

# Lesson 10 - Security

## Deus DAO

See [Rekt write up](#)

\$DEI was exploited on Arbitrum and other chains,

See [transaction](#)

Overview Logs (53) Advanced TxInfo Comments

② Transaction Hash: 0xb1141785b7b94eb37c39c37f0272744c6e79ca1517529fec3f4af59d4c3c37ef ⓘ

② Status: Success

② Block: 87626026 25486 L1 Block Confirmations

② Timestamp: 3 days 14 hrs ago (May-05-2023 05:52:45 PM +UTC)

② From: 0x189cf534de3097c08b6beaf6eb2b9179dab122d1 (Deus DEI Exploiter) ⓘ

② Interacted With (To): Contract 0xe2ee6252509382a2b6504d5a5f7a1c5018a38168 ⓘ

② ERC-20 Tokens Transferred: 6

- From 0x7dc406b9b904a... To Null: 0x000...000 For 0 (\$0.00) ⓧ DEI (DEI)
- From 0x7dc406b9b904a... To 0xe2ee6252509382a2b6504d5a5f7a1c5018a38168 For 4,602,837.090538811392635119 (\$1,554,272.22) ⓧ DEI (DEI)
- From 0xe2ee6252509382a2b6504d5a5f7a1c5018a38168 To 0x7dc406b9b904a... For 4,602,837.090538811392635119 (\$1,554,272.22) ⓧ DEI (DEI)
- From 0x7dc406b9b904a... To Deus DEI Exploiter For 5,047,470.472572 (\$5,052,517.94) ⓘ USD Coin (Ar... (USDC)
- From 0x7dc406b9b904a... To 0x6cb66a0762e7c... For 220.936180345862946846 (\$74.61) ⓧ DEI (DEI)
- From 0x7dc406b9b904a... To 0xd2193bffe8f9c5... For 1,620.198655869661610208 (\$547.10) ⓧ DEI (DEI)

② Value: 0 ETH (\$0.00)

② Transaction Fee: 0.0004075314 ETH (\$0.76)

### Call Trace

```
✓ [call][397482] [Exploiter Contract].swap(arg0=[TransparentUpgradeableProxy], arg1=[Victim]) → ()  
  > [call][34040][?] [TransparentUpgradeableProxy].approve(arg0=[Victim], arg1=115792089237316195423570985008687907853269984665640564039457584007913129639935) → ()  
  > [call][32254][?] [TransparentUpgradeableProxy].burnFrom(arg0=[Victim], arg1=0) → ()  
  > [staticcall][1412][?] [TransparentUpgradeableProxy].balanceOf(arg0=[Victim]) → ()  
  > [call][31455][?] [TransparentUpgradeableProxy].transferFrom(arg0=[Victim], arg1=[Exploiter Contract], arg2=4602837090538811392635119) → ()  
  > [call][123662][?] [Victim].sync() → ()  
  > [staticcall][1412][?] [TransparentUpgradeableProxy].balanceOf(arg0=[Exploiter Contract]) → ()  
  > [staticcall][33983][?] [Victim].getAmountOut(arg0=4602837090538811392635119, arg1=[TransparentUpgradeableProxy]) → ()  
  > [call][4152][?] [TransparentUpgradeableProxy].transfer(arg0=[Victim], arg1=4602837090538811392635119) → ()  
  [call][328] [Victim].token0() → ()  
  [call][1054] [Victim].token1() → ()  
  > [call][117704][?] [Victim].swap(arg0=0, arg1=5047470472572, arg2=0x189cf534de3097c08b6beaf6eb2b9179dab122d1, arg3=0x) → ()  
  > [staticcall][1296][?] [TransparentUpgradeableProxy].balanceOf(arg0=0x189cf534de3097c08b6beaf6eb2b9179dab122d1) → ()
```

The problem function is the `burnFrom` function

```
function burnFrom(address account, uint256 amount) public virtual {  
    uint256 currentAllowance = _allowances[_msgSender()][account];  
    _approve(account, _msgSender(), currentAllowance - amount);  
    _burn(account, amount);  
}
```

and the approve mapping

```
function _approve(
    address owner,
    address spender,
    uint256 amount
) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

The attacker can set up a large allowance to the victim

The attacker can use their own approval to burn the victim's tokens.

Then the allowance is written back so that the victim now has 'approved' a large allowance for the attacker.

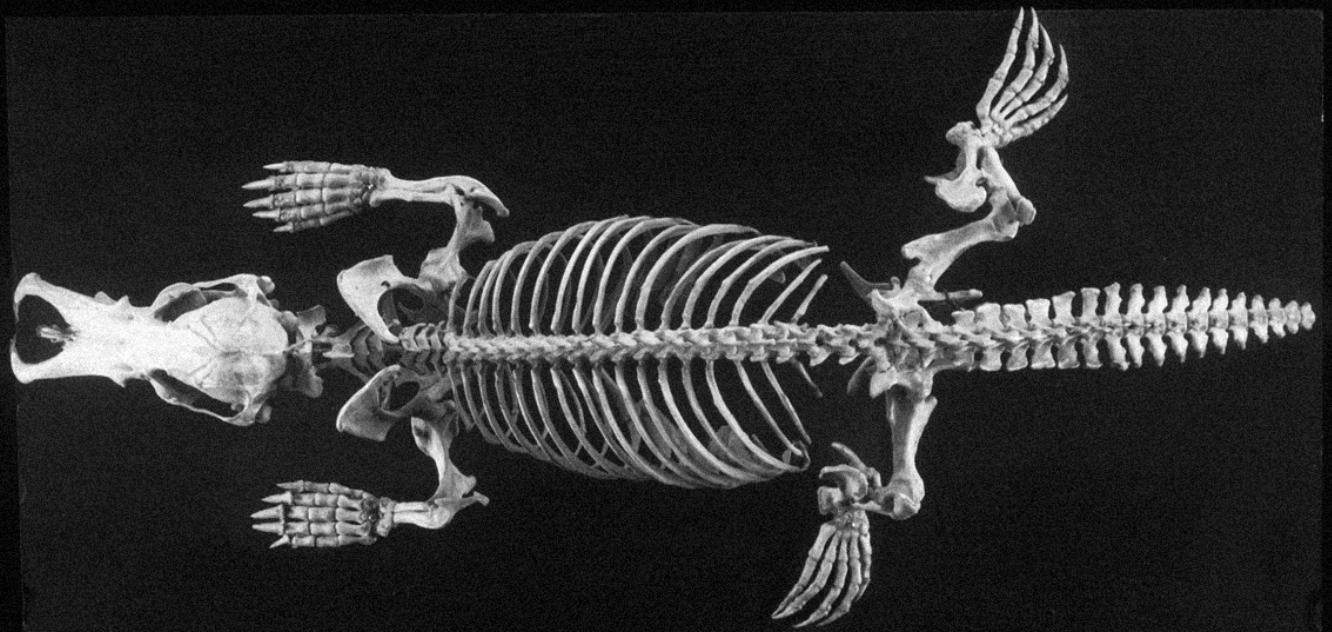
The attacker can then call `TransferFrom` to take the victim's tokens.

## From rekt.news

"Luckily, the exploit on BSC was frontrun, and an on-chain [message](#) to the Deus Deployer shows the intent to return the funds. Other [whitehats](#) also sprang into action, and over \$600k in USDC has so far been returned to a [recovery multisig](#).

*However, there were also [doubts](#) about the usefulness of giving funds back to a team that produced such a trivial bug"*

## Platypus Finance



 Rekt

References :

[Rekt.news](#)



Platypus ▲ (kusama + binance + usdt)  
@Platypusdefi

...

Dear Community,

We regret to inform you that our protocol was hacked recently, and the attacker took advantage of a flaw in our USP solvency check mechanism. They used a flashloan to exploit a logic error in the USP solvency check mechanism in the contract holding the collateral.

1:42 AM · Feb 17, 2023 · 172.5K Views

This was quite a simple exploit

It started with the taking a flash loan of 44M USDC which was deposited into Platypus. The resulting LP tokens were then used as collateral to borrow 41.7M USP.

The `emergencyWithdraw()` function only checks whether the user's position is *currently* solvent, but neglects to first check against any the effect of any borrowed funds. This allows the attacker to withdraw the supplied collateral while keeping the borrowed USP.

