CHAINSECURITY polygon

Top Hacks since Devcon VI

what did we learn?

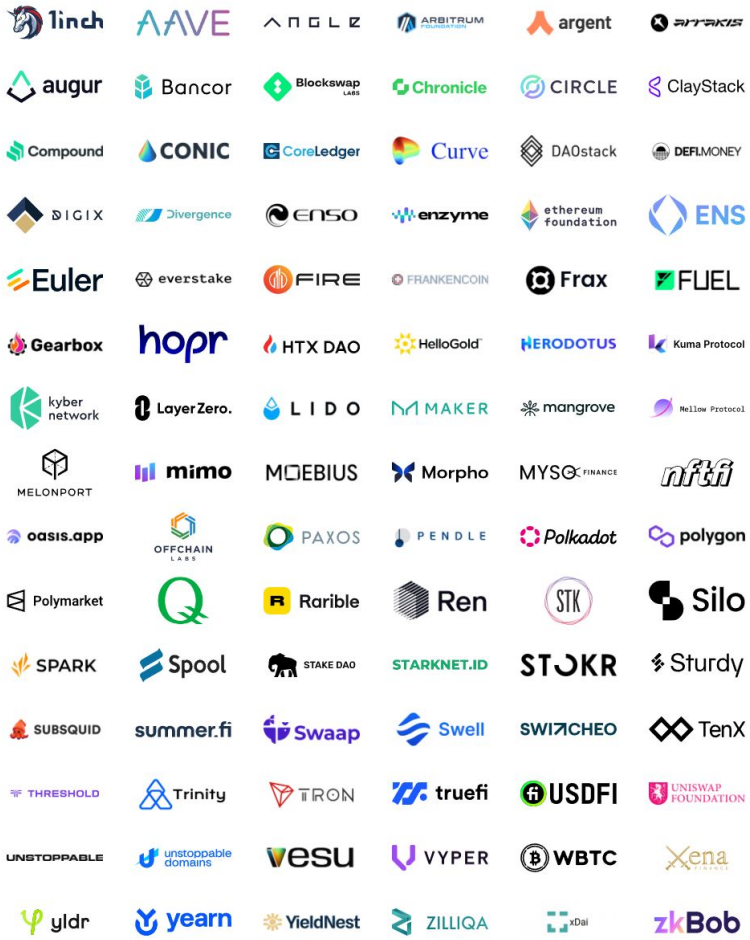
CHAINSECURITY

Who is talking?

Someone from ChainSecurity

Founding partner & CTO of ChainSecurity, a long-time blockchain security company which audited a bunch of projects over the last 7 years.

ChainSecurity caused the Constantinople upgrade to be delayed and a bunch of Berlin upgrade gas repricings, sometimes uncovers novel vulnerabilities like read-only reentrencies and overall likes to help to secure the most complex systems out there.



CHAINSECURITY

Who is talking?

Someone from polygon

CISO from that old L2 before it was called an L2.

Formerly core dev for Polymath & SushiSwap.
Prolific sharer of security knowhow on X and TG.

Full disclosure: Investor into competitors of ChainSecurity

CHAINSECURITY polygon

Who is talking?

You!

Devcon made this a workshop! One of the most effective ways to improve security is to share experiences, so this is what we are going to do:

We invite **you** to share your own experience and learnings!

CHAINSECURITY polygon

DC7 SEA

What did you learn?

Lets collect!

Participate by scanning the QR code and by adding your learnings in the document. It is using fileverse.io (no account needed, e2e encrypted, anonymous, *alpha software*)

CHAINSECURITY polygon

DC7 SEA

What did you learn?

Lets collect!

Participate by scanning the QR code and by adding your learnings in the document. Its a google doc and you need to request edit access.

CHAINSECURITY polygon

DC7 SEA

DMM Bitcoin

CEX Wallet ⅔ Multisig compromised

One hot wallet of the exchange DMM Bitcoin was compromised, resulting in a loss of $300m+ in BTC. No technical details have been released. The circumstances indicate that the keys itself haven't been lost, but that a malicious transaction has been signed. The attacker did not fully empty the account, leaving >300 BTC back. The attacker used a destination address with a similar starting and ending address.

