Blockchains & Cryptocurrencies

Smart Contracts IV & DeFi

Instructor: Matthew Green & Abhishek Jain

Spring 2023

News?

News?

BRIAN QUARMBY

FEB 25, 2023

Jump Crypto and Oasis.app ‘counter exploits’ Wormhole hacker for $225M

The asset retrieval came after the High Court of England and Wales ordered Oasis.app to work with Jump Crypto to recover the stolen funds.

15380 Total views **60** Total shares Listen to article 2:40

TECH / SECURITY / POLICY

Wormhole cryptocurrency platform hacked for $325 million after error on GitHub

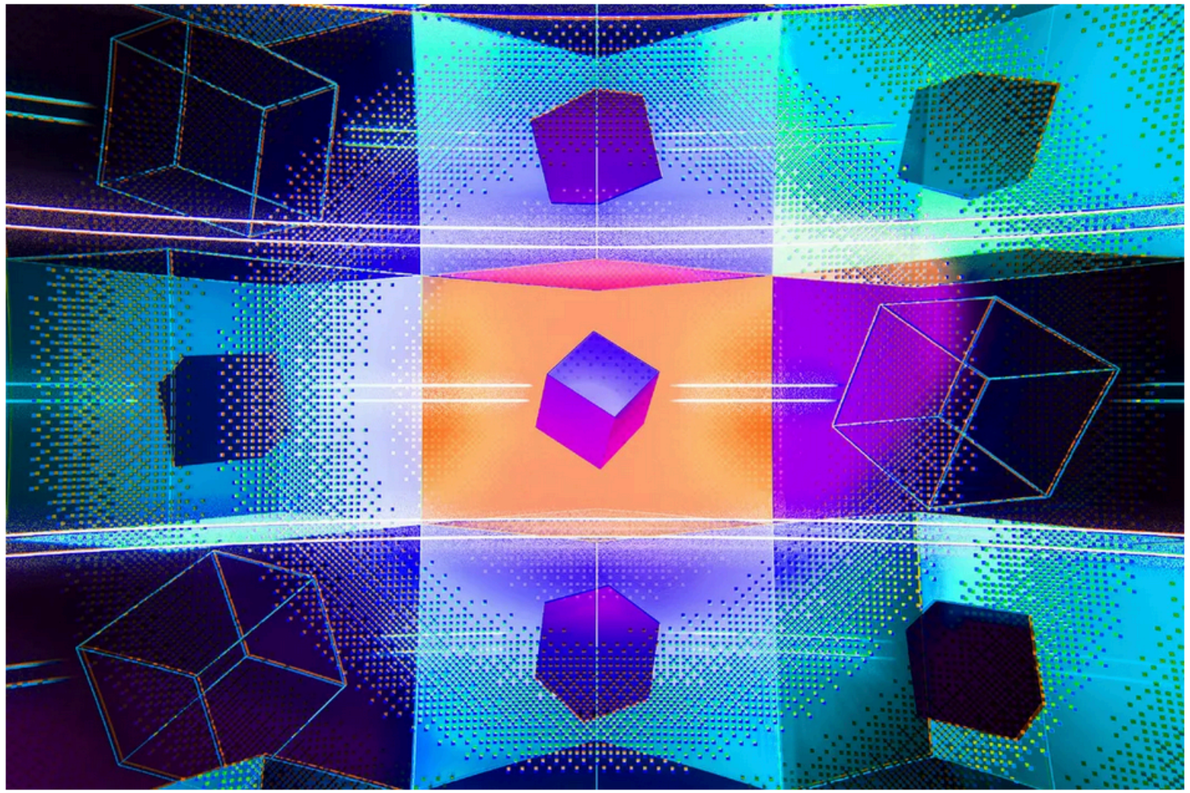


Illustration by Alex Castro / The Verge

/ A security flaw was fixed but seemingly not applied to the live application before it was hacked

By CORIN FAIFE / @corintxt

Feb 3, 2022, 12:43 PM EST | 0 Comments / 0 New

On Wednesday, the decentralized finance (DeFi) platform Wormhole became the victim of the largest cryptocurrency theft this year — and among the top five largest crypto hacks of all time

News?