```

function emergencyWithdraw(uint256 _pid) public nonReentrant {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    if (address(platypusTreasure) != address(0x00)) {
        (bool isSolvent, ) = platypusTreasure.isSolvent(msg.sender, address(poolInfo[_pid].lpToken), true);
        require(isSolvent, 'remaining amount exceeds collateral factor');
    }

    // reset rewarder before we update lpSupply and sumOfFactors
    IBoostedMultiRewarder rewarder = pool.rewarder;
    if (address(rewarder) != address(0)) {
        rewarder.onPtpReward(msg.sender, user.amount, 0, user.factor, 0);
    }

    // SafeERC20 is not needed as Asset will revert if transfer fails
    pool.lpToken.transfer(address(msg.sender), user.amount);

    // update non-dialuting factor
    pool.sumOfFactors -= user.factor;

    user.amount = 0;
    user.factor = 0;
    user.rewardDebt = 0;
}

```

The collateral withdrawn was then used to replay the flash loan, and the USP was swapped for other stablecoins.

The attacker was however tracked down

**ZachXBT** ✅ @zachxbt · Feb 17

Replying to @zachxbt

I've reviewed your transaction history across multiple chains which lead me to your ENS address `retlqw.eth`

Your OpenSea account links directly to your Twitter and you liked a Tweet about the Platypus exploit.

13 yo kid using his mom's

Collected 9 Created

Search by name

LINKS

Filters

Twitter

MORE

retlqw.eth Joined November 2017

1,963 Following 96 Followers

Tweets Tweets & replies Media Likes

DEFIFYIELD Web 3 Security ✅ @... · 4h ...

Platypus Finance's \$USP stablecoin was just flash-loan-attacked !!

The \$USP stablecoin has depegged and is currently barely collateralized 💀

20 37 611 80.1K Tip

In a further twist funds were then taken from the attacker

details from Daniel Von Fange

"The attacker forgot to code any way collect the funds after stealing them, so the funds were locked in the attack contract. They also neglected Flash Loan 101 and allowed anyone to call the flash loan callback code. No check that they had started the flash loan. "

"This allowed @BlockSecTeam and the project to retrigger the hack, but with one major twist - the project contracts had been upgraded to steal back from the attacker during the hack"

"The attack sequence involved taking flash loaned USDC, approving it, and depositing into the project. But during the retrigger, the attack code used its own stolen USDC to approve and deposit instead. The new project code simply took the attacker's USDC and ran with it."

## Invocation Flow



# Wintermute Hack

Rekt [article](#)

Mudit Gupta [blog](#)

Evgeny Gaevoy CEO of Wintermute [tweets](#)

## Attack vector

An admin address was compromised using the profanity vulnerability, Wintermute had removed funds from this address, but hadn't removed its admin role, which allowed the attacker to remove tokens from the vault.

The address was compromised because it had been created using the profanity tool, which was recently found to have a vulnerability.

Admin Address : 0x0000000fe6a514a32abdcdfcc076c85243de899b

It was created in this way to have large number of leading zeros which can save gas.

Evgeny Gaevoy explained :

"Last time we generated addresses this way was in June. We have since moved to a more secure key generation script. As we learned about the Profanity exploit last week, we accelerated the "old key" retirement

And then, due to an internal (human) error, a wrong function has been called and we blacklisted the router instead of the operator (contract that signs)

And such is a challenge of running a (truly) automated Market Maker in the Dark Forest. You must automate processes as much as possible as multisig solutions are not applicable to our way of high speed trading "

The attack started with :

<https://etherscan.io/tx/0xeeecba26d5eb7939257e5b3e646e4bc597b73e256a89cb84a6dfc58de250d8a38> where Wintermute's hot wallet

(<https://etherscan.io/address/0x0000000fe6a514a32abdcdfcc076c85243de899b>) started calling their vault contract

(<https://etherscan.io/address/0x0000000ae347930bd1e7b0f35588b92280f9e75>) to transfer tokens out to the hacker's contract

(<https://etherscan.io/address/0x0248f752802b2cfb4373cc0c3bc3964429385c26>)

## Profanity vulnerability

see 1inch [write up](#)

The author of the tool says :

"I've decided to also archive this repository to further reduce risk that someone uses this tool. The code will not receive any updates and I've left it in an uncomplilable state. Use something else! "

Evgeny Gaevoy finishes his tweets with

"And this is what we are planning to do. No lay-offs. No strategy changes. No emergency

fundraise. Not giving up on defi. Keep moving forward through this bear market with the rest of you"

---

# Nomad bridge Hack

Thanks to [Rekt](#) and [@samczsun](#) for their insights.

On 1st August 2022 the Nomad bridge was hacked and within 3 hours about \$190 M of funds were taken. The attack affected related projects such as EVMOS, Milkomena and Moonbeam.

## August 1st 2022

The exploits unfolds...

The exploit was spotted as unusual activity by @spreekaway, see this [thread](#)

### Nomad bridge getting rugged??? Looks very very sus

📄 Nomad: ERC20 Bridge	OUT	0xe4a4df7e1689589efb0...	100
📄 Nomad: ERC20 Bridge	OUT	📄 0xa8ecaf8745c56d5935c...	100
📄 Nomad: ERC20 Bridge	OUT	📄 0xd1a7feb0317bbe40ac...	100
📄 Nomad: ERC20 Bridge	OUT	⌚ bitliq.eth	100
📄 Nomad: ERC20 Bridge	OUT	0xb5c55f76f90cc528b26...	100
📄 Nomad: ERC20 Bridge	OUT	📄 0x0000000000000660def...	100
📄 Nomad: ERC20 Bridge	OUT	📄 0xf57113d8f6ff35747737...	100

10:37 PM · Aug 1, 2022 · Twitter Web App

It seemed that transactions were draining funds from the bridge, by sending 0.01 WBTC you could get 100 WBTC back

⌚ bitliq.eth	IN	📄 Nomad: ERC20 Bridge	0.01
📄 Nomad: ERC20 Bridge	OUT	0xe4a4df7e1689589efb0...	100
📄 Nomad: ERC20 Bridge	OUT	📄 0xa8ecaf8745c56d5935c...	100

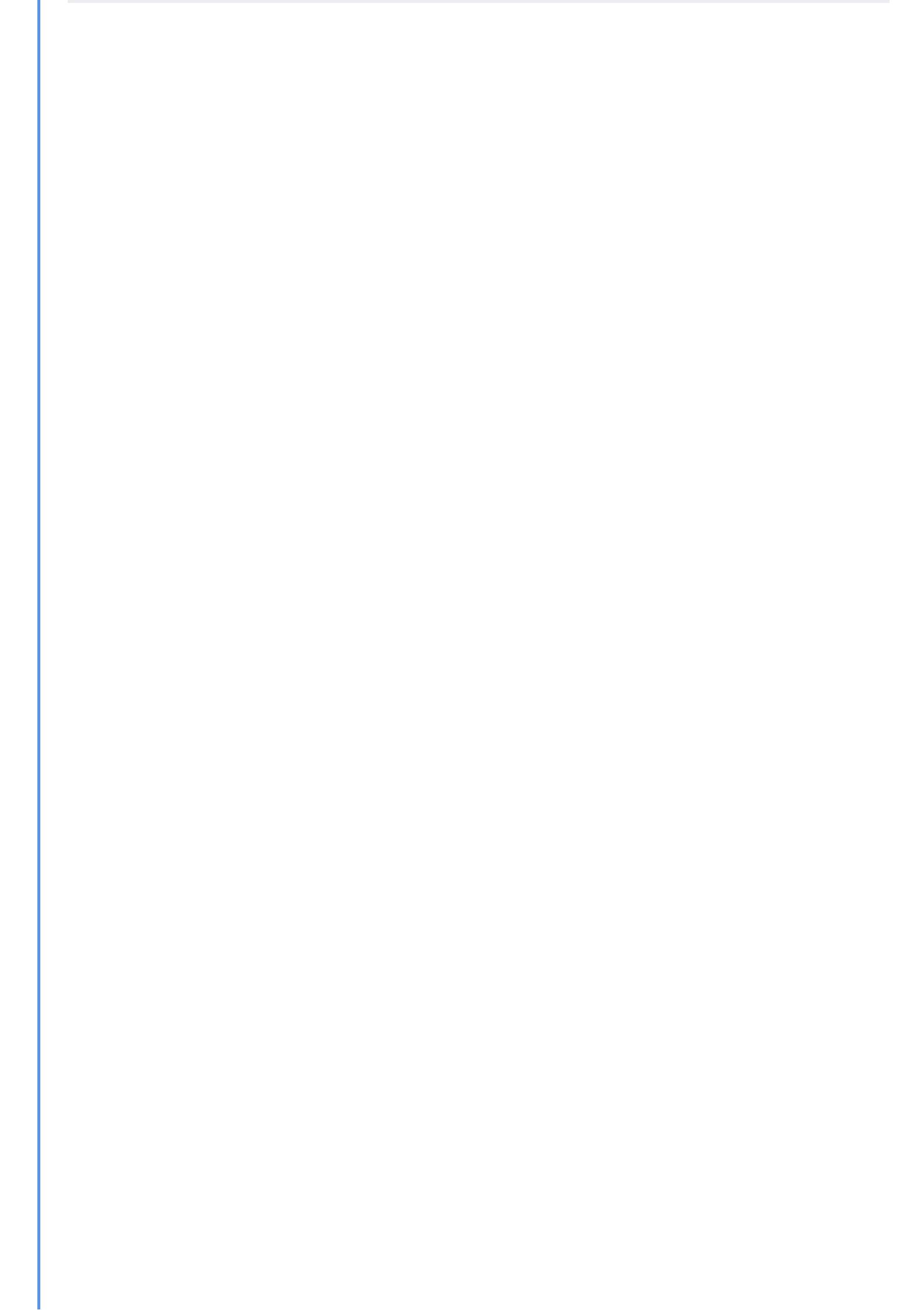
In theory the bridge was controlled by a transfer first being. proved and then a transaction could go through to process the movement of the funds.

So what *should* have been happening is that the messages submitted should have been proven, then included in a merkle tree whose root is stored and flagged as confirmed at a certain time.

What seemed to be happening is that messages were being processed that hadn't previously been confirmed. Once the initial exploiting transaction went through, anyone could simply submit a similar transaction to drain funds for themselves.

Subsequently there were many such transactions from exploiters, white hats , along with the use of MEV techniques to take advantage of the situation.

	0x13ea2c98982d1fde14...	Process	15259332	4 days 11 hrs ago	0xe4a4df7e1689589efb0...
	0x8183fe98f81deb8aa37...	Process	15259332	4 days 11 hrs ago	Nomad Bridge Exploiter 3
	0xa304a513cc1b62da3d...	Process	15259312	4 days 11 hrs ago	Resident Arbitrageur
	0xe7fd692fcfae783149b...	Process	15259312	4 days 11 hrs ago	Nomad Bridge Exploiter 1
	0xa9ddf5f8c22c33f814a...	Process	15259312	4 days 11 hrs ago	0xa5eff6157b44d7eba6b...
	0xe0112511747dd3b0c3...	Process	15259303	4 days 11 hrs ago	⌚ bitliq.eth
	0x39b107d43d88f1120e...	Process	15259303	4 days 11 hrs ago	Nomad Bridge Exploiter 3
	0xcd54193008330871d8...	Process	15259282	4 days 11 hrs ago	Resident Arbitrageur
	0x6ae090b41da74ed958...	Process	15259249	4 days 11 hrs ago	Nomad Bridge Exploiter 2
	0x370e7192707fa7d4da...	Process	15259240	4 days 11 hrs ago	Nomad Bridge Exploiter 2
	0xef73e48c1be5f025412...	Process	15259240	4 days 11 hrs ago	Nomad Bridge Exploiter 2
	0x8732887090364bd5f8...	Process	15259217	4 days 11 hrs ago	Nomad Bridge Exploiter 2
	0x282525e58e17d5f442...	Process	15259217	4 days 11 hrs ago	Nomad Bridge Exploiter 2
	0xa9108b394beeb02efc...	Process	15259217	4 days 11 hrs ago	Nomad Bridge Exploiter 2
	0x51f0ed5db858a85218...	Process	15259202	4 days 11 hrs ago	0xe4a4df7e1689589efb0...
	0xbbb009a4b5cb19d8ae...	Process	15259201	4 days 11 hrs ago	⌚ bitliq.eth
	0x11f468aa89bc97d748...	Process	15259200	4 days 11 hrs ago	Nomad Bridge Exploiter 1
	0x56b4ade16ad35792f0...	Process	15259159	4 days 11 hrs ago	0xe4a4df7e1689589efb0...
	0x73ae1f3a7f81d140e21...	Process	15259156	4 days 11 hrs ago	0xa5eff6157b44d7eba6b...
	0xb0447743d0d29c5756...	Process	15259155	4 days 11 hrs ago	⌚ bitliq.eth
	0x5c5f3325d212b8e086...	Process	15259155	4 days 11 hrs ago	Nomad Bridge Exploiter 3
	0x498a0778cc8448a100...	Process	15259104	4 days 12 hrs ago	0xe4a4df7e1689589efb0...
	0xa5fe9d044e4f3e5aa5b...	Process	15259101	4 days 12 hrs ago	⌚ bitliq.eth
	0xb1fe26cc8892f58eb46...	Process	15259101	4 days 12 hrs ago	Nomad Bridge Exploiter 3



The top 3 exploiters were

[0x56D8B635A7C88Fd1104D23d632AF40c1C3Aac4e3](#) (\$47M)

[0xBF293D5138a2a1BA407B43672643434C43827179](#) (\$40M)

[0xB5C55f76f90Cc528B2609109Ca14d8d84593590E](#) (\$8M)

Some exploiters sent their funds through Tornado Cash to obfuscate the trail of funds.

As usual the exploited project appealed for people to return their funds.

Nomad have offered a deal to the exploiters asking for return of 90% of the funds, allowing the other 10% to be regarded as a bug bounty

### Update: Nomad Bridge Hack Bounty

Nomad is announcing an **up to 10%** bounty to Nomad Bridge hackers where Nomad will consider any party who returns **at least 90%** of the total funds they hacked to be a white hat. Nomad will not pursue legal action against white hats. Funds must be returned to the official Nomad recovery wallet address:

**0x94a84433101a10aeda762968f6995c574d1bf154.**

Please be wary of impersonators and other scams.

Nomad is continuing to work with its community, law enforcement and blockchain analysis firms to ensure all funds are returned.

N O M A D

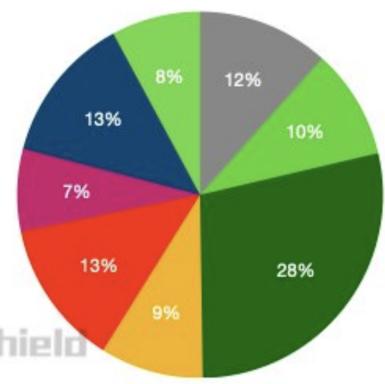
### Return of funds

From @PeckShieldAlert as of August 5th, about 11% of funds have been recovered / returned

### Nomad Funds Recovery Address

Token	Amount	Value (\$)
ETH	241	\$399,392
WETH	1,308.86	\$2,163,813.5
WBTC	92.7	\$2,145,448.8
USDC	6,261,842	\$6,268,104
USDT	2,013,335.7	\$2,015,349
CQT	34,666,035	\$2,836,778.8
FRAX	1,614,400	\$1,612,913
DAI	2,879,523.8	\$2,882,403
CARDS	158,082.5	\$41,911
IAG	247,189,221	\$1,308,560
SUSHI	4,557	\$6,835
C3	1,649,120	\$237,192
GERO	23,101,165.5	\$43,500
HBOT	1,200,320	\$4,279
FXS	12,503	\$93,647

● ETH & WETH ● WBTC ● USDC ● USDT  
 ● CQT ● FRAX ● DAI ● Other



Some white hats have managed to recover some of the funds

Thank you to

-  .eth (\$4m)
  - 0xE3F40743cc18fd45D475fAe149ce3ECC40aF68c3 (\$3.4m)
  - darkfi.eth (\$1.9m)
  - returner-of-beans.eth (\$1m)
  - anime.eth (\$900k)
- for returning a total of \$11.2m to our recovery address!

We've recovered a total of \$16.6m so far.

5:19 AM · Aug 4, 2022 · Twitter Web App

# Exploit Details

The vulnerable contract is the Replica [contract](#)

This contract was upgraded and had the following initializer

