode matches that which is on the blockchain.

Token Distribution

If there are EOAs holding large amounts of the overall token supply (for example, one private owner has more than 15%), there is a risk they will sell their tokens and the price will dump. Untrustworthy projects may send a large amount of tokens to team members during a presale or airdrop, which may result in tokens being sold off in this way.

PROJECT ANALYSIS

As well as the technical analysis of smart contracts, you should also consider a range of project features and characteristics when trying to identify scams and malicious actors.

While some of these relate to the smart contract functions already explored, others concern what the project is trying to achieve, who it is run by and how they manage both internal operations and external communications.

Governance

If a project has implemented a decentralized governance system, users can participate in its decision making and this provides a much safer investment environment. If users are eligible to participate in governance, it's important to check if the voting proceeds on-chain or off-chain.

In off-chain governance, a decision is approved at the community level through voting and is then integrated into the protocol's code by the dev team. In on-chain governance, voting rules are hardcoded into the protocol, so any changes that are voted for are

bridged with the code and performed automatically.

Documentation

The quality of a project's technical documentation can say a lot about how safe it is and whether its creators' intentions are good or bad.

It's important to check if a project has a whitepaper or another technical document describing its goals, functionality and its system of smart contracts. If these documents only contain some general descriptions and don't explain the project's value or technical specifications, this may indicate that they were developed as a formality.

You should also assess the completeness and usefulness of code commenting. All functions and variables must contain comments and the comments should be helpful and easy to understand.

In general, all code released to the public should be clear and concise, so that users can read it easily and understand what goes on under the hood of the project. Intentionally complicated code may be used to hide backdoors and other malicious functionality.

Finally, all basic smart contract functions must be documented either on the project's website or in its GitBook or GitHub.

Team and development history

The project team's positioning, behaviour and development history can help you understand if the project is open, community-oriented and likely to react quickly to external warnings about any mistakes it has made.

The key areas to assess in this regard are as follows:

Is it an open or an anonymous project team?
There have been many famous and successful projects built by anonymous developers. However, it's usually best if the project is public, so that every team member can be verified. In general, it's always safer to choose public projects over anonymous ones.

Does the project include an open software repository?
Can smart contracts be easily found on the project's website, its GitBook, or in the README of its software repository?

What is the development history of the project?
Check how the repo is, how many devs are commiting to the repo and with what frequency.

How experienced are the project team members?
When a project is public, you should check that every team member has good experience and skills. Inexperienced teams may create products that have security vulnerabilities.

Do you trust the reputations of the team members?
If the project team has worked on another project in the past, it's important to check whether it was successful or abandoned, scammed or managed inefficiently.

Social media presence

Projects that care about their community are always good communicators and publish frequent development updates. A project's social media, blog, website and other communication channels should be active and regularly updated.

Avoid projects that don't provide complete information and ignore user questions or requests. If a project offers strange promotions or

makes suspicious promises, these are warning signs that it should be avoided.

Mixers for smart contract deployment

Untrustworthy projects can use mixers like Tornado cash for money laundering, as well as to pay for deployment of smart contracts, in order to cover their tracks.

Uniqueness

Before investing in a new project, try to understand the yield farming approach it offers and what value it brings. If there are no innovative ideas behind the project and it doesn't offer any improvements compared to established DeFi projects, there is usually little reason to invest.

AUTOMATING YOUR DEFI SECURITY ANALYSIS

DEFIYIELD will release an automatic smart contract scanning system called Safe soon. It is based on machine learning technology and is capable of auditing smart contracts deployed on Ethereum, Binance Smart Chain and other chains.

We designed Safe because we know that manual smart contract verification is not easy. By using it, you can simply enter a smart-contract address and see a highly accurate security score that the system produces from an in-depth analysis.

Even when you start using DEFIYIELD Safe though, you should always do your own due diligence and review smart contracts audits that have been conducted by professionals.

Remembering to follow a whole range of security checks is important because no single safety solution is enough. For example, even when a project has been audited by professionals, it doesn't mean the project is completely safe.

It is still possible for users to be scammed by a project after an audit has been performed. That's because the audit is only correct on the day it was published and a team can introduce malicious code changes afterwards.

You can learn more by reviewing the security audits that DEFIYIELD has performed on various yield farming projects. Plus, if you want to check whether a project you are interested in is trustworthy, you can request a DEFIYIELD audit. Once enough people in the community have requested an audit, we will go ahead with it.

And finally…

Don't forget to join the best yield farming community via your preferred channel:

Telegram: https://t.me/defiyield_app
Twitter: https://twitter.com/defiyield_app