The funds have later been moved through various exchanges and mixers, a pattern often observed by Lazarus Group / North Korea related hacks.

Exploit Steps

- April 4th, 2024: 4.3k BTC are split into 9 large unspents + multiple smaller ones.
- May 31st, 2024: The 9 large unspents are transferred to the attacker address
- 50 min later: The remaining 343 BTC are secured

CHAINSECURITY polygon

DC7 SEA

DMM Bitcoin

What did we learn?

Centralized Exchanges remain the single biggest threat as measured in value lost.

This incident likely involved insiders knowledge and internal validate errors as the first 5 characters of the address are identical to a commonly used address by DMM Bitcoin.

What can be done to make the attack harder?

- Splitting up funds across many wallets with separate keys
  - Be careful not to lower the security of each individual set of keys
- Establish standard to allow for full comparison of hashes with very high costs to brute-force

CHAINSECURITY polygon

DC7 SEA

WazirX

Safe Wallet Malicious Upgrade

WazirX, a centralized exchange, was tricked into signing an upgrade for their Safe multisig to a version controlled by the attackers. The attackers withdrew all assets and laundered them in ways generally attributed to Lazarus Group / North Korea.

WazirX found no compromise of the laptops used to sign, and Liminal found no compromise of their infrastructure. No phishing was involved according to WazirX.

Exploit Steps

- Attacker buys GALA on WazirX, exhausting the hot wallet, triggering a refill from the cold wallet
- Signer #1 tries to initiate a GALA refill from the WazirX Safe using Liminal, signs malicious Safe upgrade instead
- Signer #2 tries the same, also signs malicious Safe account upgrade
- Signer #3 tries to confirm a benign USDT transfer, signs malicious Safe account upgrade
- All 3 signatures are sent to Liminal, which co-signs and publishes the transaction
- The attacker now has full control over the Safe and promptly withdraws all funds and starts to launder them.

CHAINSECURITY polygon

DC7 SEA

WazirX

What did we learn?

Large wallets holding significant funds get easily drained when the signers are tricked into signing a malicious transaction. Due to the frequency of errors even during normal transactions, even sophisticated users aren't alerted by failures. Adding third party systems without deep understanding of the security they provide can lead to a false sense of security.

What can be done to make the attack harder?

- Don't introduce single points of failure in any part of the stack - aim for 3-star-independence!
  - Single person?
  - Single "action source"?
  - Single comm channel?
  - Single UI?
  - Single brand of laptop?
  - Single brand of HSM?
  - Single type of operating system?
  - Single internet connection?
- Consider unknown errors as hack attempts
  - Prepare action plan on hack attempts

CHAINSECURITY

DC7 SEA

Teaser for Thu 5:30pm

“How to use Safes securely!”

Lazarus has stolen by far the most funds in the blockchain space. They use the same or very similar attack vectors every time yet we see the biggest crypto companies falling victim to them one after another. In this talk, Mudit will go over some of the attack vectors used by Lazarus and how people can keep themselves safe from Lazarus.

- Use hardware wallets
- Do not share your mnemonic/seed phrase/private key with anyone
- Connect hardware wallets via browser wallets
- Use a dedicated (semi-airgapped) device for signing
- Verify what you’re signing on the browser wallet
- Ensure secure threshold for multisigs
- Verify hash on hardware wallets (safe-tx-hashes-util from pcaversaccio)
- Use diverse set of hardware including mobile devices for signers
- Do not ignore failures

CHAINSECURITY polygon

DC7 SEA

Gala Games

Insider: Old MINTER address abused

Gala Games suffered from a malicious privileged minter minting 5 billion GALA tokens (valued at 200m) and converted $22m into ETH before the rest was frozen by the project. The attacker used a dormant 6 month old address with MINTER permissions. The money was returned on the next day.

Preceding the attack were internal conflicts at Gala Games, resulting in a key employee leaving the project.