Shortly after the attack, the Wormhole team also offered the hacker a $10 million bounty to return the funds, which was embedded as text in a transaction sent to the attacker's Ethereum wallet address.

Transaction Hash:	0x2d8b7901bff18ae6abe1a50aebe44b70559f39ff357b21340843d368b9486859
Status:	Success
Block:	14128723	2591733 Block Confirmations
Timestamp:	389 days 19 hrs ago (Feb-02-2022 08:15:31 PM +UTC) | Confirmed within 30 secs
Sponsored:	
From:	0x96D13cbEFFE7BAE169B9032Fe69Ed56eB07b300F (Wormhole: Deployer 14)
To:	0x629e7Da20197a5429d30da36E77d06CdF796b71A (Wormhole Network Exploiter)
Value:	0 ETH ($0.00)
Gas Limit & Usage by Txn:	25,912 | 25,912 (100%)
Gas Fees:	Base: 139.102935296 Gwei | Max: 190.464825111 Gwei | Max Priority: 190.464825111 Gwei
Burnt & Txn Savings Fees:	Burnt: 0.003604435259389952 ETH ($5.96) | Txn Savings: 0 ETH ($0.00)
Other Attributes:	Txn Type: 2 (EIP-1559) | Nonce: 79 | Position In Block: 72
Input Data:	This is the Wormhole Deployer: We noticed you were able to exploit the Solana VAA verification and mint tokens. We 'd like to offer you a whitehat agreement, and present you a bug bounty of $10 million for exploit details, and returning the wETH you 've minted. You can reach out to us at contact@certus.one

View Input As

Wormhole 🌪️ ✔

@wormholecrypto · **Follow**

The wormhole network was exploited for 120k wETH.

ETH will be added over the next hours to ensure wETH is backed 1:1. More details to come shortly.

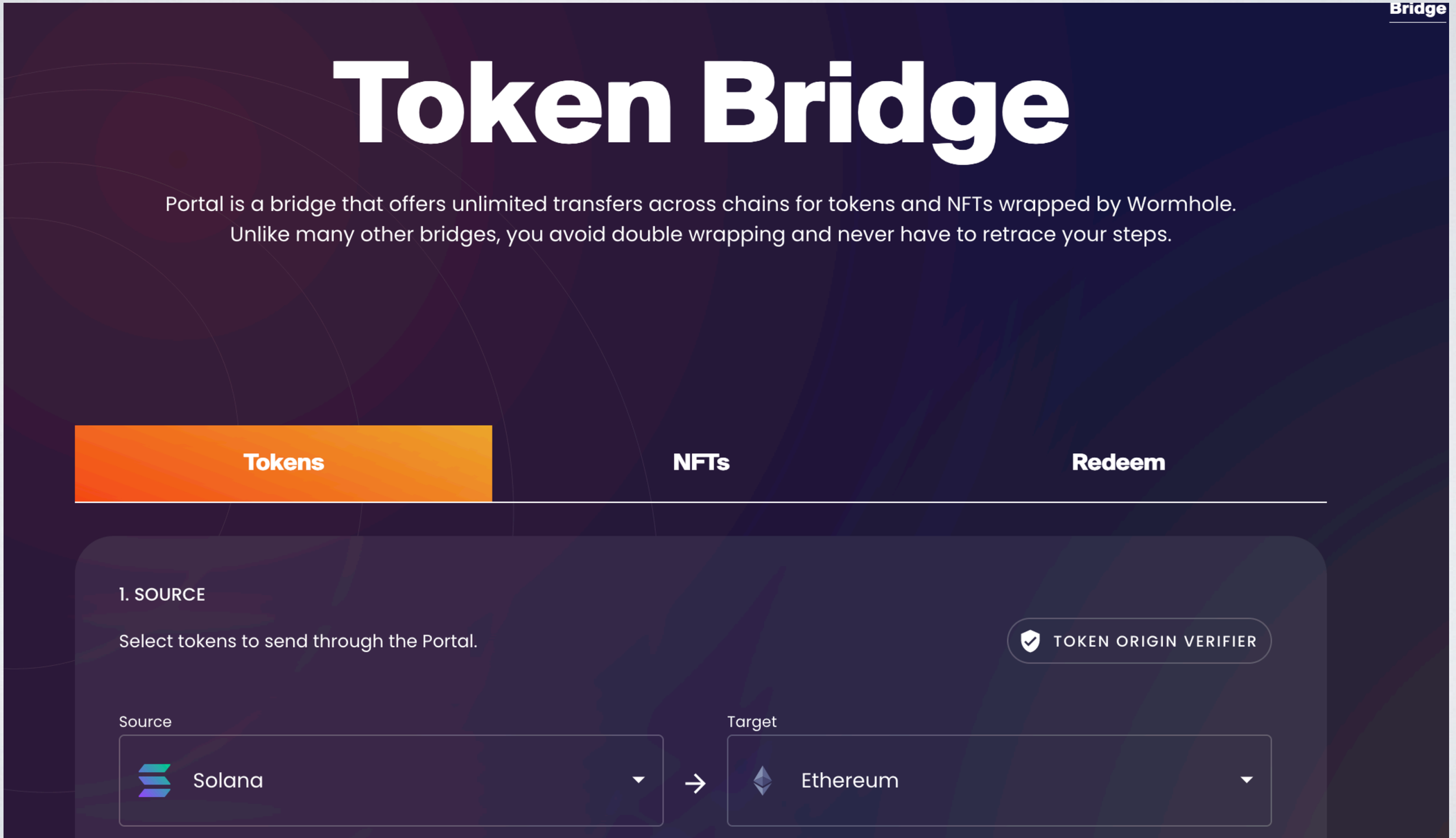
We are working to get the network back up quickly. Thanks for your patience.

