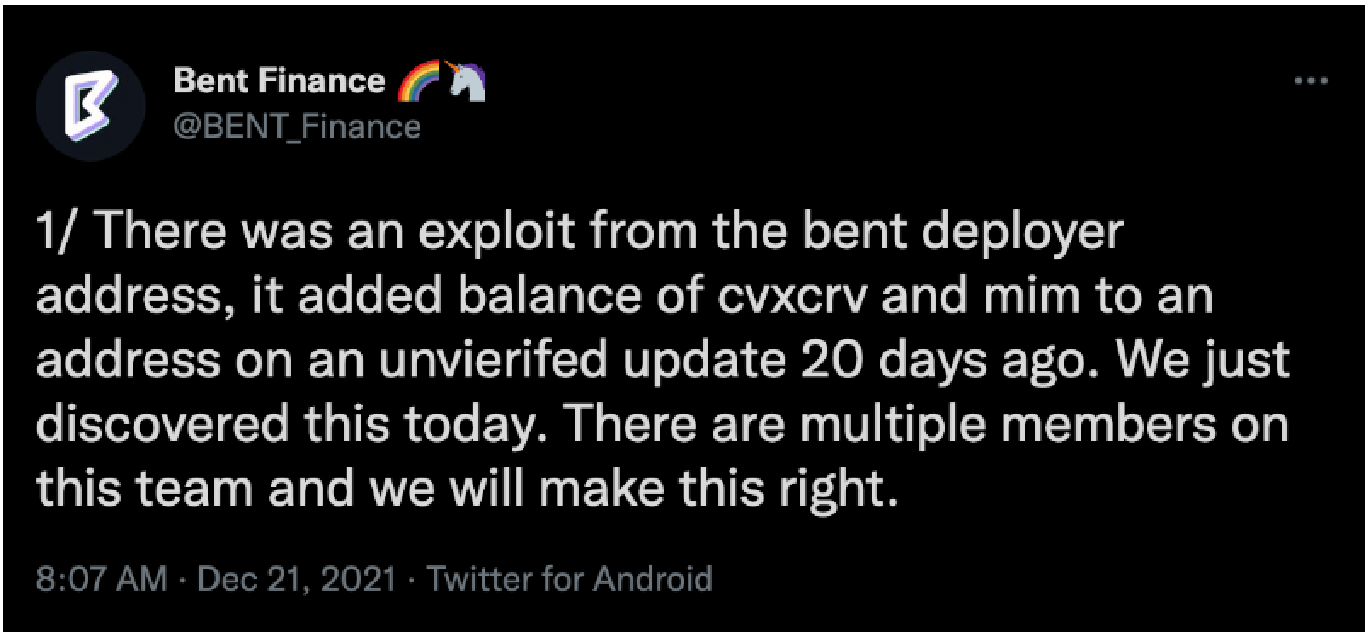
DeFi - Compound and Oracles

Recent Security Exploits

Bent Finance



Bent Finance 🌈🦄
@BENT_Finance

1/ There was an exploit from the bent deployer address, it added balance of cvxcrv and mim to an address on an unvierifed update 20 days ago. We just discovered this today. There are multiple members on this team and we will make this right.

8:07 AM · Dec 21, 2021 · Twitter for Android

Steps

- Nov 30 - The contract was manually updated to increase the balance of the deployer address. The amount exceeded the TVL of the project
- The exploit was live for 3 weeks but only became public knowledge when the Bent Finance profile was added to to DeBank and the balance could be seen.
- Tokens were withdrawn on Dec 12 and Dec 21, were swapped for ETH and sent to Tornado Cash

Compromised Key Exploits

Ascendex loses $77M

https://rekt.news/ascendex-rekt/ (https://rekt.news/ascendex-rekt/)

Vulkan Forged $140M

https://rekt.news/vulcan-forged-rekt/ (https://rekt.news/vulcan-forged-rekt/)