```
function initialize(
    uint32 _remoteDomain,
    address _updater,
    bytes32 _committedRoot,
    uint256 _optimisticSeconds
) public initializer {

    __NomadBase_initialize(_updater);
    // set storage variables
    entered = 1;
    remoteDomain = _remoteDomain;
    committedRoot = _committedRoot;
    // pre-approve the committed root.
    confirmAt[_committedRoot] = 1;
    _setOptimisticTimeout(_optimisticSeconds);
}
```

It was initialized with the zero address

## Transaction

```
Function: initialize(uint32 _remoteDomain, address _updater, bytes32 _committedRoot, uint256 _optimisticSeconds)
```

```
function initialize(
    uint32 _remoteDomain,
    address _updater,
    bytes32 _committedRoot,
    uint256 _optimisticSeconds
) public initializer {
    __NomadBase_initialize(_updater);
    // set storage variables
    entered = 1;
    remoteDomain = _remoteDomain;
    committedRoot = _committedRoot;
    // pre-approve the committed root.
    confirmAt[ committedRoot ] = 1;
```

```
    _setOptimisticTimeout(_optimisticSeconds);  
}
```

In the initialize function, committedRoot is set to 0x00

```
    committedRoot = _committedRoot;  
// pre-approve the committed root.  
confirmAt[ committedRoot ] = 1;
```

and so

```
confirmAt[ committedRoot] = 1;
```

The first attempt to [exploit](#) the contract failed, it was an expensive failure costing 215 ETH.

② Block:	15259103	28418 Block Confirmations
② Timestamp:	⌚ 4 days 10 hrs ago (Aug-01-2022 09:32:59 PM +UTC)	⚖ Confirmed within 30 secs
② From:	0xb5c55f76f90cc528b2609109ca14d8d84593590e	(Nomad Bridge Exploiter 3) <a href="#">🔗</a>
② To:	Contract 0xdb09d04d33539e3366de2591e920eba71edc879	<span style="color: red;">⚠</span> <a href="#">🔗</a> ↳ Warning! Error encountered during contract execution [execution reverted] <a href="#"> ⓘ</a>
② Value:	0.00000000000000466 Ether	(< \$0.000001) - [CANCELLED] <a href="#"> ⓘ</a>
② Transaction Fee:	215.881231529184168417 Ether	(\$371,313.56)
② Gas Price:	0.000358358465198001 Ether	(358,358.465198001 Gwei)
② Ether Price:	\$1,630.62 / ETH	
② Gas Limit & Usage by Txn:	1,232,758	602,417 (48.87%)
② Gas Fees:	Base: 18.74448131 Gwei	

This was followed by the exploiting transaction calling the process function with the following arguments

Txn Type: 0 (Legacy)   Nonce: 25   Position: 3

View Input As 

## Decode Input Data

The process function can be called by anyone, and is intended to process messages that have already been proved.

## The process function