Exploit steps

- Gained access to a dormant MINTER account on the GALA contract.
- Minted 5 billion GALA tokens to a new “Gala Game Exploiter” address.
- Exchanged minted GALA tokens for ETH, draining the pool.

CHAINSECURITY polygon

DC7 SEA

Gala Games

What did we learn?

The exploit highlights how dormant permissions and a lack of robust access control on high-privilege accounts can expose projects to significant risk.

Security must be continually reinforced at both the technical and operational levels to prevent similar vulnerabilities.

Never neglect the “people risk”.

What can be done to make the attack harder?

- Enforce multi-factor authentication and multi-party approvals for all high-privilege accounts and high-risk actions.
- Conduct routine audits to remove unused or dormant account permissions.
- Implement transaction rate limits for significant actions (e.g., minting).
- Monitor privileged accounts for unusual activities in real time.

CHAINSECURITY polygon

DC7 SEA

Mixin Network

Cloud database hack

The BTC, ETH and USDT holdings valued around $150m of Mixin network were drained by an attacker who gained access to the private keys, stored in a cloud database, likely with Google Cloud. In total, more than 10’000 wallets were drained over more than 1 hour. The notice that something was wrong went out 2 days later to the users.

Exploit steps

- Gain access to the cloud database of private keys (unknown how that was possible)
- Withdraw all funds across 10k wallets for 1h+
- Get 2 days to hide before being detected for free

CHAINSECURITY polygon

DC7 SEA

Mixin Network

What did we learn?

Segregating wallets by multiple private keys is no use if an attacker can get access to all those private keys. Absence of monitoring with the ability to automatically secure funds made this attack devastating.

What can be done to make the attack harder?

- Don’t store private keys in plain text in a cloud database
  - Or any other system you don’t fully control yourself with excellent physical and digital security - specialized dedicated HSMs are the go-to solution here

Key compromises are responsible for >50% of all $$$ hacked in 2024

CHAINSECURITY polygon

DC7 SEA

Euler Finance

Self-liquidation

Through a late addition to the protocol, allowing to donate token to pools, it was possible to create underwater positions, which then could be (self-)liquidated for a significant profit. Looping this allowed the attacker to drain around $200m.

The funds were later returned by carefully putting on pressure on the attacker, eventually leading to a full recovery, and later a relaunch going to extremes in regards to security.

Exploit steps

- Most-analyzed hack of the century.
  - Overview: https://youtu.be/3wZkidoyFko?t=234
  - Details focused on code: https:/youtu.be/vleHZqDc48M
  - The writeup the the century: https://www.euler.finance/blog/war-peace-behind-the-scenes-of-eulers-240m-exploit-recovery

CHAINSECURITY polygon

DC7 SEA

Euler Finance

What did we learn?

Auditing changes to a complex protocol is fundamentally hard. They are almost always underestimated in complexity (“2 weeks for a diff of 10 lines???Rip-off!”)

Security happens not only before a hack, but also after, and has many dimensions, from technical to psychological. Both on the attackers side and on your own, on all of these dimensions.

What can be done to make the attack harder?

- Audit diffs with a mindset of a complete audit
- Keep track of all assumptions of a protocol on a detailed level and re-use them for later audits
- Monitoring with automatic responses can defend against some attackers, especially individuals.
- Preparing with playbooks for the aftermath of an eventual hack can enable a less overpowering phase for the defenders

“*No plan survives contact with the enemy*”
- Helmuth von Moltke, Prussian military

Skipping:

Multichain, another $126m insider/key compromise with little info

Skipping:

Poloniex, another $126m key compromise with little info

CHAINSECURITY polygon

DC7 SEA

BonqDAO

Oracle manipulation

BonqDAO is a Polygon-based lending and stablecoin protocol and was exploited in an oracle manipulation attack. The attacker leveraged a Tellor price oracle vulnerability, inflating and then deflating the price of ALBT to manipulate collateral and liquidate user assets, causing $120 million in on-chain damages. Despite the scale of the exploit, the attacker managed to realize only ~$2 million in ETH and DAI due to low liquidity.