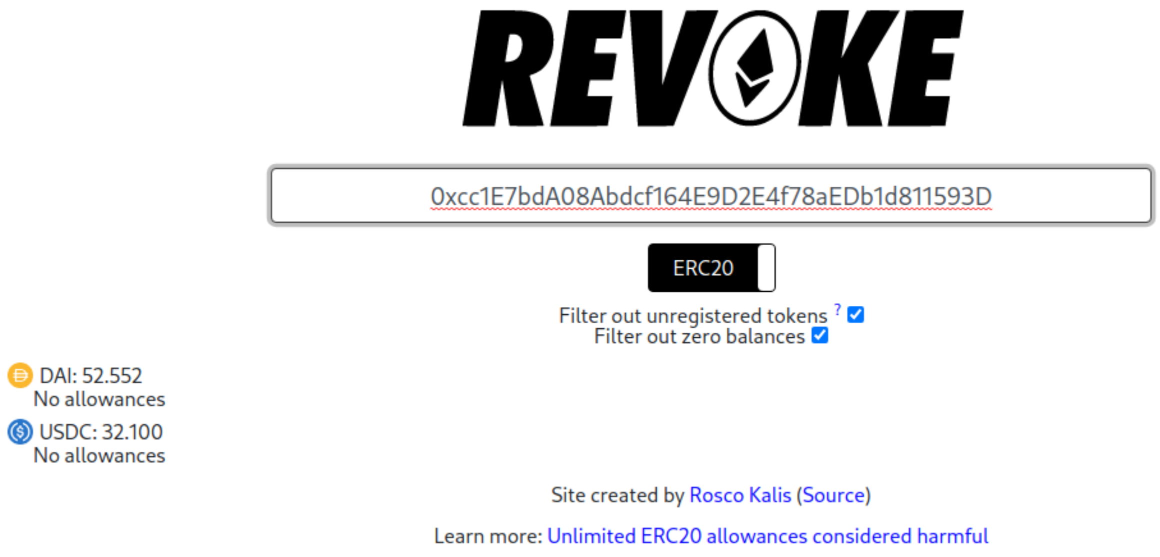
See Compromised Keys Medium Article (https://extropy-io.medium.com/compromised-keys-hacks-51d4739b77d0)

Prevention

For Users

Know what you are approving
Know how much you are approving
Be aware of approvals on proxies

Review your approvals : https://revoke.cash (https://revoke.cash)


REVOKE

0xcc1E7bdA08Abdcf164E9D2E4f78aEDb1d811593D

ERC20

Filter out unregistered tokens ? ✓
Filter out zero balances ✓

DAI: 52.552
No allowances
USDC: 32.100
No allowances

Site created by Rosco Kalis (Source)

Learn more: Unlimited ERC20 allowances considered harmful

If you are doing an infinite approval, you should have a good reason for it. It should not be your default.

For Developers

Use

- Role based access control
- Timelocks
- Governance
- Secure All Administrative Keys
- Use multiple signatures for critical administrative tasks

Review the code and question

- Who controls the privileged account? A single private key? Multiple keys with a multisig? A governance module?

- Does the privileged account hold any other role in the system? If so, could that introduce any risk worth analyzing?
- How are the private keys stored? Are there backups? Who’s got access to them? Are access operations logged, monitored and audited?
- How are the private keys generated? Do they sign transactions in standard ways or is the system using some custom implementation?