```
function process(bytes memory _message) public returns (bool _success) {
    // ensure message was meant for this domain
    bytes29 _m = _message.ref(0);
    require(_m.destination() == localDomain, "!destination");
    // ensure message has been proven
    bytes32 _messageHash = _m.keccak();
    require(acceptableRoot(messages[_messageHash]), "!proven");
    // check re-entrancy guard
    require(entered == 1, "!reentrant");
    entered = 0;
    // update message status as processed
    messages[_messageHash] = LEGACY_STATUS_PROCESSED;
    // call handle function
    IMessageRecipient(_m.recipientAddress()).handle(
        _m.origin(),
        _m.nonce(),
        _m.sender(),
        _m.body().clone()
    );
    // emit process results
    emit Process(_messageHash, true, "");
    // reset re-entrancy guard
    entered = 1;
    // return true
    return true;
}
```

This line

```
require(acceptableRoot(messages[_messageHash]), "!proven");
```

checks the validity of the merkle root of the messages

using this function

```
function acceptableRoot(bytes32 _root) public view returns (bool) {
// this is backwards-compatibility for messages proven/processed
// under previous versions
if (_root == LEGACY_STATUS_PROVEN) return true;
if (_root == LEGACY_STATUS_PROCESSED) return false;
uint256 _time = confirmAt[_root];
if (_time == 0) {
return false;
}
return block.timestamp >= _time;
}
```

using the following constants declared earlier.

```
bytes32 public constant LEGACY_STATUS_NONE = bytes32(0);
bytes32 public constant LEGACY_STATUS_PROVEN = bytes32(uint256(1));
bytes32 public constant LEGACY_STATUS_PROCESSED = bytes32(uint256(2));
```

The line

```
uint256 _time = confirmAt[_root];
```

now returns 1 (remember in initialize, `confirmAt[0x00]` is set to 1)

You can check the return value of this function yourself on etherscan as I did :

"LatestCourse/img/Screenshot 2022-08-06 at 09.51.41.png" is not created yet. Click to create.

Thus any message sent to process would succeed !

## Note

The audit report from Quanstamp is [available](#), their finding 'QSP-19 Proving With An Empty Leaf' is particularly relevant to this exploit, but perhaps not understood by the Nomad team.

---

# Poly Network hack

## Recap of function selectors

## Encoding the function signatures and parameters

## Example

```
pragma solidity ^0.8.0;                                     language=none

contract MyContract {

    Foo otherContract;

    function callOtherContract() public view returns (bool){
        bool answer = otherContract.baz(69,true);
        return answer;
    }
}

contract Foo {
    function bar(bytes3[2] memory) public pure {}
    function baz(uint32 x, bool y) public pure returns (bool r) {
        r = x > 32 || y;
    }
    function sam(bytes memory, bool, uint[] memory) public pure {}
}
```

The way the call is actually made involves encoding the function selector and parameters

If we wanted to call `baz` with the parameters `69` and `true`, we would pass 68 bytes total, which can be broken down into:

1. the Method ID. This is derived as the first 4 bytes of the Keccak hash of the ASCII form of the signature `baz(uint32,bool)`.

\*\*\*0xcdcd77c0:\*\*\*

2. the first parameter, a uint32 value 69 padded to 32 bytes

3. the second parameter - boolean true, padded to 32 bytes

In total



# The Polynetwork Hack - \$600M stolen

## Poly Network – Stolen Funds Breakdown



### Ethereum Blockchain

	Quantity stolen
USDC	96,389,444
WBTC (wrapped Bitcoin)	1,032
DAI	673,227
UNI (Uniswap)	43,023
SHIBA	259,737,345,149
renBTC	14.47
USDT	33,431,197
wETH (wrapped Ether)	26,109
FEI USD	616,082



### Binance Smart Chain

	Quantity stolen
BNB	6,613.44
USDC	87,603,373
ETH	299
BTCB	26,629
BUSD	1,023



### Polygon Blockchain

	Quantity stolen
USDC	85,089,610

## Polynetwork react to attack



**Poly Network** @PolyNetwork2 · Aug 10, 2021

...

Replies to @PolyNetwork2

We call on miners of affected blockchain and crypto exchanges to blacklist tokens coming from the above addresses.

@Tether\_to  
@circlepay

19

195

420



Tip



**Poly Network** @PolyNetwork2 · Aug 10, 2021

...

We will take legal actions and we urge the hackers to return the assets.

188

313

527



Tip



**Poly Network** @PolyNetwork2 · Aug 10, 2021

...

Assets involved include \$WBTC \$WETH \$RenBTC.

ETH:0xC8a65Fadf0e0dDAf421F28FEAb69Bf6E2E589963

We call on miners of affected blockchain and crypto exchanges to blacklist tokens coming from the above addresses.

@BitGo @renBTCFinance

14

51

262



Tip



**Poly Network** @PolyNetwork2 · Aug 10, 2021

...

Assets involved include \$DAI \$UNI \$SHIB \$FEI.

ETH:0xC8a65Fadf0e0dDAf421F28FEAb69Bf6E2E589963

We call on miners of affected blockchain and crypto exchanges to blacklist tokens coming from the above addresses.

@MakerDAO @Uniswap @Shibtoken @feiprotocol

25

76

274



Tip

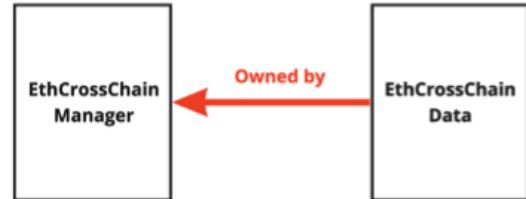
## Example messages to the attacker

- Can I have some ETH? i been losing a lot of money Thank you
- Consider donating to public goods after all the pleasure you've got from using public infra 🌱
- ngmi
- Welcome to the crypto world. Fee Tips: Use tornado.cash to laundry your money ASAP.
- Forget your USDT. Forget your USDC.
- Swap tokens to ETH then deposit to tornado.cash . Good luck.
- Hi. Boos. Can. You. Give me eth thanks
- You can use Tornado for currency mixing
- DONT USE YOUR USDT TOKEN YOU VE GOT BLACKLISTED
- You can buy every pudgy penguin on opensea :)
- Dad, this is my only asset. Please accept it
- JOINING FOR EPIC SCREENCAP

- Please gice me some eth
  - DONT USE YOUR USDT TOKEN  
YOU VE GOT BLACKLISTED,god bless you
  - Remove liquidity from curve pool in form of DAI,  
and exchange it for Eth and launder with Tornado
-

### Mismanagement of access rights between two contracts

EthCrossChainManager is an owner of EthCrossChainData,  
= **EthCrossChainManager can execute privileged functions!**



*\_method* is user defined  
= **can be set at will.**