5:25 PM · Feb 2, 2022

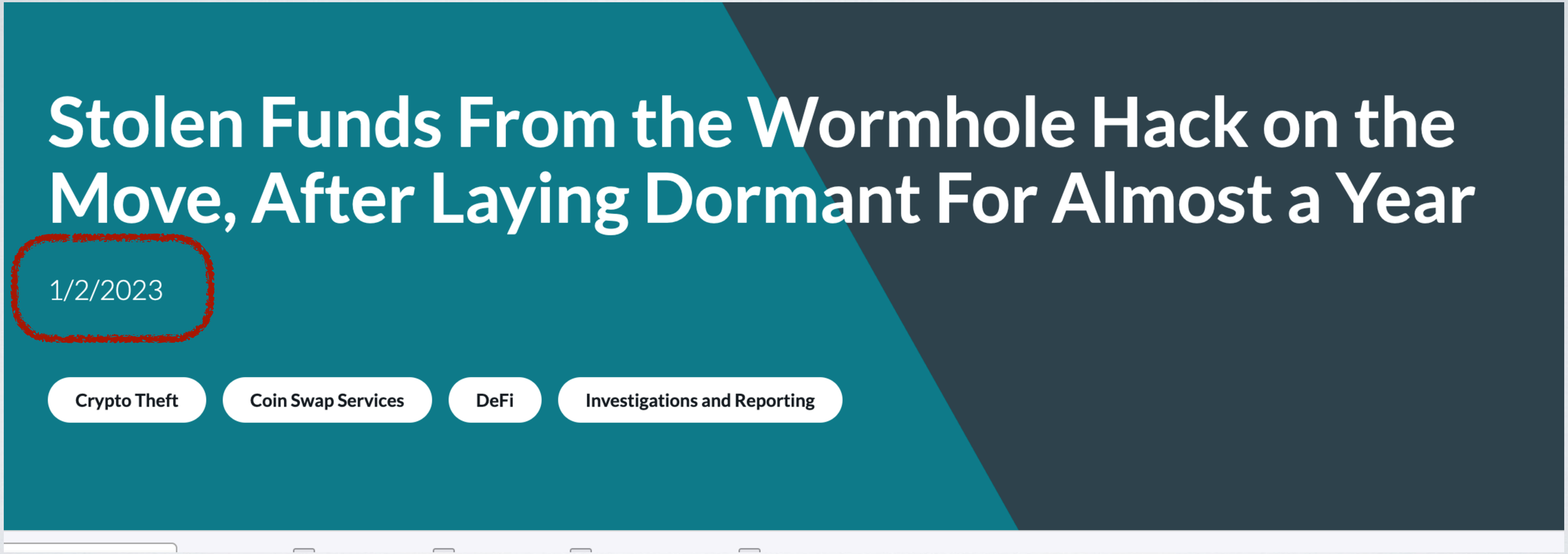
❤ 4.5K Reply Copy link

Read 579 replies

News?