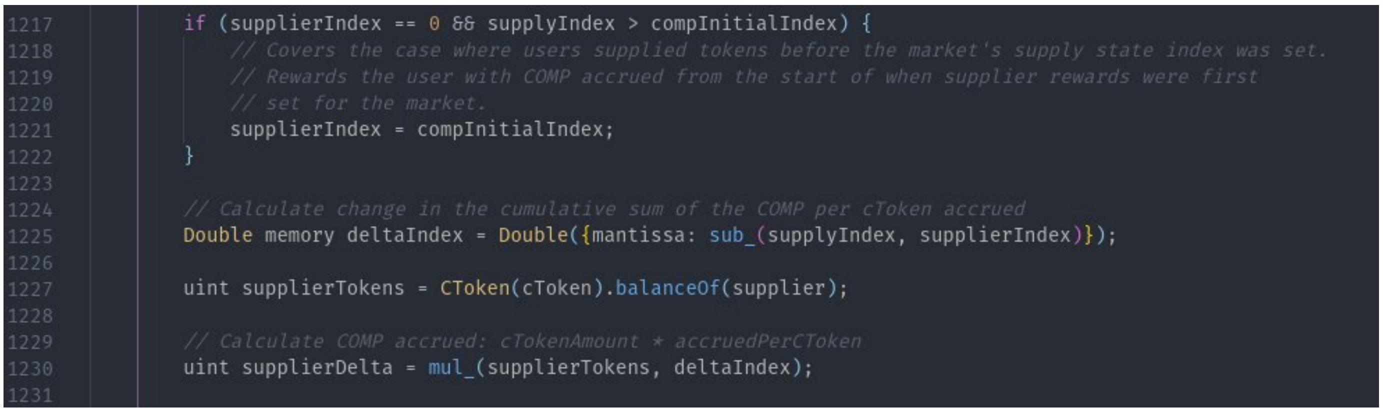
Monitor

- Use of priviledged keys
- Spikes in Account Activity
- Drop in system funds

For more information about these exploits and others see
Rekt News (https://rekt.news)
Extropy Security Bulletin (https://security.extropy.io/The-Security-Bulletin.html)
Compromised Keys Medium Article (https://extropy-io.medium.com/compromised-keys-hacks-51d4739b77d0)

Compound

See https://rekt.news/overcompensated/ (https://rekt.news/overcompensated/)
A vulnerability in one of the cornerstones of DeFi
The vulnerability was introduced / triggered following a
governance proposal (https://compound.finance/governance/proposals/62)



```
1217        if (supplierIndex == 0 && supplyIndex > compInitialIndex) {
1218            // Covers the case where users supplied tokens before the market's supply state index was set.
1219            // Rewards the user with COMP accrued from the start of when supplier rewards were first
1220            // set for the market.
1221            supplierIndex = compInitialIndex;
1222        }
1223
1224        // Calculate change in the cumulative sum of the COMP per cToken accrued
1225        Double memory deltaIndex = Double({mantissa: sub_(supplyIndex, supplierIndex)});
1226
1227        uint supplierTokens = CToken(cToken).balanceOf(supplier);
1228
1229        // Calculate COMP accrued: cTokenAmount * accruedPerCToken
1230        uint supplierDelta = mul_(supplierTokens, deltaIndex);
1231
```

A “>” should have been a “>=”

Mudit Gupta provided the analysis (https://twitter.com/Mudit__Gupta/status/1443454935639609345)

“The bug happens when someone supplies tokens for a market with zero comp rewards like cSUSHI, and cTUSD before the market is initialized or migrated.
`supplyIndex` for such tokens remains equal to `compInitialIndex` which means that the if block on L1217 is not triggered.

The last version of comptroller had the same checks but it was fine then because the initial value of `supplyIndex` was 0 rather than 1e36.
Logically, the check should have been `>=` even then but since the default was 0, `>` was functionally equivalent but a bit more optimal."

The effect of this was that ~$80M of COMP was distributed erroneously, the comptroller fund now only has about $250k left.

Because of the way that Compound governance works, it will be 7 days before a fix can be effected, and it is already too late.

The audit process was questioned, this exploit shows that audits need to re examine the whole protocol, rather than just the change.