```
bytes4(keccak256(abi.encodePacked(_method, "(bytes,bytes,uint64)"))),
```

miro

5

#### Vulnerable Contract 2: EthCrossChainData

**Very High Privileged contract !** ➔ Can only be called by its owners.

Set + manage list of “**Keepers**”

= **list of public keys that manage the wallets in the underlying liquidity chain**

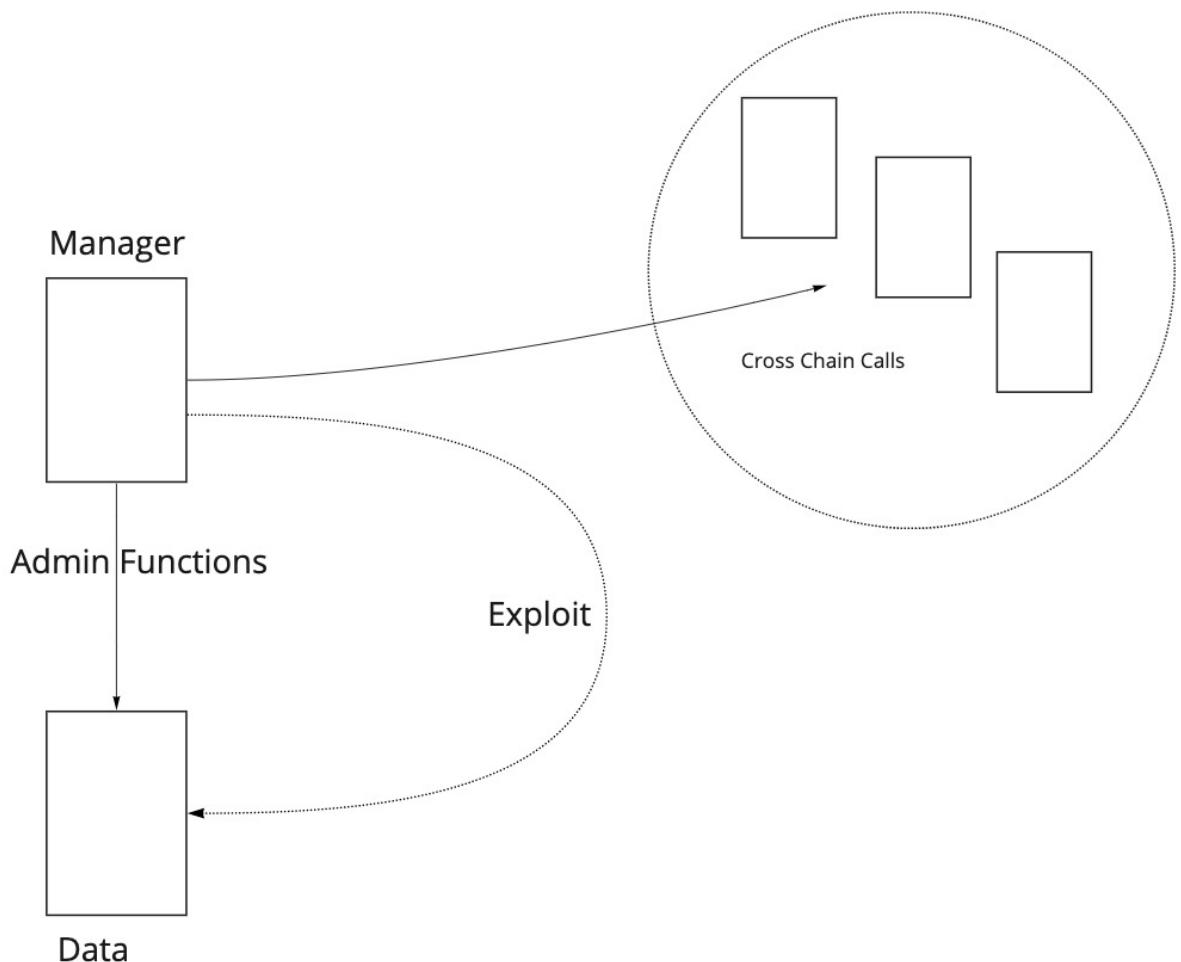
➔ **Keepers have the right to execute large transactions**, transfer large amounts to other wallets.

Vulnerable function: *putCurEpochConPubKeyBytes*

= become a “**Keeper**”

Set the public key (passed as parameter) as a Keeper

10



Aiming to find X such that

$\text{Hash}(X, \text{bytes}, \text{bytes}, \text{uint64})$

=

$\text{Hash}(\text{putCurEpochConPubKeyBytes}, \text{bytes}) = 0x41973cd9$

The Brute Force solution was

$X = f1121318093$

## The Attacker reconsiders

The attacker received a rather cryptic message

"Dont instant tornado funds, dont instant move blacklistable tokens to DAI/ETH? Insider confirmed!"

The attacker was looking at Tornado Cash and sent themselves a message

"Wonder why Tornado? Will miners stop me? Teach me please"

Then someone found a link between an address used by the attacker and some exchanges and tweeted

"Did the PolyNetwork Exploiter accidentally use the wrong sender address for this tx 0xb12681d9e? The sender address is tied to FTX, Binance, Okex accounts."

The hacker's attitude started to change, he suggested he could return "some tokens" or even abandon them, saying that they were "not so interested in the money".

Followed by a suggestion : "What if I make a new token and let the DAO decide where the tokens go"

---

Finally the attacker messaged "Ready to return the fund !"

See this [spreadsheet](#) for all communications

The attacker starts to return the funds

PolyNetwork starts to refer to the attacker as 'Mr White Hat' and offer him a job and bounty



'We appreciate you sharing your experience and believe your action constitutes white hat behavior. But we can't touch user assets and Poly Network doesn't have its own token. Since , we believe your action is white hat behavior, we plan to offer you a \$500,000 bug bounty after you complete the refund fully. Also we assure you that you will not be accountable for this incident. We hope that you can return all tokens as soon as possible. You can reserve the equivalent value of 500,000 USD in any assets to the current owner address. We will make up this part of the assets to Poly Network users.

Your contribution is very helpful to us. Again, we think this behavior is white hat behavior, therefor this 500,000 USD will be seen as completely legal bounty reward. We will also ensure that you will not be held accountable for this incident, and we will publicly express our gratitude to you.'

Lasttime Refund	Chain	Hacker Address	Hack	Refund
12/8/2021 15:40:04	ETH Chain	0xC8a65Fadf0e0dDAf421F28FEAb69Bf6E2E589	272 mil	Almost
	BSC Chain	0x0D6e286A7cfD25E0c01fEe9756765D8033B32	253mil	ALL
	Polygon Chain	0x5dc3603c9d42ff184153a8a9094a73d46166321	85 mil	ALL
Still waiting more Fund	Chain	Receive Address (Polynetwork Multisig)	Balance	Received
8394,74 Hours	ETH Chain	0x71Fb9dB587F6d47Ac8192Cd76110E05B8fd21	Almost	Almost
	New Multisig W	0x34d6b21d7b773225a102b382815e00ad876e2:	ALL	ALL
	BSC Chain	0xEEBb0c4a5017bEd8079B88F35528eF2c722b:	ALL	ALL
	Polygon Chain	0xA4b291Ed1220310d3120f515B5B7AccaecD66	ALL	ALL

A number of rather lengthy messages follow the return of the funds

Q & A, PART TWO:

Q: WHAT REALLY HAPPENED 30 HOURS AGO?

A: LONG STORY.

BELIEVE IT OR NOT, I WAS FORCED TO PLAY THE GAME.

THE POLY NETWORK IS A SOPHISTICATED SYSTEM, I DIDN'T MANAGE TO BUILD A LOCAL TESTING ENVIRONMENT. I FAILED TO PRODUCE A POC AT THE BEGINNING. HOWEVER, THE AHA MOMEMNT CAME JUST BEFORE I WAS TO GIVE UP. AFTER DEBUGGING ALL NIGHT, I CRAFTED A SINGLE MESSAGE TO THE ONTOLOGY NETWORK.

I WAS PLANNING TO LAUNCH A COOL BLITZKRIEG TO TAKE OVER THE FOUR NETWORK: ETH, BSC, POLYGON & HECO. HOWEVER THE HECO NETWORK GOES WRONG! THE RELAYER DOES NOT BEHAVE LIKE THE OTHERS, A KEEPER JUST RELAYED MY EXPLOIT DIRECTLY, AND THE KEY WAS UPDATED TO SOME WRONG PARAMETERS. IT RUINED MY PLAN.

I SHOULD HAVE STOPPED AT THAT MOMENT, BUT I DECIDED TO LET THE SHOW GO ON! WHAT IF THEY PATCH THE BUG SECRETLY WITHOUT ANY NOTIFICATION?

HOWEVER, I DIDN'T WANT TO CAUSE REAL PANIC OF THE CRYPTO WORLD. SO I CHOSE TO IGNORE SHIT COINS, SO PEOPLE DIDN'T HAVE TO WORRY ABOUT THEM GOING TO ZERO. I TOOK IMPORTANT TOKENS (EXCEPT FOR SHIB) AND DIDN'T SELL ANY OF THEM.

Q: THEN WHY SELLING/SWAPPING THE STABLES?

A: I WAS PISSED BY THE POLY TEAM FOR THEIR INITIAL REPONSE.

THEY URGED OTHERS TO BLAME & HATE ME BEFORE I HAD ANY CHANCE TO REPLY! OF COURSE I KNEW THERE ARE FAKE DEFI COINS, BUT I DIDN'T TAKE IT SERIOUSLY SINCE I HAD NO PLAN LAUNDERING THEM.

IN THE MEANWHILE, DEPOSITING THE STABLES COULD EARN SOME INTEREST TO COVER POTENTIAL COST SO THAT I HAVE MORE TIME TO NEGOTIATE WITH THE POLY TEAM.

# Arbitrum critical vulnerability

A critical vulnerability that was spotted in time, caused by too much optimisation.

See [writeup](#) from white hat.

A bridge address is used to receive funds deposited in Arbitrum, it is vital that this address is set up correctly and cannot be switched to a malicious contract.

Upgradeable contracts can go very wrong if the initialize function isn't adequately secured.

In this contract there were 2 flags in storage to check if initialize had been run or was running

```
modifier initializer() {
    // If the contract is initializing we ignore whether _initialized is set in order to support multiple
    // inheritance patterns, but we only do this in the context of a constructor, because in other contexts the
    // contract may have been reentered.
    require(_initializing ? _isConstructor() : !_initialized, "Initializable: contract is already initialized");

    bool isTopLevelCall = !_initializing;
    if (isTopLevelCall) {
        _initializing = true;
        _initialized = true;
    }