Bridge
Token Bridge
Portal is a bridge that offers unlimited transfers across chains for tokens and NFTs wrapped by Wormhole.
Unlike many other bridges, you avoid double wrapping and never have to retrace your steps.
Tokens
NFTs
Redeem
1. SOURCE
Select tokens to send through the Portal.
TOKEN ORIGIN VERIFIER
Source
Solana
Target
Ethereum

News?


Stolen Funds From the Wormhole Hack on the Move, After Laying Dormant For Almost a Year
1/2/2023
Crypto Theft
Coin Swap Services
DeFi
Investigations and Reporting

News?

In February 2022, we reported on news that the Wormhole Portal – a decentralized finance (DeFi) bridge between Solana and other blockchains was hacked – with 120,000 Ether (ETH) then worth around $325 million stolen.

The exploit allowed the attacker to mint 120,000 wrapped ETH on the Solana blockchain, 93,750 ETH of which was then transferred to the Ethereum blockchain.

Subsequently, this ETH was bridged to Solana, where it remained until January 14th 2023, when the funds started to move again.

https://hub.elliptic.co/analysis/stolen-funds-from-the-wormhole-hack-on-the-move-after-laying-dormant-for-almost-a-year/

had been bridged across to 0x8184 on the 14th and swapped for Ether were transferred back to 0x629e in one transaction of 1,888.198678438568800875 ETH.

The exploiter then swapped 95,630 ETH for 95,677.79824465 stETH, which were then wrapped to make them usable across a wider number of DeFi applications as 86,473.48660411 wstETH.

stETH is a rebasable token, where daily token balance changes reflect accrued staking rewards. Some DeFi protocols require a constant balance mechanism so stETH can be wrapped into wstETH to keep the

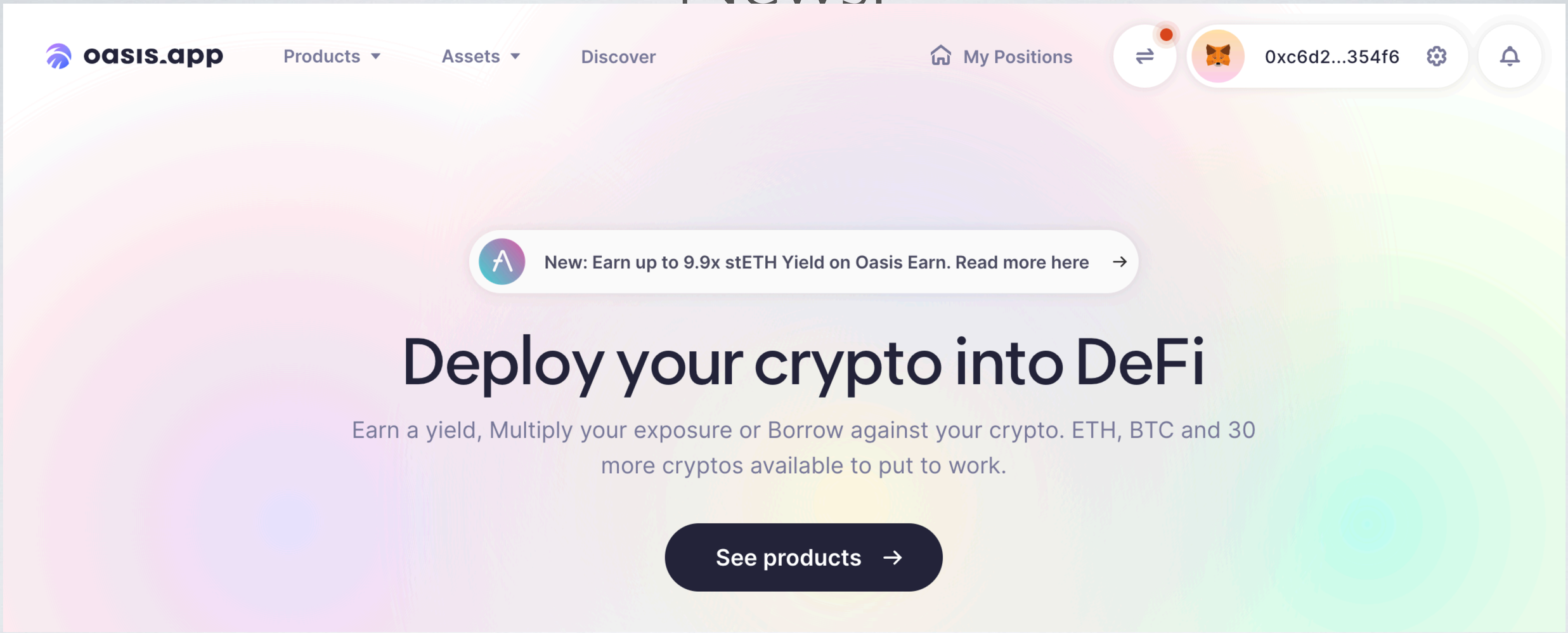
in one transaction of 1,888.198678438568800875 ETH.