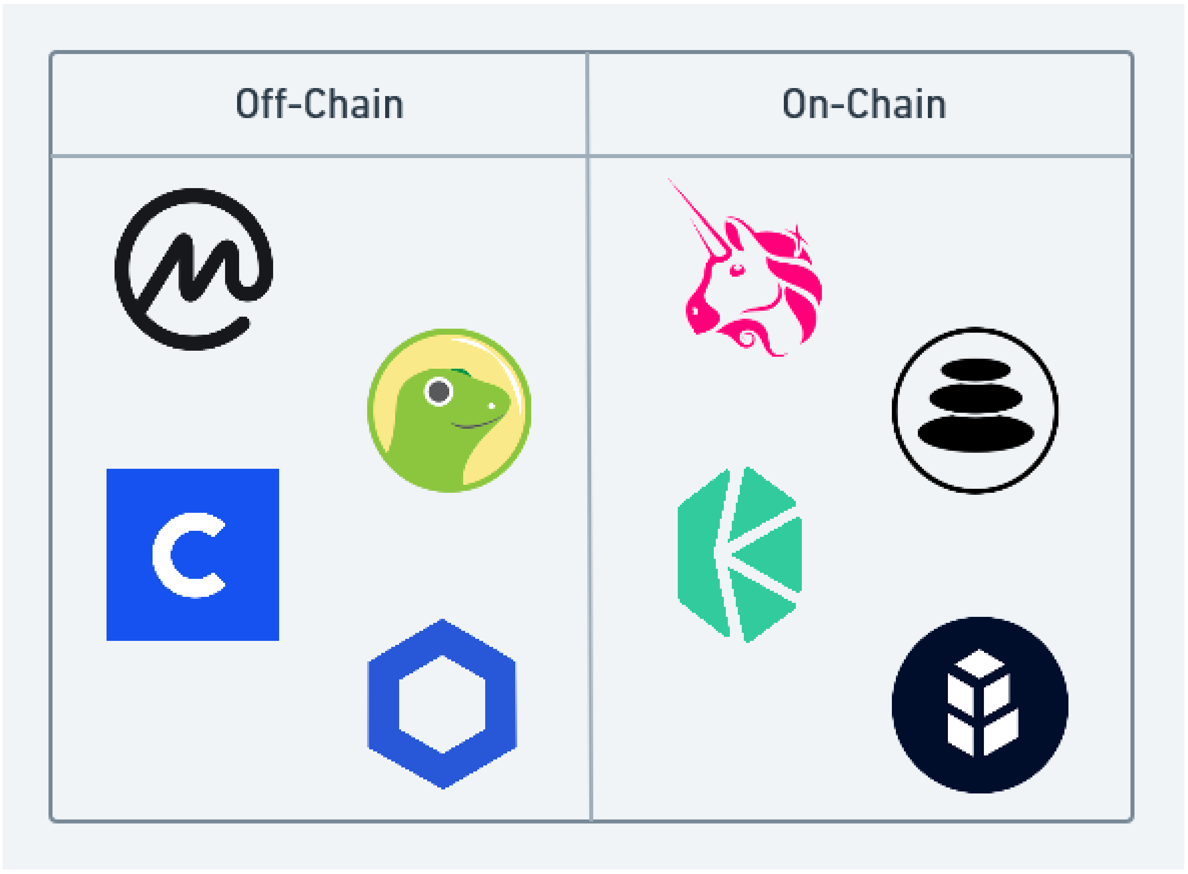
"I used to be one of the biggest proponents of upgradable smart contracts.
However, over time, I've come to see upgradablity as more of a bug than a feature.
It's still good in some scenarios but probably not great for large primitives like Compound, Aave, Uni, Sushi, Maker etc."

- Mudit Gupta

Mudit has a good introduction to contract auditing (https://www.youtube.com/watch?v=LLiJK_VeAvQ)

Oracles

We can split oracles into on chain and off chain
You can take the existing off-chain price data from price APIs or exchanges and bring it on-chain, alternatively, you can calculate the instantaneous price by consulting on-chain decentralized exchanges.



Off-chain data is generally slower to react to volatility, it typically requires a handful of privileged users to push the data on-chain.
On-chain data doesn’t require any privileged access and is always up-to-date, but this means that it’s easily manipulated by attackers.

There is also the questions of degree of centralization

Off-chain Centralized Oracle

This type of oracle simply accepts new prices from an off-chain source, typically an account controlled by the project. Due to the need to quickly update the oracle with new exchange rates, the account is typically an EOA and not a multisig. There may be some sanity checking to ensure that prices don’t fluctuate too wildly. Compound Finance (https://compound.finance/) and Synthetix (https://www.synthetix.io/) mainly use this type of oracle for most assets

Off-chain Decentralized Oracle

This type of oracle accepts new prices from multiple off-chain sources and merges the values through a mathematical function, such as an average. In this model, a multisig wallet is typically used to manage the list of authorized sources. Maker (https://makerdao.com/feeds/) and Chainlink (https://chain.link/) use this type of oracle for ETH and other assets.

On-chain Centralized Oracle

This type of oracle determines the price of assets using an on-chain source, such as a DEX. However, only a central authority can trigger the oracle to read from the on-chain source. Like an off-chain centralized oracle, this type of oracle requires rapid updates and as such the triggering account is likely an EOA and not a multisig. dYdX (https://dydx.exchange/) and Nuo (https://nuo.network/) use this type of oracle for certain assets