    _;

    if (isTopLevelCall) {
        _initializing = false;
    }
}
```

The initialize function also had check to see if the bridge address was already set

```
105     function initialize(IBridge _bridge, ISequencerInbox _sequencerInbox)
106         external
107         initializer
108         onlyDelegated
109     {
110         -         if (address(bridge) != address(0)) revert AlreadyInit();
111         bridge = _bridge;
112         sequencerInbox = _sequencerInbox;
113         allowListEnabled = false;
114         __Pausable_init();
    }
```

Unfortunately that checked had been taken out to reduce gas, but things would still be ok, as long as the flags were still set...

A further optimisation cleared out the first 3 storage slots after upgrade

```
function postUpgradeInit(IBridge _bridge) external onlyDelegated onlyProxyOwner {
    uint8 slotsToWipe = 3;
    for (uint8 i = 0; i < slotsToWipe; i++) {
        assembly {
            sstore(i, 0)
        }
    }
    allowListEnabled = false;
    bridge = _bridge;
}
```

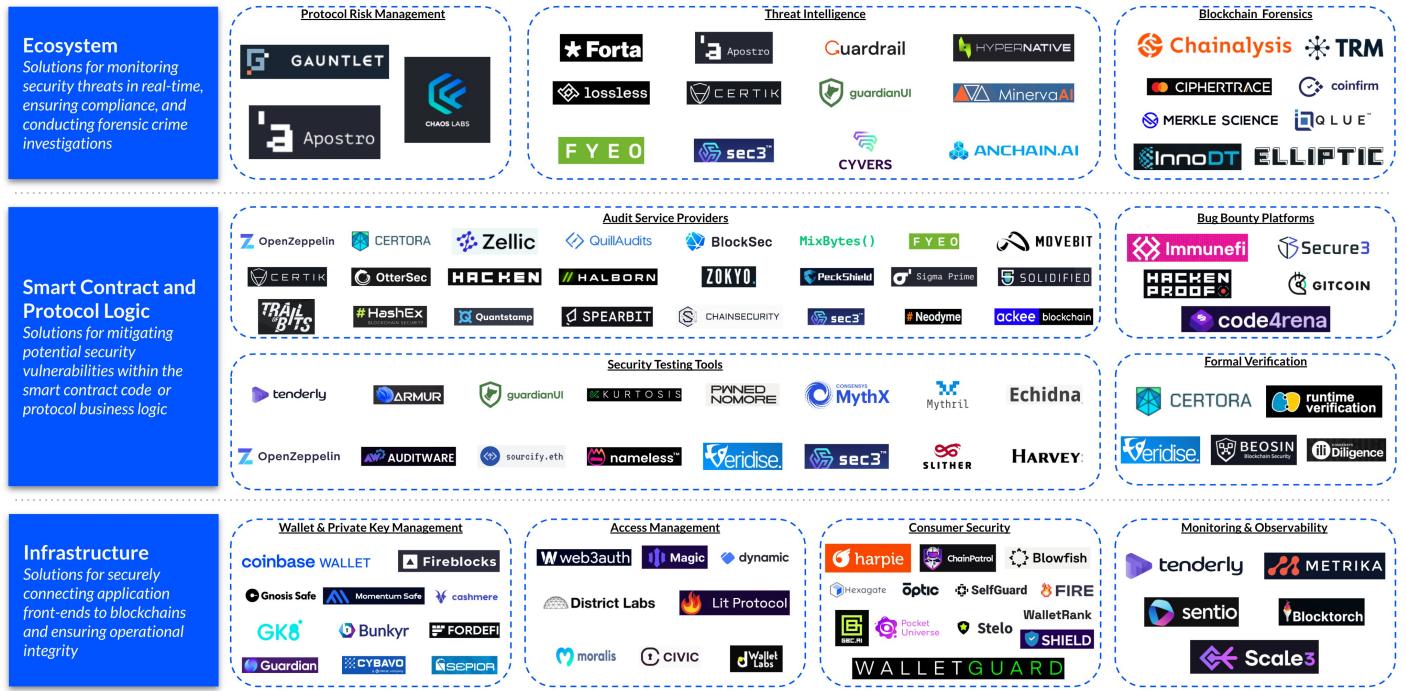
So, an attacker could now call the initialize function, setting the address of the bridge contract to a contract that they control, and thus receive any ETH that users deposit.

---

# Basic Security

## Ecosystem

### The Web3 Security Stack



\*This landscape is not exhaustive and contains Coinbase Ventures portfolio companies. See disclosures at the end of the article\*

## Consensys Best Practices

### General

- [Prepare for Failure](#)
- [Stay up to Date](#)
- [Keep it Simple](#)
- [Rolling out](#)
- [Blockchain Properties](#)
- [Simplicity vs. Complexity](#)

### Precautions

- [General](#)
- [Upgradeability](#)
- [Circuit Breakers](#)
- [Speed Bumps](#)
- [Rate Limiting](#)
- [Deployment](#)
- [Safe Haven](#)

### Solidity Specific

- [Assert, Require, Revert](#)

- [Modifiers as Guards](#)
- [Integer Division](#)
- [Abstract vs Interfaces](#)
- [Fallback Functions](#)
- [Payability](#)
- [Visibility](#)
- [Locking Pragmas](#)
- [Event Monitoring](#)
- [Shadowing](#)
- [tx.origin](#)
- [Timestamp Dependence](#)
- [Complex Inheritance](#)
- [Interface Types](#)
- [EXTCODESIZE Checks](#)

## Token Specific

- [Standardization](#)
- [Frontrunning](#)
- [Zero Address](#)
- [Contract Address](#)

## Documentation

- [General](#)
- [Specification](#)
- [Status](#)
- [Procedures](#)
- [Known Issues](#)
- [History](#)
- [Contact](#)

## Attacks

- [Reentrancy](#)
- [Oracle Manipulation](#)
- [Frontrunning](#)
- [Timestamp Dependence](#)
- [Insecure Arithmetic](#)
- [Denial of Service](#)
- [Griefing](#)
- [Force Feeding](#)

# Advanced Security

## Oracle Manipulation

A smart contract must determine the value of an asset, such as when a user deposits ETH into the system. To accomplish this, the protocol refers to its corresponding Uniswap pool as a source of price information.

This behaviour can be exploited by an attacker, who might use a flash loan to empty one side of the Uniswap pool.

Due to the reliance on a single data source, the protocol's internal price can be directly tampered with, for instance, increasing it to 100 times its original value. The attacker can then capitalise on this inflated value by executing an arbitrage trade based on the new price discrepancy or securing a favourable position within the system.

The issues can be summarised as:

1. The smart contract's reliance on a single price feed source makes it susceptible to on-chain manipulation through flash loans.
2. Even when a significant anomaly is present, smart contracts that use the price data continue to function based on the distorted information.

### Example

The [following code](#) shows that on deposit, the price feed is fetched directly from Uniswap:

```
uint160 sqrtPrice = TickMath.get.SqrtRatioAtTick(currentTick()); uint256 price =  
FullMath.mulDiv(uint256(sqrtPrice).mul(uint256(sqrtPrice)), PRECISION, 2**(96 *  
2));
```

Here, `currentTick()` directly fetches the current price tick from a Uniswap pool:

```
// @return tick Uniswap pool's current price tick function currentTick() public  
view returns (int24 tick) { (, tick, , , , ) = pool.slot0(); }
```

As this price data is fetched from an on-chain dependency, and the price data is determined in the current transaction context, this spot price can be manipulated in the same transaction.

1. An attacker can take out a flash loan on the incoming asset A and on the relevant Uniswap pool, swap asset A for asset B with a large volume.
2. This trade will increase the price of asset B (increased demand) and reduce the cost of asset A (increased supply).
3. When asset B is deposited into the above function, its price is still pumped up by the flash loan.
4. Consequentially, asset B gives the attacker an over-proportional amount of shares.
5. These shares can be withdrawn, giving the attacker equal parts of asset A and asset B from the pool.
6. Repeating this process will drain the vulnerable pool of all funds.

7. With the money gained from the withdrawal of their shares, the attacker can repay the flash loan.

## Read only reentrancy

Also see this : <https://youtu.be/8D5ZJyU-dX0>

Classical reentrancy exploits occur in state-modifying functions, leading to inconsistent state usage for malicious storage writes. To mitigate this, contracts often implement reentrancy locks to protect the state.

In contrast, the read-only reentrancy is a reentrancy scenario where a view function is reentered, which in most cases is unguarded as it does not modify the contract's state. However, if the state is inconsistent, wrong values could be reported.

As a result, other protocols blindly relying on a return value (e.g. from Oracles or Price Feeds) can be tricked into reading the wrong state to perform unwanted actions.

There have been two exploits using this vulnerability, both in LPs interacting with certain Curve Pools.

An attacker manipulates the collateral price, borrows all the funds from the pool, and walks away with bad debts.

To simplify, that's what happened in four steps in both cases.

1. Deposit large amounts of liquidity.
2. Remove liquidity. (that triggers a callback)
3. During the callback, the attacker performs malicious actions.
4. Profit.

The reentrancy occurs during step 3, when balances still need to be fully updated while total LP token supply decreases.

This is achieved by compromising Curve's Oracle's price feed and borrowing funds at an inflated price.

see [Oracle manipulation post mortem report](#)

---

# Security Tools

## Overview

TABLE I  
OVERVIEW OF ALL TOOLS INDICATING PURPOSE, CODE LEVEL, TYPE, PRE-PROCESSING, AND METHODS OF ANALYSIS

Tool	Purpose			Level	Type	Code transformation					Analysis method					
	Security issues	Exploits	Formal guarantees			Bytecode	Solidity code	Static analysis	Dynamic analysis	Contextualization	Disassembly	Control flow graph	Call graph	AST analysis	Decompilation	Code instrumentation
contractLarva	x	x	x	x	x	✓	x	✓	x	x	x	x	x	x	✓	x
E-EVM	x	x	x	x	✓	x	✓	x	x	x	✓	x	x	x	x	x
Erays	x	x	x	x	✓	x	✓	x	x	✓	✓	x	x	✓	x	x
EthIR	x	x	x	x	✓+	x	✓	x	x	✓	✓	x	x	✓	x	✓
EtherTrust	x	x	✓	✓	✓	x	✓	x	x	x	x	x	x	x	✓	✓
FSolidM	x	x	x	x	form.spec	✓	x	x	x	x	x	x	x	x	x	✓
KEVM	x	x	✓	x	✓	x	✓	x	x	x	x	x	x	x	x	x
MAILAN	✓	✓+	x	✓	✓+	x	✓	✓	x	✓	✓	x	x	✓	✓	x
Manticore	✓	✓	x	x	✓+	x	✓	x	✓	✓	x	x	x	✓	✓	x
Mythril	✓	✓	x	x	✓+	x	✓	x	✓	✓	✓	x	x	x	✓	x
Osiris	✓	x	x	✓	✓+	x	✓	x	✓	✓	✓	x	x	✓	✓	x
Oyente	✓	x	x	✓	✓+	x	✓	x	✓	✓	✓	x	x	✓	✓	x
Porosity	✓	x	x	x	✓+	x	✓	x	x	✓	✓	x	x	✓	x	x
Rattle	x	x	x	x	✓	x	✓	x	x	✓	✓	x	x	✓	x	x
Remix-IDE	✓	x	x	x	x	✓	✓	x	✓	x	x	x	x	x	x	x
Securify*	✓	x	✓	✓	✓+	x	✓	x	✓	✓	x	x	x	✓	✓	x
SmartCheck*	✓	x	x	x	x	✓	✓	x	✓	x	x	x	✓	x	x	x
Solgraph	✓	x	x	x	x	✓	✓	x	✓	x	x	✓	✓	x	x	x
SolMet	x	x	x	x	x	✓	✓	x	x	x	x	x	✓	x	x	x
Vandal	✓	x	x	✓	✓	x	✓	x	x	✓	✓	x	✓	x	✓	✓
Ether*	✓	x	x	✓	✓+	x	✓	x	x	x	x	✓	x	x	x	x
Gasper	✓	x	x	✓	✓	x	✓	x	x	✓	✓	x	x	✓	✓	x
ReGuard	✓	x	x	✓	✓	x	✓	x	x	✓	✓	x	✓	x	x	x
SASC	✓	x	x	✓	x	✓	✓	x	✓	✓	✓	✓	✓	x	✓	x
sCompile	✓	x	x	✓	✓	x	✓	✓	✓	✓	✓	x	x	✓	✓	x
teEther	✓	✓+	x	✓	✓	x	✓	✓	x	✓	✓	x	x	✓	x	x
Zeus	✓	x	✓	✓	✓	x	✓	x	x	x	x	✓	x	✓	x	x

LEGEND. Tool: \* indicates that we consider the academic version instead of the enhanced company version of the tool, due to availability. *Security issues*: detection of vulnerabilities and potential security problems. *Exploit*: generates exploits, with ✓+ indicating multi-transactional exploits. *Formal guarantees*: proves that a contract has a certain property. *Bulk analysis*: suitable for the analysis of large sets of contracts. *Level* of code at which analysis is performed; ✓+ indicates analysis of bytecode including information from the corresponding Solidity code such as the ABI; ‘form.spec’ means formal specification, from which Solidity code is generated.

From

[A Survey of Tools for Analyzing Ethereum Smart Contracts](#)

## Introduction to Slither

See [repo](#)

Slither performs static analysis on source code, it has 'detectors' for known vulnerabilities / patterns.

It also includes some tools to help visualise / understand contracts and program flow.

## Features

- Detects vulnerable Solidity code with low false positives (see the list of [trophies](#))
- Identifies where the error condition occurs in the source code
- Easily integrates into continuous integration and Truffle builds

- Built-in 'printers' quickly report crucial contract information
- Detector API to write custom analyses in Python
- Ability to analyze contracts written with Solidity  $\geq 0.4$
- Intermediate representation ([SlithIR](#)) enables simple, high-precision analyses
- Correctly parses 99.9% of all public Solidity code
- Average execution time of less than 1 second per contract

Slither can

1. produce markdown reports:
2. Display inheritance
3. produce call graphs
4. Show updatable variables and function authorisation
5. Check updatability
6. Flatten code
7. Read storage in a contract.

## Installation

```
pip3 install slither-analyzer
```

---

# Ethereum Security Toolbox

Trail of Bits have a security [toolbox](#) :

This includes

- [Echidna](#) property-based fuzz tester
- [Etheno](#) integration tool and differential tester
- [Manticore](#) symbolic analyzer and formal contract verifier
- [Slither](#) static analysis tool
- [Rattle](#) EVM lifter
- [Not So Smart Contracts](#) repository

Install in docker with