The exploiter then swapped 95,630 ETH for 95,677.79824465 stETH, which were then wrapped to make them usable across a wider number of DeFi applications as 86,473.48660411 wstETH.

stETH is a rebasable token, where daily token balance changes reflect accrued staking rewards. Some DeFi protocols require a constant balance mechanism so stETH can be wrapped into wstETH to keep the balance fixed and allow use across more DeFi applications.

The exploiter then used 25,000 wstETH as collateral to borrow 14,500,000 DAI on Maker. The active vault can be checked on Oasis.

News?


oasis.app
Products
Assets
Discover
My Positions
0xc6d2...354f6
New: Earn up to 9.9x stETH Yield on Oasis Earn. Read more here
Deploy your crypto into DeFi
Earn a yield, Multiply your exposure or Borrow against your crypto. ETH, BTC and 30 more cryptos available to put to work.
See products

Vault History

Moved 120,695.43 WSTETH and 76.39M Dai debt from the vault	Feb 21, 2023, 7:19am
history.undefined executed	Feb 21, 2023, 7:19am
Auto-Buy trigger executed	Feb 16, 2023, 11:13am
Deposited 8,791.05 WSTETH into Vault	Feb 11, 2023, 5:32pm
Generated 14.99M Dai from Vault	Feb 11, 2023, 4:52pm
Deposited 24,357.00 WSTETH into Vault	Feb 11, 2023, 4:52pm
Auto-Buy trigger added	Feb 10, 2023, 12:14pm
Generated 14.99M Dai from Vault	Feb 10, 2023, 11:59am
Deposited 25,000.00 WSTETH into Vault	Feb 10, 2023, 11:59am

Vault History

Moved 120,695.43 WSTETH and 76.39M Dai debt from the vault	Feb 21, 2023, 7:19am
history.undefined executed	Feb 21, 2023, 7:19am
Auto-Buy trigger executed	Feb 16, 2023, 11:13am
Deposited 8,791.05 WSTETH into Vault	Feb 11, 2023, 5:32pm
Generated 14.99M Dai from Vault	Feb 11, 2023, 4:52pm
Deposited 24,357.00 WSTETH into Vault	Feb 11, 2023, 4:52pm
Auto-Buy trigger added	Feb 10, 2023, 12:14pm
Generated 14.99M Dai from Vault	Feb 10, 2023, 11:59am
Deposited 25,000.00 WSTETH into Vault	Feb 10, 2023, 11:59am

News?

