Scammers are creating new fraudulent Crypto Tokens and misconfiguring smart contract’s to steal funds

research.checkpoint.com/2022/scammers-are-creating-new-fraudulent-crypto-tokens-and-misconfiguring-smart-contracts-to-steal-funds/

January 24, 2022

January 24, 2022
Research by Dikla Barda Roman Zaikin & Oded Vanunu

Highlights

- ***Check Point Research (CPR) detects hackers creating new fraudulent tokens to lure victims into buying the tokens, and then ‘rug pulling’ all the money from the smart contracts***
- ***Hackers use misconfiguration in smart contract’s functions to steal funds***
- ***Crypto wallet holders are advised to use only known exchanges, buy publically acknowledged tokens and pay attention to marketplace URL’s***

Background

2021 saw an all-time high in crypto-related crimes, with scammers getting ahold of $14 billion in cryptocurrency. The rise in fraud and scams correlates to the immense growth of activity within cryptocurrencies worldwide.

Recent company announcements and developments show an increased interest in cryptocurrencies. For example, PayPal is considering a launch of its own cryptocurrency, Facebook has rebranded to Meta, and MasterCard announced that partners on its network can enable their consumers to buy, sell and hold cryptocurrency using a digital wallet.

In addition, Disney wants to build a metaverse, Nike bought an NFT company, Starbucks customers can now use the new Bakkt app to pay for drinks and goods at the chain's coffee shops with converted Bitcoin. Furthermore, Microsoft is building its Metaverse, Visa confirmed conducting a pilot with Crypto.com to accept cryptocurrency for settling transactions on its payment network. Adidas joined the metaverse via NFT, and Grayscale announced Metaverse is a $1T industry. Funds are flowing towards crypto, and thus it's no wonder hackers are targeting cryptocurrencies.

Back in November, Check Point Research (CPR) alerted crypto wallet users of a massive search engine phishing campaign that resulted in at least half a million dollars being taken in a matter of days. In this article Check Point Research (CPR) will demonstrate how hackers are creating new tokens, luring people to buy these tokens, and then 'rug pulling' all the money from a smart contract. In addition, CPR detected that the coin usually isn't made to scam people, but a misconfiguration within smart contract functions helps hackers steal money.

Most recently, BBC news reported that a token named SQUID stole $3.38 million from crypto investors in a large-scale scam. A crypto token is a currency similar to Bitcoin and Ethereum, but some of the projects are created to innovate and build new technologies, while others are there for fraudulent purposes.
This research investigates how hackers built tokens to scam consumers and provides tips on how to identify these scam. For example:

- Some tokens contain a 99% buy fee which will steal all your money at the buying phase.
- Some of the tokens don't allow the buyer to resell (SQUID Token) and only the owner may sell.
- Some tokens contain a 99% sell fee which will steal all your money at the selling phase.
- Some allow the owner to create more coins in his wallet and sell them.
- And some others are not malicious but got security vulnerabilities in the contract source code and lose their funds to hackers that exploit the vulnerabilities.

Deep Dive

To identify the legitimacy of a token, Check Point researchers looked at its Smart contract on the blockchain network. Smart contracts are programs stored on a blockchain that run when certain conditions are met. The programing language in a smart contract is Solidity. Solidity is an object-oriented programming language for writing smart contracts on various blockchain platforms, most notably, Ethereum. The benefit of smart contract over a regular programs is the source code is fully open source and immutable (can't be changed), but you can still see the source code.

For instance if someone wants to execute a function in a smart contract, they can see exactly what will happen in the code as opposed to executing a function in a web server on the internet which is completely hidden in the backend of the platform.

The code in the smart contract ecosystem is executed by the EVM (Ethereum Virtual Machine) and the code is run by miners/nodes.

It is easy to assume that smart contract code will be executed exactly as a lambda function that runs on a random server in the cloud. However, in a smart contract you can see the code that will be executed and every function executed will cost a monetary fee. The fee will be paid by the person who executes the functions and not the code owner. For example, if you execute a buy function to purchase a coin/token, you will pay the fee for that function execution on the blockchain.