Exploit steps

- Attacker staked 10 TRB tokens to gain access to Tellor's oracle and set the WALBT price.
- Inflated ALBT collateral price, then minted 100M BEUR (Euro-pegged stablecoin) against 0.1 WALBT.
- Subsequently deflated the WALBT price to near-zero, triggering liquidation of user assets.
- Liquidated user collateral, resulting in an additional 113M WALBT gained by the attacker.
- Attacker converted stolen BEUR and ALBT to ETH and DAI, securing ~$1.7M in proceeds.

CHAINSECURITY polygon

DC7 SEA

BonqDAO

What did we learn?

You really need to understand the underlying infrastructure you build on top of and their security assumptions. Especially oracles.

Following advice of auditors or other whitehats without having the solution be re-audited holistically is a major source of criticals.

What can be done to make the attack harder?

- Use rock solid oracles which are extremely well understood by you and your auditors
- Use multiple oracles
- Have overall hard caps in your lending protocol which are manually updated
- Perform holistic audits to include any new contract implementations post-deployment.

Skipping:

Atomic Wallet, $100m+, unknown reason for parallel private key compromise of multiple users

Skipping:

HECO Bridge, HTX, $91m, unknown reason for operator and hot wallet private key compromise

CHAINSECURITY polygon

DC7 SEA

WOOFi

Flash loan attack

WOOFi, a cross-chain DEX on Arbitrum, was exploited for $8.5m. The attack used the ability to flash-loan very high USDC amounts, the availability of large amounts of WOO to flash-loan and to borrow, the absence of the fallback price oracle, and the design of WooFi which allows for profits in back-and-forth swaps in some circumstances.

The attack was immediately picked up, and kicked of a very fast manual response by SEAL911 and others, resulting in the protocol being paused after 13 minutes, but another two attack transactions succeeded before that.

Exploit steps

- Flash loan ~10.6m USDC from Uniswap and 2.7m WOO from TraderJoe
- Deposit 7m USDC in Silo, to get a total of 7.7m WOO.
- Oracle Manipulation through Swaps:
  - USDC -> WETH: Raise USDC reserves.
  - USDC -> WOO: Continue reserve build-up.
  - WOO -> USDC: Dump 7.7m Woo, which is now valued at 0.00000009 USDC
  - USDC -> WOO: Repurchase WOO at the manipulated minimal price.
- Repay loans and convert the stolen funds to ETH and other assets.

WOOFi

What did we learn?

On an audit, it is really important to take extreme values into consideration. It is very likely that over the lifetime of a protocol some of these extreme settings will become real.

Having fallbacks can prevent hacks, but if they are frequently the last thing that stops an attack, is a matter of time until also that barrier is overcome.

It is dangerous to deploy when fallbacks aren't available.

What can be done to make the attack harder?

- Configure hard limits for extreme values which are known to still be safe
- Automatically pause a protocol
- Use deployment validation to ensure safe configuration states
- Do holistic audits

Skipping:

Orbit Bridge, $81.5m, private key compromise of operator

CHAINSECURITY polygon

DC7 SEA

WOOFi

Flash loan attack

WOOFi, a cross-chain DEX on Arbitrum, was exploited for $8.5m. The attack used the ability to flash-loan very high USDC amounts, the availability of large amounts of WOO to flash-loan and to borrow, the absence of the fallback price oracle, and the design of WooFi which allows for profits in back-and-forth swaps in some circumstances.

The attack was immediately picked up, and kicked of a very fast manual response by SEAL911 and others, resulting in the protocol being paused after 13 minutes, but another two attack transactions succeeded before that.

Exploit steps

- Flash loan ~10.6m USDC from Uniswap and 2.7m WOO from TraderJoe
- Deposit 7m USDC in Silo, to get a total of 7.7m WOO.
- Oracle Manipulation through Swaps:
  - USDC -> WETH: Raise USDC reserves.
  - USDC -> WOO: Continue reserve build-up.
  - WOO -> USDC: Dump 7.7m Woo, which is now valued at 0.00000009 USDC
  - USDC -> WOO: Repurchase WOO at the manipulated minimal price.
- Repay loans and convert the stolen funds to ETH and other assets.

