**Understanding the Cream Finance Hack**

**TLDR**

The Cream Finance attack consisted of a flash loan transaction leveraging a price oracle vulnerability in the Cream Finance protocol. The total amount stolen was **$130M** with most of the pools being completely drained in the Ethereum mainnet.

Cream started as a fork of Compound. However, the protocol has taken aggressive steps in providing a wider distribution of smaller cap tokens as well as derivate LP tokens such as yUSD. Some of these assets required a custom oracle proxy created by the Cream team, which introduced the attack vector.

**Protocols Involved In The Hack**

This hack was elegant and fatal. To understand it, we need to go deeper into some other protocols that were involved. It’s important to understand Cream, Yearn, and Curve as well as how they were interlinked in the hack. The hack was solely based on exploiting weaknesses in the Cream Finance protocol, and the other protocols were not manipulated or abused and worked just as expected.

**Cream Finance** is a decentralized protocol that provides lending and borrowing capabilities in a permissionless manner. Cream has a lending pool where you can provide liquidity with yUSD tokens, as well as use these yUSD tokens as collateral to borrow other assets.

**yUSD** is an LP token that represents a share of a Yearn Finance vault referred to as **Curve yCRV** in the Yearn UI’s listing of vaults. The vault is a stable coin fund that accrues value from interest, swap fees, and demand for the underlying tokens. The yUSD vault consists of four stable coins: yDai, yUSDT, yTUSD and yUSDC. The principal strategy of the vault is to allocate assets into the Curve**4Pool** pool of yDAI, yUSDC, yUSDT, and yTUSD (aka “4Pool’’ or simply “Y”, as it is called on the Curve UI).

**CREAM yUSD Pool**

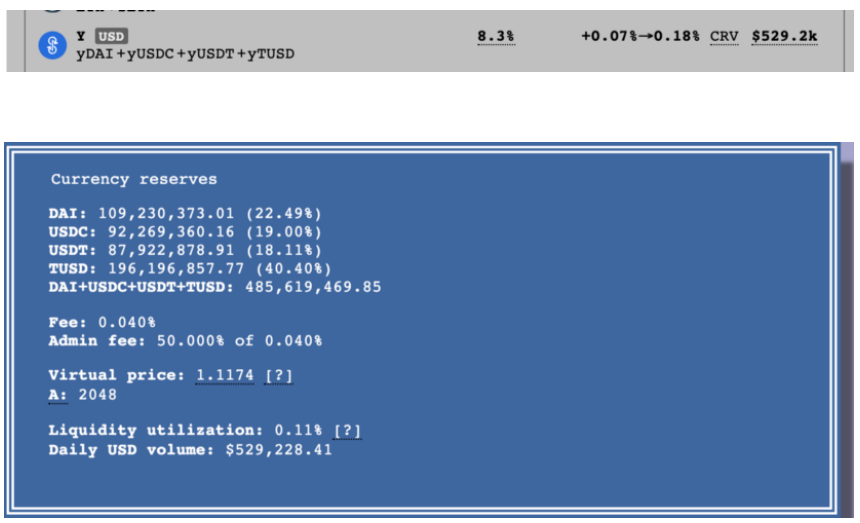
Cream Pool YUSD contract: <https://etherscan.io/address/0x4BAa77013ccD6705ab0522853cB0E9d453579Dd4#readProxyContract>

**yUSD**

Underlying in Cream Pool yUSD contract -> [Curve Y Pool yVault (yUSD)](https://etherscan.io/token/0x4b5bfd52124784745c1071dcb244c6688d2533d3): <https://etherscan.io/address/0x4b5bfd52124784745c1071dcb244c6688d2533d3>