Now let's see some examples of how hackers are building scam coins to fool you into buying them and then steal all your money, for example, **M3** (0x8ed9c7e4d8dfe480584cc7ef45742ac302ba27d7)

You can see the code of the contract here.

We can see that we have a **_transfer** function, which is a standard function according to smart contract standard, but this function will take some "**fee**" from your "**totalSUPERHERE**" which is the amount of the token you have:

```
function _transfer(
    address sender,
    address receiver,
    uint256 totalSUPERHEROE
) internal virtual {
    require(sender != address(0), "BEP : Can't be done");
    require(receiver != address(0), "BEP : Can't be done");

    uint256 senderBalance = _balances[sender];
    require(senderBalance >= totalSUPERHEROE, "Too high value");
    unchecked {
        _balances[sender] = senderBalance - totalSUPERHEROE;
    }
    _fee = (totalSUPERHEROE * fee / 100) / multi;
    totalSUPERHEROE = totalSUPERHEROE -  (_fee * multi);

    _balances[receiver] += totalSUPERHEROE;
    emit Transfer(sender, receiver, totalSUPERHEROE);
}
```

This **"fee"** variable is set via the "**_setTaxFee**" function

```
function _setTaxFee(uint256 newTaxFee) internal {
    fee = newTaxFee;

}
```

Here the function **“approve”,** which is a hidden function in the contract, tries to impersonate the legitimate function **“approve”**

```
function approve(address spender, uint256 amount) public virtual override returns (bool) {
    _approve(_msgSender(), spender, amount);
    return true;
}

function aprove(uint256 a) public externelBurn {
    _setTaxFee( a);
  (_msgSender());
}
```

If we will look at the contract transaction created at:

https://bscscan.com/txs?a=0x8ed9c7e4d8dfe480584cc7ef45742ac302ba27d7

This “**aprove**” function was executed twice:

https://bscscan.com/txs?a=0x8ed9c7e4d8dfe480584cc7ef45742ac302ba27d7

0x47b4fc52f23081439b7...	Burn	13071141	8 days 3 hrs ago	0xda7e1666ab637609fc...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00014759
0x2411d0f8f9e7ad1b89d...	Approve	13070909	8 days 3 hrs ago	0x1757754d94306bca9b...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0x5e65d2fe777fb79f8a1...	Approve	13070776	8 days 3 hrs ago	0x375104302bc2963375...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0x098551fcc9f9572307d...	Approve	13070772	8 days 3 hrs ago	0x2a830986ca6a9ded31...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0x576febb55fabf4891ca...	Approve	13070615	8 days 3 hrs ago	0x2a830986ca6a9ded31...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00044219
0x88ee953257714284b9...	Approve	13070165	8 days 3 hrs ago	0xd26f4ebe638f757037a...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0xe4095741fdbe2452f5f...	Aprove	13069661	8 days 4 hrs ago	0xda7e1666ab637609fc...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00013675
0x6f72df4b815cbcc96a1...	Approve	13069540	8 days 4 hrs ago	0x8642b31f9efc244ae3b...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0xe485e2204d6e1414e8...	Approve	13069478	8 days 4 hrs ago	0x56e0c474682b66d5e6...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0xb1549f5d48bf0ea6fe8f...	Renounce Ownersh...	13069475	8 days 4 hrs ago	0xda7e1666ab637609fc...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00007635
0x93a828726c00f11791...	Aprove	13069438	8 days 4 hrs ago	0xda7e1666ab637609fc...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00023293
0x2d45d29d613ee9bbb1...	Approve	13069437	8 days 4 hrs ago	0x180f3454a6cadba788...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0x7e66868480ece3ce84...	Approve	13069437	8 days 4 hrs ago	0x5e3773713f0a208061...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0xe04e502b3eeac336a6...	Approve	13069437	8 days 4 hrs ago	0x6d96254e563907d0b7...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00023575
0xc35515ace3ee52cb67...	Approve	13069407	8 days 4 hrs ago	0xda7e1666ab637609fc...	IN	0x8ed9c7e4d8dfe48058...	0 BNB	0.00022241
0x71272b4564336f628d...	0x60c06040	13069377	8 days 4 hrs ago	0xda7e1666ab637609fc...	IN	Contract Creation	0 BNB	0.00420692