HOME < NEWS < DEFI

Jump Crypto Just Counter-Exploited the Wormhole Hacker for $140 Million

The Chicago trading firm appears to have recovered the 120,000 ether stolen during the 2022 Wormhole exploit

BY JON RICE & DAN SMITH / FEBRUARY 24, 2023 03:58 PM

In a Feb. 24 blog post, the Oasis.app team confirmed that a counter exploit had taken place, outlining that it had “received an order from the High Court of England and Wales” to retrieve certain assets related to the “address associated with the Wormhole Exploit.”

The team stated that the retrieval was initiated via “the Oasis Multisig and a court-authorized third party,” which was identified as Jump Crypto in a preceding report from Blockworks Research.

Both vaults' transaction history indicates that Oasis moved 120,695 wsETH and 3,213 rETH on Feb. 21 and placed in wallets under Jump Crypto’s control. The hacker also had around $78 million debt in MakerDAO’s Dai DAI ▼ $1.00 stablecoin, which was retrieved.

“We can also confirm the assets were immediately passed onto a wallet controlled by the authorized third party, as required by the court order. We retain no control or access to these assets,” the blog post reads.

News?

Our team first became aware of the possibility to assist in the retrieval of the assets after a Whitehat group reached out to the team on the evening of Thursday 16th February 2023, that showed it would be possible to retrieve the assets and provided a Proof of Concept on how it could be achieved. What occurred on 21st February 2023 was only possible due to a previously unknown vulnerability in the design of the admin multisig access. We stress that this access was there with the sole intention to protect user assets in the event of any potential attack, and would have allowed us to move quickly to patch any vulnerability disclosed to us. It should be noted that at no point, in the past or present, have user assets been at risk of being accessed by any unauthorised party.

The Counter Exploit Mechanics

The $227M counter exploit took place on February 21. After a series of transactions across multiple wallets, the final executions delivered 120.7k wstETH and 3.2k rETH to the **Holder** where they currently reside. The **Holder** has a limited transaction history and made its first transaction on February 20, one day before the counter exploit. The recovered assets were sent to the **Holder** by the **Sender** who is responsible for executing a majority of the counter exploit. The **Sender** has no prior transaction history and was funded from KuCoin on February 20.

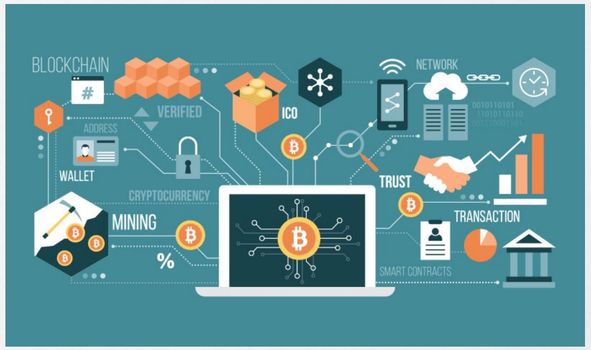
The process began on February 21 when the **Sender** was added as a signer to the **Oasis Multisig**. The **Sender** executed five transactions to facilitate the counter exploit and was subsequently removed as a signer from the **Oasis Multisig**. The **Sender** was an eligible signer for just 1 hour and 53 minutes.

Transaction Hash	Method	Block	Age	From		To
0x547acd21899672cb12...	Exec Transact...	16677095	2 days 15 hrs ago	Oasis Msig Owner	IN	Oasis Msig (0x85)
0x920e63573a3226402a...	Exec Transact...	16677047	2 days 15 hrs ago	Sender (0x04)	IN	Oasis Msig (0x85)
0x27d830955477016fa5...	Exec Transact...	16676978	2 days 15 hrs ago	Sender (0x04)	IN	Oasis Msig (0x85)
0x4f4117317a9f69915cb...	Exec Transact...	16676831	2 days 16 hrs ago	Sender (0x04)	IN	Oasis Msig (0x85)
0x47d1b2433fc2483f15d...	Exec Transact...	16676821	2 days 16 hrs ago	Sender (0x04)	IN	Oasis Msig (0x85)
0x08b0f18305dd7a4baa9...	Exec Transact...	16676590	2 days 16 hrs ago	Sender (0x04)	IN	Oasis Msig (0x85)
0x1c50b75828613d9408...	Exec Transact...	16676532	2 days 17 hrs ago	Oasis Msig Owner	IN	Oasis Msig (0x85)

https://www.blockworksresearch.com/research/we-do-a-little-counter-exploit

Today

- Finish Ethereum, finally
- Talk about how some actual smart contracts work
- DeFi!