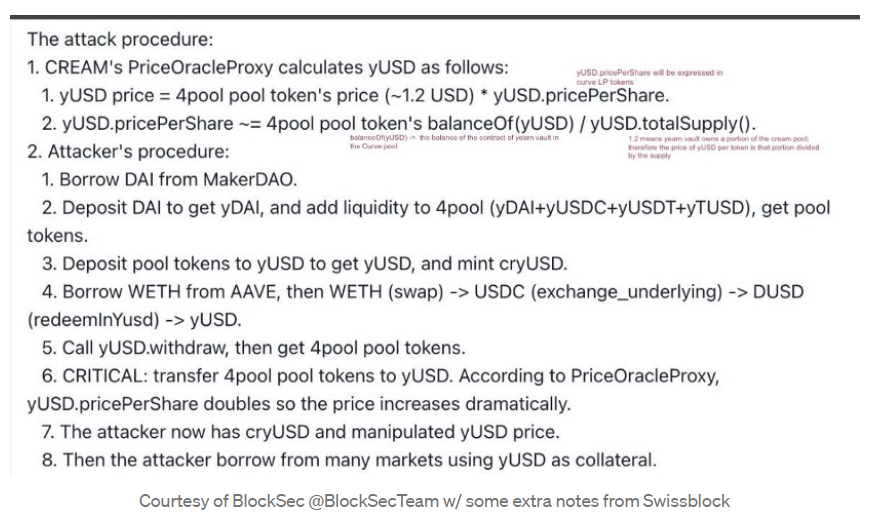
**CURVE 4Pool**

<https://etherscan.io/address/0xdf5e0e81dff6faf3a7e52ba697820c5e32d806a8>



**Anatomy of the hack**

The hacker used a flash loan attack that took advantage of a badly implemented oracle price proxy. The oracle proxy calculated the pricePerShare using on-chain calls in 4Pool and yUSD contracts. Below is a high level of all the steps in the hack prepared by the experts at BlockSec within hours of the hack.



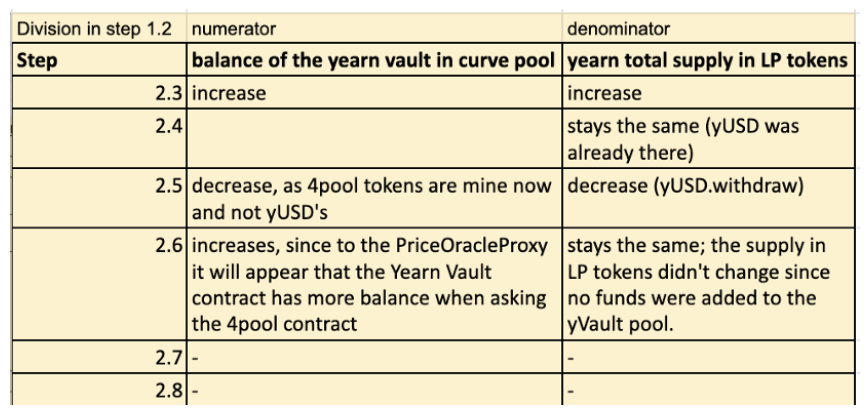
Courtesy of BlockSec @BlockSecTeam w/ some extra notes from Swissblock

In section 1, you can see how the price oracle computes the yUSD price. In line 1.2, you can see that yUSD.pricePerShare is calculated by taking the percentage that yUSD (Yearn Vault) owns of 4 Pool and dividing it by the total number of LP tokens of yUSD. In other words, this is giving what percentage of 4 Pool does each yVault LP token own. If you multiply this by the price of 4 Pool tokens, you will get the price of yUSD in USD terms.

The attacker used an attack vector in the price definition in step 1.2 inside the oracle proxy. The attacker sent a token to the contract address directly instead of passing through the defined contract calls that keep track of the accounting properly. By sending the 4Pool token directly to the yUSD contract, he was able to:

* increase the numerator（分子） in step 1.2 since the ERC20 4Pool contract will return that yUSD owns more LP tokens
* maintain the denominator（分母） in step 1.2 the same since no new yUSD token was minted since the attacker didn’t use the established smart contract method to deposit into yUSD

This allowed the attacker to manipulate the price, therefore using yUSD to borrow from many markets. Below is a table that goes through each step explaining how it affects the numerator and denominator of step 1.2.