```
docker pull trailofbits/eth-security-toolbox
docker run -it trailofbits/eth-security-toolbox
```

# ZIION from Halborn

See [Docs](#)

[Download](#)

## ZIION BENEFITS



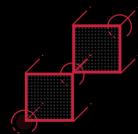
### TURN-KEY VM

Get engineers ready faster by having a turnkey VM with all the best blockchain security tools needed, already compiled



### OPENSOURCE

Community driven which means it's always free to use and open-source



### COMPREHENSIVE

Reduce friction points by having a working and tested VM for each architecture and protocol family

## ZIION FEATURES

ZIION contains over 100 tools, utilities, dependencies, and packages to immediately begin working with Solidity/EVM and Rust-based blockchains like:

### Blockchains targeted

- Ethereum (EVM)
- Bitcoin
- Solana
- Polygon
- BSC
- Substrate
- Near

### Tools include

Manticore, Slither, Mythril, Solgraph, Echidna, Certora-cli, Foundry

# Tenderly

## Product

### Monitoring

Stay up to date with your smart contracts and ensure that everything is going as planned by relying on accurate, real-time blockchain data.

### Alerting

Get real-time alert notifications directly to your inbox to keep track of important or unexpected events related to your smart contracts or wallets.

### Web3 Gateway

Eliminate node management overhead, connect to the blockchain with one URL, and get 8x faster read-heavy workloads backed by our all-in-one development tooling.

### Web3 Actions

Use an automated serverless backend to react quickly to on-chain and off-chain events by writing custom code that executes in less than a second.

### Simulator

Test custom solutions and play out transaction outcomes before sending them on-chain for confident execution both in development and production.

### War Rooms

Follow a structured war room framework to guide your team, secure funds, minimize potential damage, and act quickly in case of a security breach.

### Analytics

Rely on real-time blockchain data and track metrics essential to your project to respond to crucial events and stay one step ahead of the competition.

## Supported Networks



Ethereum Mainnet



Polygon



BSC



Sepolia



Görli



Gnosis



POA



RSK



Avalanche



Optimism



Arbitrum



Fantom



Moonbeam



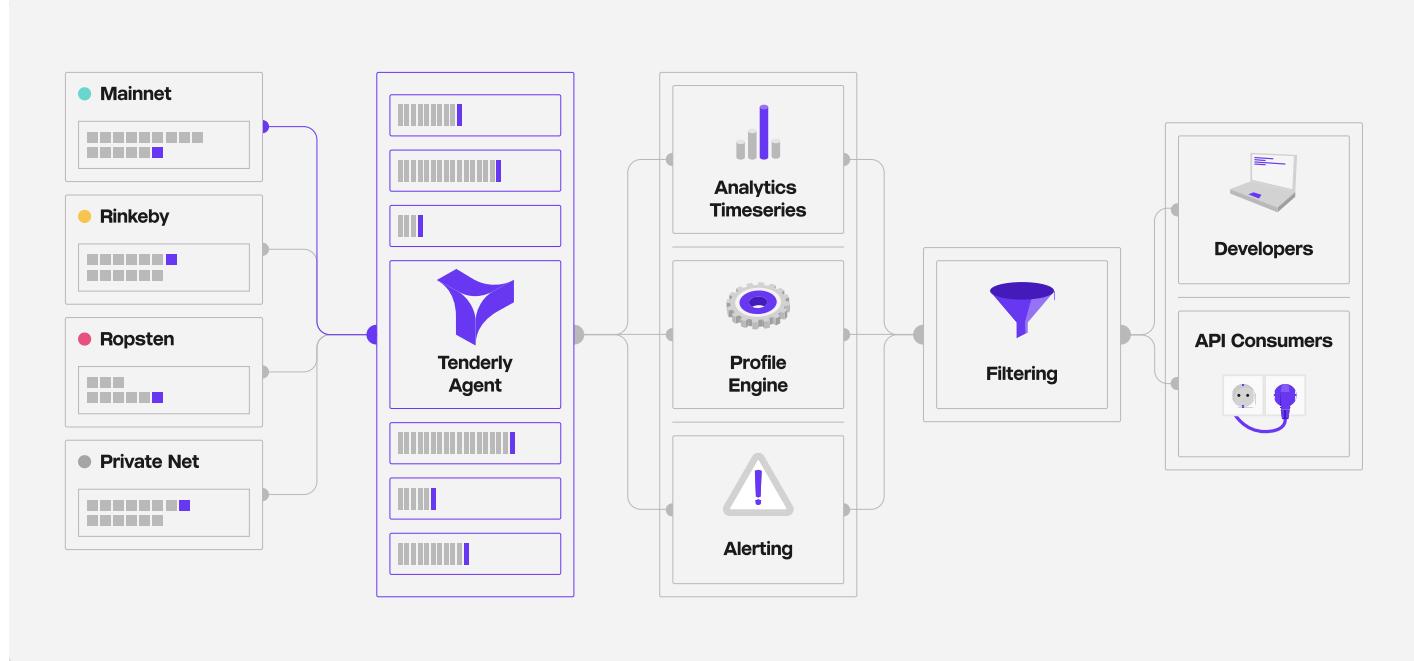
Moonriver



Cronos

See [Documentation] (<https://docs.tenderly.co/>)

# Monitoring



# Alerts

The screenshot shows a user interface for selecting an alert trigger type. At the top left is a circular icon with the number '1' and the word 'Type'. Below it is the text 'Select an alert trigger type.' The main area contains ten cards arranged in three rows:

- Successful Transaction**: Triggers whenever a successful transaction happens.
- Failed Transaction**: Triggers whenever a failed transaction happens.
- Function Call**: Triggers whenever a specific function is called in one of your contracts.
- Event Emitted**: Triggers whenever a specific event is emitted in one of your contracts.
- Event Parameter**: Triggers whenever a specific argument in an event matches the set conditions.

- ERC20 Token Transfer**: Triggers whenever an ERC20 transfer event is emitted in one of your contracts.
- Allowlisted Callers**: Triggers whenever an address that is not allowlisted calls one of your contracts.
- Blocklisted Callers**: Triggers whenever an address from this list calls one of your contracts.
- ETH Balance**: Triggers when the ETH balance of an address falls below a certain threshold.
- Transaction Value**: Triggers whenever a transaction value matches set conditions.

- State Change**: Triggers whenever a state variable in one of your contracts changes.
- View Function**: Triggers whenever a view function return value changes.
- More Coming Soon**: Have an idea? Click here and send us what you think can be the next alert type.

## Web3 Gateway

See [Docs](#)

Tenderly Web3 Gateway is a production node that offers reliable, fast, and consistent access to the blockchain.

Use the Tenderly node to:

- Read, stream, and analyze blockchain data with 100% consistency.
- [Run tx simulations before sending them on-chain](#) using a single RPC URL.

## Simulators

Transaction Simulator allows you to see a transaction's execution without sending it to the blockchain. It gives you the results of running a transaction against any point in blockchain history, including the latest block.

With Transaction Simulator, your transactions are run in a lightweight simulation environment, delivering detailed information about state changes, emitted events (logs), gas usage, and all calls that a simulated transaction made.

For example you can simulate a DAI approval

The screenshot shows the Tenderly Transaction Simulator interface. On the left, there's a sidebar with various icons. The main area is titled "New Simulation".  
**Contract:** Set to "Dai".  
**Function:** Set to "approve".  
**Input Parameters:** Two fields: "address" with value "0xf1" and "uint256" with value "29900000000000000000000000000000".  
**Transaction Parameters:** "From" field has a purple arrow pointing to it, containing the value "0xdc6bdc37b2714ee601...".  
**Block Header Overrides:** "Override Block Number" is checked, and the "Block Number" field is empty. "Override Timestamp" is also checked.  
A note at the bottom left says: "You can now execute simulations in the Web3 Gateway. Click the 'Open in RPC Builder' button to see how to simulate via JSON-RPC." and "\*Applicable for Web3 Gateway supported networks: Mainnet, Ropsten, Rinkeby, Görli, Sepolia."

### Editing contract Source in a simulation

You can edit the contract source on the fly while setting up your simulation or re-simulating, by showing the entire code and giving access to any line or parameter you would want to change.

ON-CHAIN DATA

Transactions

Contracts

Wallets

Simulator

Forks

Local Transactions

MONITORING

Networks

Analytics

Alerting

Actions

Beta

PROJECT

Collaborators

Settings

MANAGED BY

Tenderly

## New Simulation

### Contract

Use Custom Contract

UniswapV2Router02

Edit Contract Source

### Function

addLiquidity

### Input Parameters

address	tokenA
address	tokenB
uint256	amountADesired
uint256	amountBDesired
uint256	amountAMin
uint256	amountBMin
address	to
uint256	deadline

### Transaction Parameters

Use Pending Block

Block number

/

Current block: 14274432

Tx index

/

Maximum Block Index: 43

From

0x000

Use default from address

Gas

8000000

Use default gas value

Gas Price

0

Value

0

### State Override

### Optional Access Lists

Simulate Transaction

ON-CHAIN DATA

Transactions

Contracts

Simulator

Forks

Local Transactions

MONITORING

Networks

Analytics

Alerting

PROJECT

Collaborators

Settings

## New Simulation

### Contract

Use Custom Contract

KyberNetworkProxy

Compiler Info

KyberNetworkPr...

### Function

trade

### Input Parameters

address	0x00
uint256	0
address	0x00
address	0x00
uint256	0
uint256	0
address	0x00

### Transaction Parameters

Use Pending Block

### Editing Source

#### Address

KyberNetworkProxy

0x818e6fec516ecc3849da6845e3ec868087b755

#### File

KyberNetworkProxy.sol

```

1 pragma solidity 0.4.18;
2
3 // File: contracts/ERC20Interface.sol
4
5 // https://github.com/ethereum/EIPs/issues/20
6 interface ERC20 {
7     function totalSupply() public view returns (uint supply);
8     function balanceOf(address _owner) public view returns (uint balance);
9     function transfer(address _to, uint _value) public returns (bool success);
10    function transferFrom(address _from, address _to, uint _value) public returns (bool success);
11    function approve(address _spender, uint _value) public returns (bool success);
12    function allowance(address _owner, address _spender) public view returns (uint digits);
13    function decimals() public view returns(uint digits);
14    event Approval(address indexed _owner, address indexed _spender, uint _value);
15 }
16
17 // File: contracts/KyberNetworkInterface.sol
18

```

#### Select Contract Name

KyberNetworkProxy

You can also change the following compiler parameters for the simulation execution:

- Compiler Version
- Optimization Used
- Optimization Count
- EVM Version

You can add a custom contract source to any contract you have chosen to edit or any address.

## Forks

Tenderly Forks are a lightweight blockchain replica where you can run transaction simulations. You can base your Fork on one of [Tenderly's supported networks](#) and any of the blocks in a specific network's history. All transactions you simulate through Forks are recorded in an isolated timeline.

Additionally, you can do various custom actions on a Fork:

- [Advance/mine a block](#) (`evm_increaseBlocks`)
- [Advance time](#) on the Fork (setting the `timestamp`)
- [Move the head](#) of the Fork (`evm_snapshot` and `evm_revert`)
- [Manage account balances](#) (`tenderly_setBalance` and `tenderly_addBalance`)
- [Override smart contract storage](#) (`tenderly_setStorageAt`)

The screenshot shows the Tenderly web interface with a "Test Fork" selected. The main area displays a table of simulated transactions. The columns are: Simulation Id, Tx Hash, From, To, and Function. The transactions listed are:

Simulation Id	Tx Hash	From	To	Function
06436995-3391-4f4f-a8b0-c168d3103bd7 Add a note	0x997a7b6c...b115	0xf7ddedc6...e5dd	Dai	transferF
185d77a5-dbf5-4068-a82e-8828b5912709 Add a note	0x1de69eee...5f8c	0xe58b9ee9...8ce1	Dai	approve
00ea4f42-4921-4195-a713-766f3f838a95 Add a note	0x60c96da5...6af7	0xf2f2f2f2...f2f2	Dai	mint

Below the table, there are pagination controls: "Show 20 Per Page" and "Page 1".

## CI / CD

See [Docs](#)

The docs have examples of using a fork for your CI, and an example github action workflow yaml file.