Ethereum consensus (todo!)

- **We have not really talked much about Ethereum consensus**
  - The original version of Ethereum worked basically like Bitcoin (Nakamoto consensus, PoW mining)
  - The new version of Ethereum uses Proof-of-Stake
  - We will talk a lot more about this in a later lecture!

From concept to practice

- **How does a developer see Ethereum?**
  - So far we have talked about:
    - Init function (at deploy, creation)
      - (A note: contracts can ‘spawn’ new contracts!)
    - A single stateUpdate function (triggered by message Tx)
    - Databases and VMs

From concept to practice

- Ethereum programs are in "EVM byte code"
  - This is great for running things in a VM, works across platforms
  - Not made for human comprehension

```
00000d8b  PUSH1   #2 {var_e0_25}
00000d8d  EXP      {var_c0_53}
00000d8e  SUB      {var_c0_53} {var_a0_34}
00000d8f  DUP4     {var_40_4} {var_c0_54}
00000d90  AND      {var_a0_35} {var_a0_34} {var_c0_54}
00000d91  PUSH1   #0 {var_c0_55}
00000d93  SWAP1    {var_a0_35} {var_a0_36} {var_c0_56}
00000d94  DUP2     {var_e0_26}
```

Source: https://blog.ret2.io/2018/05/16/practical-eth-decompilation/

From concept to practice

- **How does a developer see Ethereum?**
  - So far we have talked about:
    - Init function (at deploy)
    - A single stateUpdate function (triggered by message Tx)
    - Databases and VMs

From concept to practice

- **How does a developer see Ethereum?**
  - So far we have talked about:
    - Init function (at deploy)
    - A single stateUpdate function (triggered by message Tx)
    - Databases and VMs
  - But this sucks for software developers
    - Let's instead think of contracts as object-oriented programs

From concept to practice

- Developers typically write programs in a high-level language
  - Technically any language can compile to EVM bytecode
  - And there are a few: Agoric (Javascript), Vyper
    - Some other chains (e.g., Solana) use rust
  - However, most Ethereum smart contracts are written in **Solidity**

Source: https://blog.ret2.io/2018/05/16/practical-eth-decompilation/

Solidity

- **How does a Solidity developer see Ethereum?**
  - Solidity is object-oriented
    - "contract" programs are like classes, with methods and variables
    - Each contract will have a constructor method that initializes any state variables
    - There are "view" (read-only) methods, and methods that (may) change state
    - Methods can have modifiers attached, that execute specific checks

```
contract TokenContractFragment {

    // Balances for each account
    mapping(address => uint256) balances;

    // Owner of account approves the transfer of an amount to another account
    mapping(address => mapping (address => uint256)) allowed;

    // Get the token balance for account `tokenOwner`
    function balanceOf(address tokenOwner) public constant returns (uint balance) {
        return balances[tokenOwner];
    }

    // Transfer the balance from owner's account to another account
    function transfer(address to, uint tokens) public returns (bool success) {
        balances[msg.sender] = balances[msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        Transfer(msg.sender, to, tokens);
        return true;
    }

    // Send `tokens` amount of tokens from address `from` to address `to`
    // The transferFrom method is used for a withdraw workflow, allowing contracts to send
    // tokens on your behalf, for example to "deposit" to a contract address and/or to charge
    // fees in sub-currencies; the command should fail unless the _from account has
    // deliberately authorized the sender of the message via some mechanism; we propose
    // these standardized APIs for approval:
    function transferFrom(address from, address to, uint tokens) public returns (bool success) {
        balances[from] = balances[from].sub(tokens);
        allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        Transfer(from, to, tokens);
        return true;
    }

    // Allow `spender` to withdraw from your account, multiple times, up to the `tokens` amount.
    // If this function is called again it overwrites the current allowance with _value.
    function approve(address spender, uint tokens) public returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        Approval(msg.sender, spender, tokens);
        return true;
    }
}
```

Concurrency and re-entrancy

- Ethereum transactions run sequentially and atomically
  - In principle this is good: there **appear** to be no concurrency issues (no threads) and your methods always run to completion or don't complete at all!