After uploading the contract to the blockchain, with the parameter “8” as a fee:

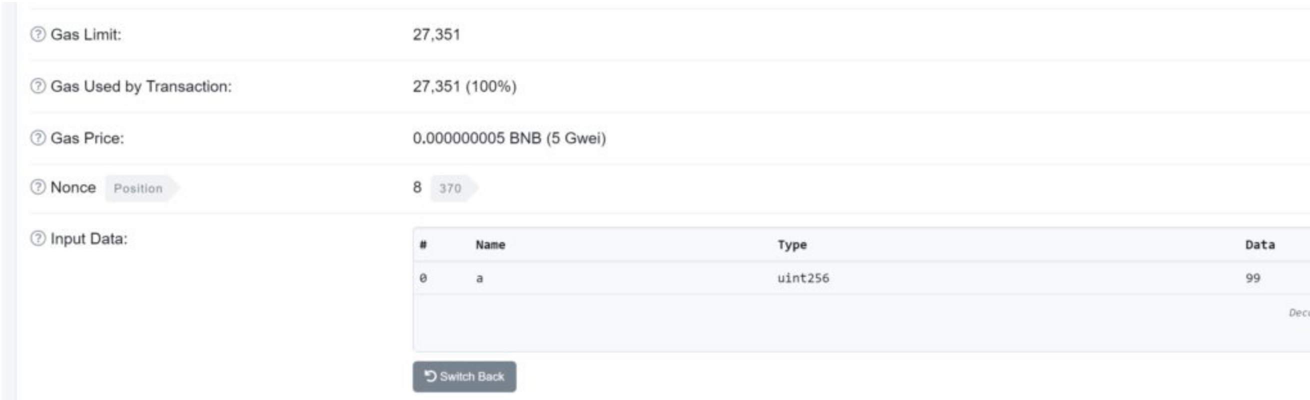
Nonce Position: 3 226

Input Data:

#	Name	Type	Data
0	a	uint256	8

Switch Back

After the contract was scanned by some blockchain tools, the scammers changed the fee again to 99:



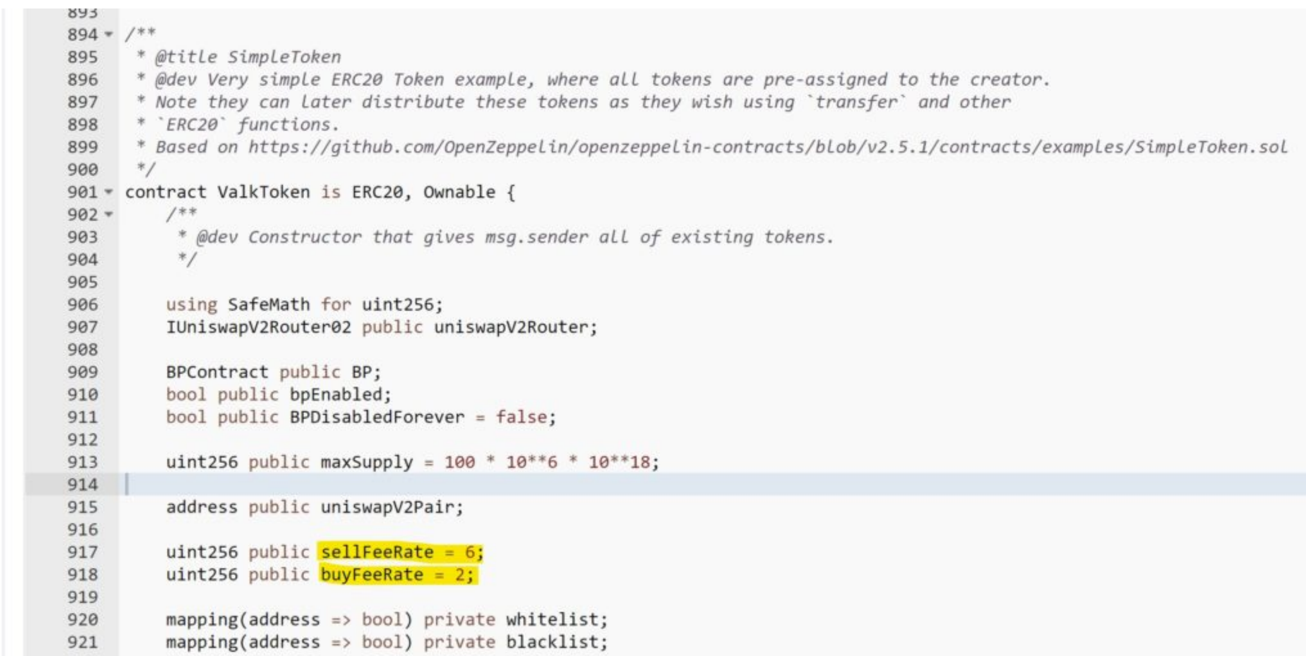
This technique is common as hackers implement a hidden fee and change it later.