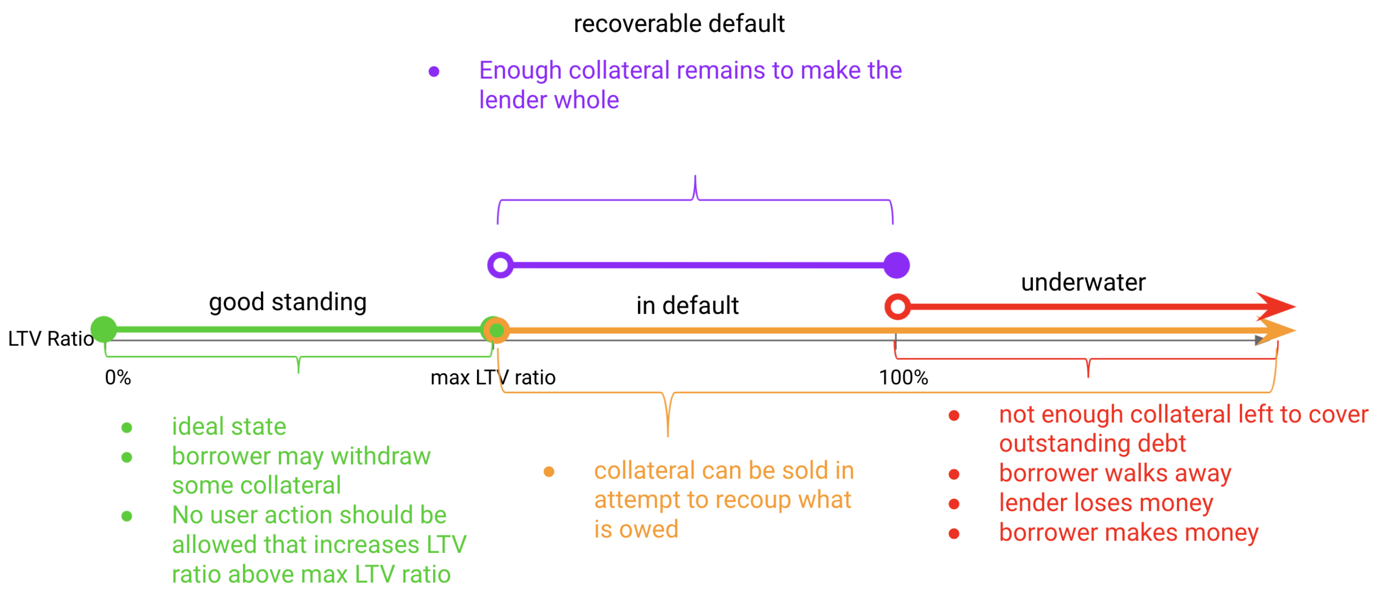
On-chain Decentralized Oracle

This type of oracle determines the price of assets using an on-chain source, but can be updated by anyone. There may be some sanity checking to ensure that prices don't fluctuate too wildly. DDEX (https://margin.ddex.io/) uses this type oracle for DAI, while bZx (https://bzx.network/) uses this type of oracle for all assets.

Attacks involving Oracles

Manipulating the Uniswap price

Imagine that you are building a decentralised lending platform using a price oracle. You have prices for the borrow token and the collateral. Your users have positions on your platform and have borrowed tokens against collateral.
Your platform can liquidate positions that are in default, usually it allows others to liquidate the positions. (MEV can be involved in this in a similar way to arbitrage)


recoverable default
• Enough collateral remains to make the lender whole
good standing
in default
underwater
LTV Ratio
0%
max LTV ratio
100%
• ideal state
• borrower may withdraw some collateral
• No user action should be allowed that increases LTV ratio above max LTV ratio
• collateral can be sold in attempt to recoup what is owed
• not enough collateral left to cover outstanding debt
• borrower walks away
• lender loses money
• borrower makes money

If an attacker can manipulate the price oracle, they can steal money. The process is :

- Adjust the borrow token price up (or the collateral price down), which puts the borrower in default,
- Liquidate the collateral for approximately $0.

This is usually the case for on-chain centralized oracles, ie. DEXs, whereby an attacker can use flashloans to manipulate the price of AMMs in order to change the spot price of a token just before the lender smart contract looks it up.

You would assume that Uniswap prices are always roughly correct whenever you want to buy or sell ETH as any deviations are quickly corrected by arbitrageurs.
However, as it turns out, the spot price on a decentralized exchange may be wildly incorrect during a transaction.