Example (ok)

```
function transfer(uint amount, address recipient) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }

    // Transfer the money
    balances[msg.sender] -= amount;
    balances[recipient] += amount;
}
```

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
    balances[notifyContract] += amount;
}
```

Re-entrancy

- There is still one scary "gotcha"!
- Ethereum is not re-entrancy "safe". *You can still have multiple calls to the same routine within any given call-stack.*

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

What if this contract call calls us?

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

What if this contract call calls us?

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

What if this contract call calls us?

Re-entrancy

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert('Something bad happened');
    }
    balances[notifyContract] += amount;

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, "You are getting a deposit!");

    // Transfer the money
    balances[msg.sender] -= amount;
}
```

What if this contract call calls us?

“The DAO”

```
function splitDAO(
  uint _proposalID,
  address _newCurator
) noEther onlyTokenholders returns (bool _success) {

  ...
  // XXXXX Move ether and assign new Tokens.  Notice how this is done first!
  uint fundsToBeMoved =
      (balances[msg.sender] * p.splitData[0].splitBalance) /
      p.splitData[0].totalSupply;
  if (p.splitData[0].newDAO.createTokenProxy.value(fundsToBeMoved)(msg.sender)
== false) // XXXXX This is the line the attacker wants to run more than once
      throw;

  ...
  // Burn DAO Tokens
  Transfer(msg.sender, 0, balances[msg.sender]);
  withdrawRewardFor(msg.sender); // be nice, and get his rewards
  // XXXXX Notice the preceding line is critically before the next few
  totalSupply -= balances[msg.sender]; // XXXXX AND THIS IS DONE LAST
  balances[msg.sender] = 0; // XXXXX AND THIS IS DONE LAST TOO
  paidOut[msg.sender] = 0;
  return true;
}
```

Solutions?

```
function transferAndNotify(uint amount, address notifyContract) … {
    if (globalLock == true) {
        revert(“Locked.”);
    }
    globalLock = true;

    if (balances[msg.sender] < amount) {
        // insufficient balance
        revert(‘Something bad happened’);
    }

    // Call the specified contract to notify it that a deposit is coming
    notifyContract.notify(amount, “You are getting a deposit!”);

    // Transfer the money
    balances[msg.sender] -= amount;
    balances[notifyContract] += amount;

    globalLock=false;
}
```

```
function transferAndNotify(uint amount, address notifyContract) … {
   if (globalLock == true) {
      revert(“Locked.”);
   }
   globalLock =

   if (balances[msg.s
      // insufficient b
      revert(‘Someth
   }

   // Call the specifi
   notifyContract.no

   // Transfer the m
   balances[msg.sen
   balances[notifyContract] += amount;

   globalLock=false;
}
```

Drawbacks of (global) locks:

1. Extra gas (due to stores/loads)

2. Can get “stuck” if you’re careless

3. Sometimes re-entrant calls are useful!

Check-Effects-Interaction pattern

- Most common solution is to follow a code pattern:
  - First perform all contract checks (CHECKS)
  - Second, update contract state (EFFECTS)
  - Finally, make any contract calls (INTERACTION)

Check-Effects-Interaction pattern

```
function transferAndNotify(uint amount, address notifyContract) … {
   // CHECK
   require (amount < balances[msg.sender]);

   // EFFECTS
   balances[msg.sender] -= amount;
   balances[notifyContract] += amount;

   // INTERACTION
   notifyContract.notify(amount, "You are getting a deposit!");
}
```

Contract upgrades

- Ethereum contracts are not (natively) upgradeable
  - Once a contract is deployed, it can self-destruct
  - But its code cannot be changed
  - But some contracts need to be upgraded (bug fixes, etc.)
    - How are we going to handle this?