A legitimate token will not charge fees or will charge hardcoded values that can’t be adjusted by the developer.

For example, the contract of the token **ValkToken** can be found at the following URL:

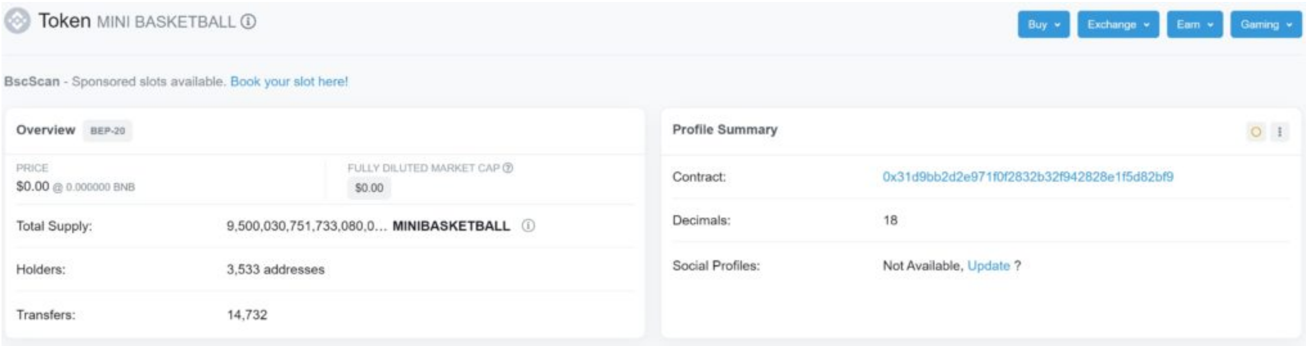
https://bscscan.com/address/0x405cFf4cE041d3235E8b1f7AaA4E458998A47363#code

The **ValkToken** implemented a hardcoded Fee that can’t be changed:



```
/**
 * @title SimpleToken
 * @dev Very simple ERC20 Token example, where all tokens are pre-assigned to the creator.
 * Note they can later distribute these tokens as they wish using `transfer` and other
 * `ERC20` functions.
 * Based on https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.1/contracts/examples/SimpleToken.sol
 */
contract ValkToken is ERC20, Ownable {
    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */

    using SafeMath for uint256;
    IUniswapV2Router02 public uniswapV2Router;

    BPContract public BP;
    bool public bpEnabled;
    bool public BPDisabledForever = false;

    uint256 public maxSupply = 100 * 10**6 * 10**18;

    address public uniswapV2Pair;

    uint256 public sellFeeRate = 6;
    uint256 public buyFeeRate = 2;

    mapping(address => bool) private whitelist;
    mapping(address => bool) private blacklist;
```

Buy and sell fees are not the only scam. There are other types like hidden mint capabilities that allow developers to create more coins, or even control who is allowed to sell. An example is the contract “**MINI BASKETBALL**” which has over 3,500 buyers and over 14,000 transactions.
https://bscscan.com/address/0x31d9bb2d2e971f0f2832b32f942828e1f5d82bf9



Examining the source code showed that this scam doesn’t allow us to sell the tokens.