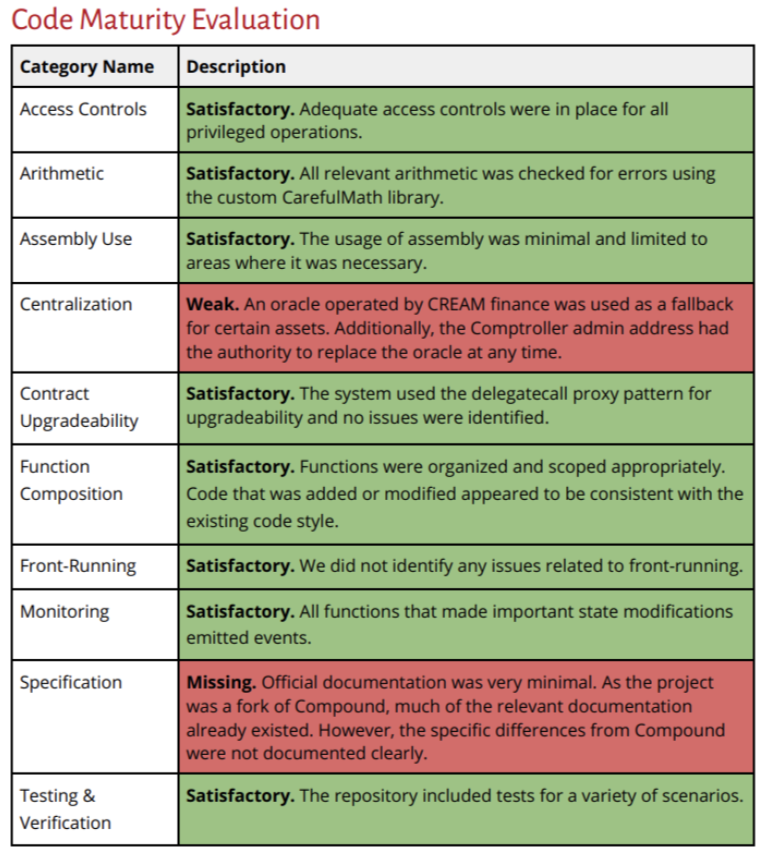


**Could this hack be prevented?**

Price oracle hacks are nothing new, and there’s a good amount of documentation around this topic. It’s worth noting that the Price Oracle proxy vulnerability was included in the Cream Finance audit as seen below in the table. Leveraging a proper oracle to get the price will definitely protect against this attack. It would be ideal if the prices are provided through TWAP to avoid market manipulation in a similar fashion as **mark price** add market manipulation protection in perpetual swap instruments.

Additionally, the Cream team could have avoided using tokens in the protocol that can easily be burnt or minted and that are interlinked with other protocols since this increases the complexity heavily. With increasing complexity, it’s harder to be able to foresee all edge cases.

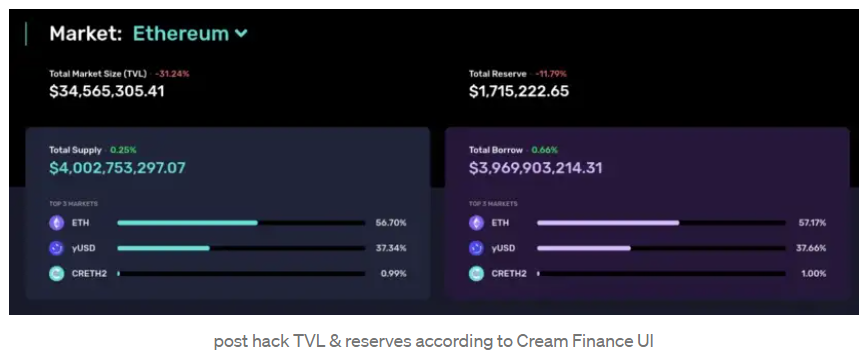
Another good measure would be to add a fund limit per pool in USD that a specific party can do within a certain time frame. This would act as a fail-safe, which if triggered would allow the protocol and 3rd parties to have some time to react in case something goes wrong. There’s no reason to allow one party to take away the entire liquidity of all pools within one transaction.



CREAM Finance audit: <https://files.gitbook.com/v0/b/gitbook-x-prod.appspot.com/o/spaces%2F-MKJWcUfnKdJ1BjWKoMM-3555916521%2Fuploads%2Fgit-blob-1dd4323e898465d6750e678f0000f254227453b9%2FCREAMSummary.pdf?alt=media>

**Open questions**

* How will CREAM Finance address the issue? If they choose to solve the issue, where will they obtain the funds from?
* Are there any funds within the contracts that are redeemable and can be liquidated to pay the affected parties?（合约中是否有任何资金是可赎回的，可以清算支付给受影响的各方?）



post hack TVL & reserves according to Cream Finance UI

**Important Links**

You can see the infamous transaction below: <https://etherscan.io/tx/0x0fe2542079644e107cbf13690eb9c2c65963ccb79089ff96bfaf8dced2331c92>

Exploiter Externally Owned Address: <https://etherscan.io/address/0x24354d31bc9d90f62fe5f2454709c32049cf866b>

The hacker’s flash loan interacted with the contract on the link below. The contract was deployed specifically for the hack. The contract was deployed at 13:22 UTC and the attack started 20 min afterward. <https://etherscan.io/address/0x961d2b694d9097f35cfffa363ef98823928a330d>

**Acknowledgment**

This post was a collaborative effort by Daniel Kohlsdorf, Oscar Berger, and myself. We would also like to thank BlockSec for sharing their insights early on.