WOOFi

What did we learn?

On an audit, it is really important to take extreme values into consideration. It is very likely that over the lifetime of a protocol some of these extreme settings will become real.

Having fallbacks can prevent hacks, but if they are frequently the last thing that stops an attack, is a matter of time until also that barrier is overcome.

It is dangerous to deploy when fallbacks aren't available.

What can be done to make the attack harder?

- Configure hard limits for extreme values which are known to still be safe
- Automatically pause a protocol
- Use deployment validation to ensure safe configuration states
- Do holistic audits

CHAINSECURITY polygon

DC7 SEA

Curve/Vyper

Compiler Bug + Reentrancy

Vyper had a compiler bug in the way storage locations are computed which manifested itself in the build-in reentrancy guard when used in a certain way. Several older Curve pools were affected as they could be manipulated with the fall of the reentrancy guard, most notably the CRVUSD pool. A total of $69m was hacked across affected pools.

After the first exploits and public tweets about the vulnerability, the Curve team, Vyper, and members of the security community rushed to protect further pools, but couldn't prevent a few further hacks on time.

Exploit steps

```
@external
@nonreentrant('lock')
def remove_liquidity(_amount: uint256, min_amounts: uint256[N_COINS], use_eth: bool = False):
    """
    This withdrawal method is very safe, does no complex math
    """
    _coins: address[N_COINS] = coins
    total_supply: uint256 = CurveToken(token).totalSupply()
    CurveToken(token).burnFrom(msg.sender, _amount)
    balances: uint256[N_COINS] = self.balances
    amount: uint256 = _amount - 1  # Make rounding errors favoring other LPs a tiny bit

    for i in range(N_COINS):
        d_balance: uint256 = balances[i] * amount / total_supply
        assert d_balance >= min_amounts[i]
        self.balances[i] = balances[i] - d_balance
        balances[i] = d_balance  # now it's the amounts going out
        if use_eth and i == ETH_INDEX:
            raw_call(msg.sender, b"", value=d_balance)
        else:
            if i == ETH_INDEX:
                WETH(_coins[i]).deposit(value=d_balance)
            assert ERC20(_coins[i]).transfer(msg.sender, d_balance)

    D: uint256 = self.D
    self.D = D - D * amount / total_supply

    log RemoveLiquidity(msg.sender, balances, total_supply - _amount)
```


```vyper
@external
@nonreentrant('lock')
def remove_liquidity(_amount: uint256, min_amounts: uint256[N_COINS], use_eth: bool = False):
    """
    This withdrawal method is very safe, does no complex math
    """
    _coins: address[N_COINS] = coins
    total_supply: uint256 = CurveToken(token).totalSupply()
    CurveToken(token).burnFrom(msg.sender, _amount)
    balances: uint256[N_COINS] = self.balances
    amount: uint256 = _amount - 1  # Make rounding errors favoring other LPs a tiny bit

    for i in range(N_COINS):
        d_balance: uint256 = balances[i] * amount / total_supply
        assert d_balance >= min_amounts[i]
        self.balances[i] = balances[i] - d_balance
        balances[i] = d_balance  # now it's the amounts going out
        if use_eth and i == ETH_INDEX:
            raw_call(msg.sender, b"", value=d_balance)
        else:
            if i == ETH_INDEX:
                WETH(_coins[i]).deposit(value=d_balance)
            assert ERC20(_coins[i]).transfer(msg.sender, d_balance)

    D: uint256 = self.D
    self.D = D - D * amount / total_supply

    log RemoveLiquidity(msg.sender, balances, total_supply - _amount)
```

CHAINSECURITY polygon

DC7 SEA

Curve/Vyper

What did we learn?

Compiler bugs are one of the hardest stress tests for the ecosystem due to their outsized impact. The initial hacker didn't realize the impact and only follow-up hacks exploited it to the maximum.

Learnings from this hack led to the founding of SEAL for coordinated security response.