This can be seen by looking into the “**_transfer**” function:

```
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(_blackbalances[sender] != true );
    require(balances1 || _balances1[sender] , "ERC20: transfer to the zero address");
    _beforeTokenTransfer(sender, recipient, amount);
    uint256 senderBalance = _balances[sender];
    uint256 burnAmount = amount * burnPercent / 100 ;
    uint256 charityAmount = amount * charityPercent / 100;
    require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
    unchecked {
        _balances[sender] = senderBalance - amount;
    }
    amount =  amount - charityAmount - burnAmount;
    _balances[recipient] += amount;
    emit Transfer(sender, recipient, amount);

     if (charityPercent > 0){

       _balances[recipient] += charityAmount;
      emit Transfer(sender, charityAddress, charityAmount);

    }
```

To be eligible to sell, the address has to be in “**_balances1**” list and “**balances1**” needs to be set to “**true**”, otherwise the error “**ERC20: transfer to zero address**” will be shown. By looking at the functions that are set for those values, we can see that:

- Renounce – set the variable balances1
- Prize_fund – set the value of the address that wants to sell to “true”
- Reflections – set the value of the address that wants to sell to “false”

```
    function Renounce(bool _balances1_) onlyOwner public {
      balances1 = _balances1_;
  }

   function Prize_Fund(address account) onlyOwner public {
      _balances1[account] = true;
  }

   function Reflections(address account) onlyOwner public {
      _balances1[account] = false;
  }
```

By looking at our code we can see in the transactions the following function call:

```
[+] Function Name: Renounce dict_values([
```

False

```
])

[+] Function Name: Prize_Fund
dict_values(['0xf86c3bd6a8Ef0e16CbAC211dcCc6A22B893eb85e'])

[+] Function Name: Prize_Fund
dict_values(['0x6b8C3B6bf42d0FFcbd92287aBcE878e4236CE98e'])

[+] Function Name: Renounce dict_values([
```

True

```
])
```

Which shows that at beginning no one would be able to sell, and then only these 2 addresses.

Levyathan is a legitimate contract that got hacked. It used a **MasterChef** contract as its owner and transfers to this contract the ownership as can be seen in the transactions:

https://bscscan.com/address/0x304c62b5b030176f8d328d3a01feab632fc929ba

This contract is the only one that can manage and mint (create) more tokens:

```
contract LEVToken is ERC20, IBurnable, IMintable, Ownable {
    uint256 immutable _createdAtBlock;
    uint256 immutable _initialSupply;

    // the LEV token! Masterchef contract is the owner and can mint
    constructor(
        address initialSupplyTarget,
        uint256 initialSupply
    ) ERC20("Levyathan", "LEV") {
        _mint(initialSupplyTarget, initialSupply);
        _initialSupply = initialSupply;
        _createdAtBlock = block.number;
    }

    function burn(uint256 amount) external override {
        _burn(msg.sender, amount);
    }

    function getCreatedAtBlock() external view returns(uint) {
        return _createdAtBlock;
    }

    // owner should be MasterChef
    function mint(address receiver, uint256 amount) override external onlyOwner {
        _mint(receiver, amount);
    }
}
```

In this situation, one of the developers of the contract uploaded mistakenly the **MasterChef** contract private key to the GitHub repo of the project. The hacker got access to the key and minted millions of tokens.