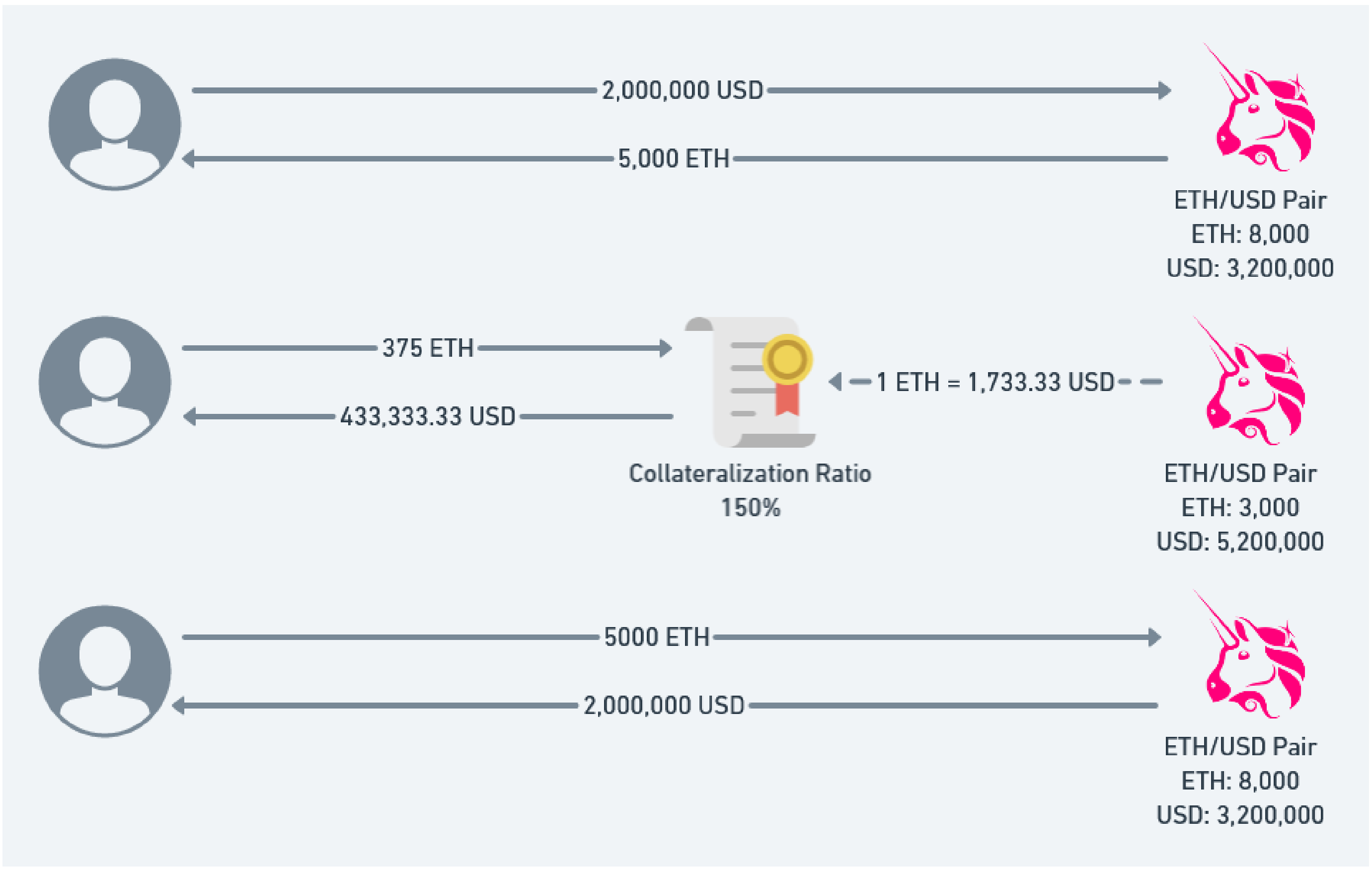
Consider how a Uniswap reserve functions.

The price is calculated based on the amount of assets held by the reserve, but the assets held by the reserve changes as users trade between say ETH and USD.
What if a malicious user performs a trade before and after taking a loan from your platform?

Let’s assume that a user wants to borrow USD using ETH as collateral :

Imagine the current price of ETH is 4000 USD,
the Loan to value ratio is 66.6% (i.e. a collateralization ratio of 150%).
If the user deposits 3750 ETH, they’ll have deposited 1,500,000 USD of collateral. They can borrow 1 USD for every 1.5 USD of collateral, so they’ll be able to borrow a maximum 1,000,000 USD from the system.