What can be done to make the attack harder?

- Increase eyes on security on infrastructure
- Develop ability for fast detection and proactiv rescue of vulnerable contracts

CHAINSECURITY polygon

DC7 SEA

Munchables

Malicious deployment

Munchables was hacked for $62m on Blast after the employee who deployed the contract dumped their hidden tokens. During deployment, the attacker first used an unverified implementation contract to mint themselves tokens without emitting an event, and later updated to the proper implementation.

All funds were recovered by joined effort to uncover the identity of the rough dev and by preventing effective exit the Ethereum through the bridge, including a threat to halt the bridge or chain.

Exploit steps

- Deploy Lock proxy contract
- Initialize proxy to malicious unverified implementation contract
  - Manipulate storage to secretly mint 1m ETH balance to the attacker
- Update implementation to benign version
- Wait for significant value locked in the protocol, followed by withdrawing the ETH from the Lock contract

CHAINSECURITY polygon

DC7 SEA

Munchables

What did we learn?

Hiring rogue devs can compromise a project in many ways. Trusting other unverified sources for a devs credibility doesn't help.

Validating deployments to ensure no malicious (or accidental) changes have been done is crucial.

For insiders, putting on significant pressure and a public hunt can be very effective if there are enough initial leads.

What can be done to make the attack harder?

- Shameless plug: Use Deployment Validation https://github.com/ChainSecurity/deployment_validation
  - It's free, open source, and looking for contributors
- Do detailed due diligence on privileged protocol members
- Keep track of exit routes for an attacker and keep channels open to restrict those as needed

Skipping:

AlphaPo, $60m, private key compromise of the hot wallets

Skipping:

Crypto Whale, $55m, signing a malicious signature transferring ownership of the DSProxy

Skipping:

CoinEx, $54m, private key compromise of hot wallet

CHAINSECURITY polygon

DC7 SEA

Radiant Capital

Safe signature phishing

Radiant capital was hacked by 3 of their 11 signers being tricked into signing a malicious upgrade of the multisig Safe to a contract under the attackers control. The attacker was able to use these signatures across multiple chains for a profit of $53m, including being able to circumvent the pausing of the protocol.

Later investigation showed that 3 core contributors machines were compromised to sign malicious transfers while the Safe website showed a benign transaction.

Exploit steps

- Attacker deployed smart contracts later used to control the address 2 weeks before the hack
- During the two weeks, they managed to harvest 3 signatures from multisig signers
- On receiving all three signatures, they used transferOwnership on the Safe, followed by an upgrade of the implementation contracts
- All funds were withdrawn from the now compromised pools

CHAINSECURITY polygon

DC7 SEA

Radiant Capital

What did we learn?

The attack is very similar to what hit WazirX. Having a large multisig with a low threshold increases the attack surface, and having dual-use devices for signing increases it further.

It is not clear yet how the developers have actually been compromised, as the means to do so by state actors are very broad. 0days for popular software used on developer machines are relatively cheap.

What can be done to make the attack harder?

- Quoting Peter: “Buy your devs cheap chromebooks for signing”
- Increase the signing threshold and lower the amount of signers overall
- Implement potentially even automated extreme fallbacks beyond pausing to withdraw funds quickly to cold wallets

CHAINSECURITY polygon

DC7 SEA

KyberSwap

Rounding errors on liquidity boundaries

KyberSwap was hacked for $48m. KyberSwap uses tick-based liquidity, and a rounding error allowed the attacker to double-count liquidity in a certain area by careful manipulation of the liquidity.

After the attack, Kyber offered a 10% “whitehat” bounty, and the hacker asked for full control over Kyber via on-chain messages. The attacker is linked to previous sophisticated smart contract hacks.

Together with security researchers, Kyber recovered around $5m of the funds.