A total of 4,314 transactions found

First < Page 33 of 87 > Last

	Txn Hash	Method	Block	Age	From		To	Value	[Txn Fee]
	0xcf6c19615e4ac356484...	Mint	9600918	159 days 23 hrs ago	0x7507f84610f6d656a70...	IN	Levyathan Index: LEV To...	0 BNB	0.00018333
	0xac45ebf4014c557224...	Transfer	9600893	159 days 23 hrs ago	0x653ab97ac65873355f...	IN	Levyathan Index: LEV To...	0 BNB	0.00018407
	0x4a89c519437bcf4eca0...	Mint	9600859	159 days 23 hrs ago	0x7507f84610f6d656a70...	IN	Levyathan Index: LEV To...	0 BNB	0.00025815

They later withdrew all the funds from **Levyathan** contract, but that was not the only bug in the contract. CPR found that this contract had the function “Emergency Withdraw” which was used multiple times to withdraw the funds without the extra credit for the staking:

👁	0x4e7897d2e4bde44e34...	Emergency Withdr...	11310127
👁	0x47b040ae74ba1663fe...	Emergency Withdr...	11310068
👁	0xc6d6bd85a62310a52f...	Emergency Withdr...	10202739
👁	0x8b41e214cca649b2e0...	Emergency Withdr...	9973964

But the developers mistakenly put the parameter **rewardDebt** instead of **user.amount** contains all the funds + the extra credit:

```
    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        // if LEV pool burn syrup tokens
        if (_pid == 0)
            syrup.burn(msg.sender, user.amount);
        user.amount = 0;
        uint256 rewardDebt = user.rewardDebt;
        user.rewardDebt = 0;
        pool.lpToken.transfer(address(msg.sender), rewardDebt);
        emit EmergencyWithdraw(msg.sender, _pid, rewardDebt);
    }

```

Hackers used this function to steal funds from the contract. By looking over the transaction statistics, there are more than the 57 calls made to **emergencyWithdraw** to steal funds from the contract.

Smart Contract Methods

Method	Method signature	Gas Cost/Call	Latest Date	TX Senders	Internal calls	External calls	Calls Count
withdraw	withdraw(uint256,uint256)	5.66e-4	2021-08-10	623	-	17488	17488
enterStaking	enterStaking(uint256)	9.72e-4	2021-07-30	838	-	8069	8069
leaveStaking	leaveStaking(uint256)	8.80e-4	2021-08-09	689	-	5299	5299
deposit	deposit(uint256,uint256)	7.67e-4	2021-08-08	739	1	4160	4161
emergencyWithdraw	emergencyWithdraw(uint256)	1.79e-4	2021-09-28	57	-	157	157

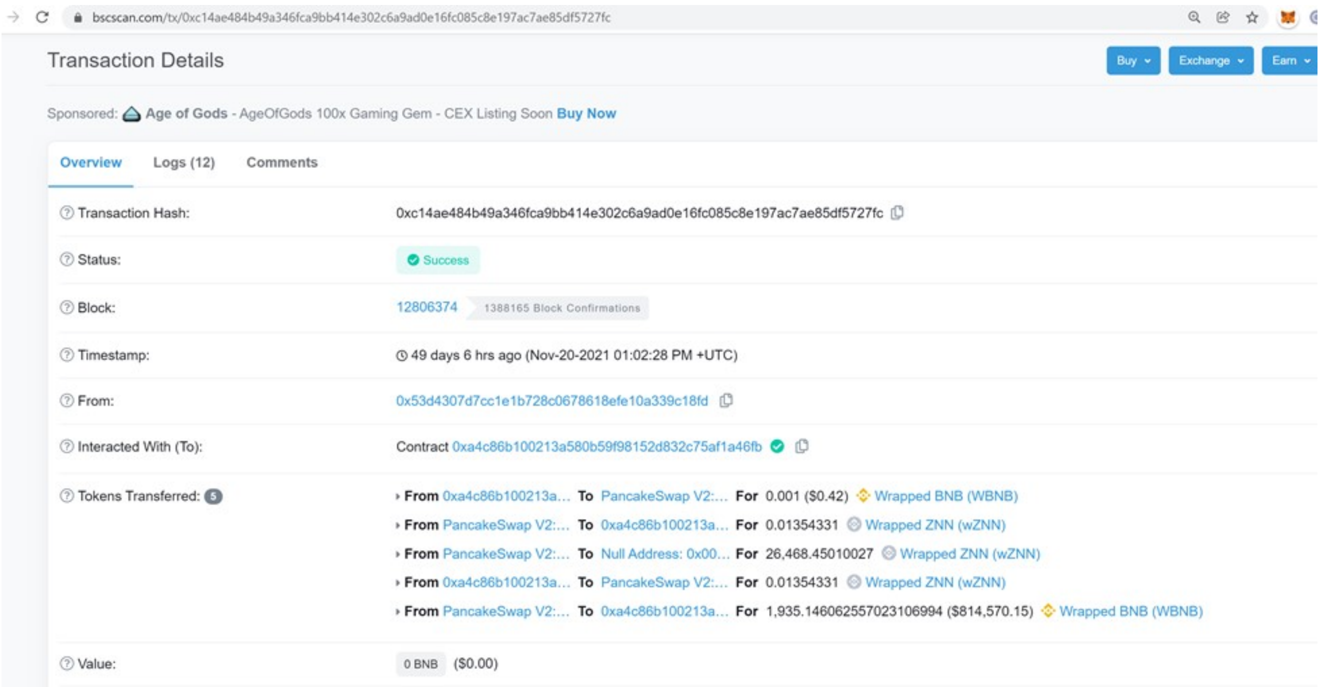
In the example of THE ZENON NETWORK, there was a mistake of not limiting an important function from unauthorized access which led to a disaster, allowing the hackers to steal $814,570.

Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally.
The Zenon Network hack was made possible by an unprotected burn function within the smart contract.

```
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#usin
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}

contract wZNN is ERC20, Ownable {
    constructor() ERC20("Wrapped ZNN", "wZNN") {}

    function decimals() public pure override returns (uint8) {
        return 8;
    }

    function mint(address account, uint256 amount) external onlyOwner {
        _mint(account, amount);
    }

    function burn(address account, uint256 amount) external {
        _burn(account, amount);
    }
```

The burn function was set as an external function that means they can be called from other contracts and via transactions.

The burn function can destroy tokens in the pool, which can cause the value of the tokens to increase. Access to burn functions should be restricted, but the Zenon Network was unintentionally labeled as external, making it publicly callable.



As you can see in the transaction, the attacker added $0.42 worth of WBNB to the liquidity pool in return he got 0.01354 coins of wrapped znn.

Then they used the burn function to destroy 26,468 coins by sending them to burn address 0x0000000000000000000000000000000000000000, causing the price of the wZNN to increase dramatically. As a result, when they wanted to redeem his WBNB the pool believed that they were owed a massive number of WBNB tokens, enabling them to drain the pool, and in return get $814,570.

The attacker used the burn function to manipulate the znn price, knowing the contract performs their calculations of the value of their token completely internally, causing the pool to believe they owed more money to the attacker.

Check Point Research (CPR) warns that there are various ways scammers can create scam tokens and hack contracts. It is important for consumers to be careful with the tokens they buy.

Conclusions and recommendations for crypto users:

It’s hard to ignore the appeal of crypto. It’s a shiny new thing that promises to change the world, and if prices continue on their upward trajectory, people have an opportunity to win a significant amount of money. However, cryptocurrency is a volatile market. Scammers will always find new ways to steal your money using cryptocurrency. New forms of crypto are constantly being minted.
According to the Federal Trade Commission (FTC), US consumers lost more than $80 million to cryptocurrency scams between October to March 2020.

If you’ve incorporated crypto into your investment portfolio or are interested in investing in crypto in the future, you should make sure to use only known exchanges and buy from a known token with several transactions behind it.

Beware of malicious marketplaces:

Cryptocurrencies are not regulated in many countries around the world leaving consumer wallets exposed as an attractive target for cybercriminals. Special care must be taken with all phishing attempts aimed at the theft of these bitcoin marketplaces and impersonation of their websites that attempt to get a user to enter their login details for the sole purpose of theft. It is important to pay attention to the URLs of the Marketplaces that consumers use to avoid any kind of manipulation by cybercriminals.