Before the user takes out a loan, they buy 5,000 ETH for 2,000,000 USD.
The Uniswap exchange now calculates the price to be 1 ETH = 1,733.33 USD.
Now, their 3750 ETH can act as collateral for up to 433,333.33 USD worth of assets, which they borrow.
Finally, they trade back the 5,000 ETH for their original 2,000,000 USD, which resets the price.
The net result is that your loan platform just allowed the user to borrow an additional 333,333.33 USD without putting up any collateral.



Arbitrage Risk

The latest reported price is, on average, at least several seconds behind the current “public knowledge” price, which can open up arbitrage opportunities.

In Chainlink, oracles only update every 20 minutes or if the price changes over 0.5%.

Preventative Techniques

It’s not always obvious that you’re using a price oracle and if you don’t follow the proper precautions, an attacker could exploit your protocol.

Don’t use an on-chain decentralized oracle without some sort of validation

Due to the nature of on-chain decentralized oracles, ensure that you’re validating the rate being returned, whether it’s by

- taking the order (thereby nullifying any gains which may have been realized),
- comparing the rate against known good rates (in the case of DAI),
- or comparing the rate in both directions.

Consider the implications of dependencies on third-party projects

An accurate rate for a DeFi project is an accurate rate for a DEX, but the opposite might not be true.

Shallow Markets

Consider whether the token is liquid enough to warrant integration with your platform.

Reality Check

The best way to know for sure the exchange rate between two assets is to simply swap the assets directly. This approach is great because there’s no take-backs and no what-ifs. However, it may not work for protocols such as lending platforms which are required to hold on to the original asset.

Speed Bumps

Manipulating price oracles is a time-sensitive operation.
If an attacker wants to minimize risk, they’ll want to do the two trades required to manipulate a price oracle in a single transaction so there’s no chance that an arbitrageur can jump in the middle.
As a protocol developer, if your system supports it, **it may be enough to simply implement a delay of as short as 1 block between a user entering and exiting your system.**

This would **remove the ability to perform deposits and withdrawals within a single transaction** and therefore, make flash-loan based attacks infeasible.
On the users’ side, this would mean that during depositing, their tokens will be transferred into a project in one transaction. The users would subsequently claim their share in another transaction, ideally inside a different block.
This would constitute a UX change and potentially incur a higher, yet still acceptable, gas cost for the depositors.

Of course, this might impact composability and miner collaboration with traders is on the rise. In the future, it may be possible for bad actors to perform price oracle manipulation across multiple transactions knowing that the miner they’ve partnered with will guarantee that no one can jump in the middle and take a bite out of their earnings.

Uniswap Time-Weighted Average Price (TWAP)

You must decide on the time interval to use, which can be tricky. A shorter time interval means that you will see price updates quicker, but also lowers the cost of attack to manipulate the oracle. A longer time interval makes it much harder to manipulate the average price, but also means you won't be able to react to volatility in the markets.

Curve Virtual Price

Curve pools provide a function to calculate the price of a single LP token in a flash loan resistant manner.

Maker Price Feed

Maker operates their own network of price feeds which exposes data to whitelisted contracts on-chain. Projects can apply for access to the price data through the Maker governance process.

M-of-N Reporters

This approach is used by many large projects today: Maker runs a set of price feeds (https://developer.makerdao.com/feeds/) operated by trusted entities, and Chainlink aggregates price data from Chainlink operators and exposes it on-chain.
Just keep in mind that if you choose to use one of these solutions, you've now delegated trust to a third party and your users will have to do the same. Requiring reporters to manually post updates on-chain also means that during times of high market volatility and chain congestion, price updates may not arrive on time.

Chainlink

See Architecture overview (https://docs.chain.link/docs/architecture-overview/)
Chainlink supports over 100 price feeds on Ethereum mainnet, mainly for pairs with ETH and pairs with USD. Developers can access this data for free simply by querying the smart contracts when needed.

When to use

Use Chainlink if you need pricing data for an asset that Maker or Uniswap doesn't support, or if the latency associated with a TWAP oracle is unacceptable to your project.
Currently supported chains

- Ethereum
- Binance Smart Chain
- Polygon (Matic)
- xDai

- Huobi Eco Chain
- Avalanche
- Fantom
- Arbitrum
- Harmony
- Optimism
- Moonriver
- Solana
- Terra