Exploit steps

- Flash loan 500 ETHx
- Swap on the ETHx->USDC pool, mostly draining it, to a very specific tick
- Provide the exact liquidity through a new position and further partial withdrawing of it to set the stage for the rounding error
- Swap a precise amount getting to the price edge of liquidity boundary. Final state tick is computed with a rounding error and outside the liquidity boundary. Boundary was effectively crossed without liquidity updates.
- Swap back, crossing the liquidity boundary again, wrongly doubling the virtual liquidity, and allowing the attacker to profit significantly.
- Repay flashloan

CHAINSECURITY polygon

DC7 SEA

KyberSwap: Deployer to KyberSwap Exploiter 1

We have reached out to law enforcement and cybersecurity on this case. We have your footprints to track you.

So it's better for you if you take the first offer from our previous message before law enforcement and cybersecurity track you down.

If the situation is not Fixed, a bounty offer will Float to the community instead.

If you don t return 90% of the funds you took from users to 0x8180a5CA4E3B94045e05A9313777955f7518D757 by 10am UTC, 27 November, we will also initiate a public bounty program to incentivize anyone who provides additional information to support law enforcement and cybersecurity in your arrest and the recovery of users funds.

at txn 0xf4159cab74bba... Nov-25-2023 12:58:11 PM UTC (353 days ago)

KyberSwap: Deployer to KyberSwap Exploiter 1

You have done one of the most sophisticated hacks ser. That was high EV and everyone missed it. On the table is a bounty equivalent to 10% of users' funds taken from them by your hack, for the safe return of all of the users' funds.
But we both know how this works, so lets cut to the chase so you and these users can all get on with life.
We can set aside the funds for the bounty to distribute directly to users. You can return 90% of the funds you took from users to 0x8180a5CA4E3B94045e05A9313777955f7518D757 by 6am UTC, 25 November, or you stay on the run.
If you want to discuss in private, email vutran54@proton.me

at txn 0xfeb8bd91320ed... Nov-24-2023 05:37:11 AM UTC (354 days ago)

KyberSwap Exploiter 1 to KyberSwap: Deployer

Dear Kyberswap Developers, Employees, DAO members and LPs,

Negotiations will start in a few hours when I am fully rested.

Thank you.

at txn 0x7a8912583520... Nov-22-2023 11:57:11 PM UTC (356 days ago)

CHAINSECURITY polygon

DC7 SEA

Txn Type: 2 (EIP-1559) | Nonce: 14 | Position In Block: 48

To ALL relevant and/or interested parties,

I thank you for your attention and patience during this uncertain time for Kyber (the protocol/DAO) as well as Kyber (the company). Below I have delineated a treaty for us to agree to.

My demands are as follows:

* Complete executive control over Kyber (the company)

* Temporary full authority and ownership over the governance mechanism (KyberDAO) in order to enact legislative changes. My current wallet address is fine for this.

* All documents and information related to company / protocol formation, structure, operation, revenues, profits, expenses, assets, liabilities, investors, salaries, etc.

* Surrender of all Kyber (the company) assets. This is both On-chain and Off-chain assets. It includes but is not limited to: shares, equity, tokens (KNC and non-KNC), partnerships, blogs, websites, servers, passwords, code, social channels, any and all creative and intellectual property of Kyber.

CHAINSECURITY polygon

DC7 SEA

KyberSwap

What did we learn?

Rounding errors remain a hard problem in smart contracts, often very subtle and hard to detect with tools, formal verification or manual analysis.

Complex optimizations with opposing goals like “give the best price” and “ensure solvency” with the same mechanism makes reviewing the code for correctness hard.

Post-exploit support by auditors and the security community is crucial to secure remaining funds and partial recovery.

What can be done to make the attack harder?

- Design smart contracts defensively to facilitate adhering to critical safety properties more easily
- Plan for significantly longer duration of a manual audit to allow to manually explore deep state in systems with complex state machines

Skipping:

BinX, $44m, private key compromise of hot wallet

CHAINSECURITY polygon

Time left?

Opening the floor: Let's hear about your top hack since last Devcon!

Do you have an insightful hack which didn't make it to the top 20, but is something watch and learn from? Explain it shortly, and let's come up with learnings together!

Thank you for participating in

“Top Hacks since Devcon VI:
what did we learn?”