1 Curve Vulnerability Report

In the wee hours of the morning on September 19th, I discovered an exploit against Curve contracts that allows an attacker to withdraw a large fraction of token balances when a contract's amplification coefficient, A, is updated. Swerve, which uses the Curve contracts, was due for an update to A in a matter of hours. The potential losses for Swerve were huge, standing at 36.9% of contract balances or approximately 92 million USD if a well-optimized attack were carried out. I decided that nothing could be done to remedy the situation on such short notice. Thankfully, the Swerve update passed without incident. Later that afternoon, I notified Curve. Within a few hours, they confirmed the existence of the vulnerability and we began working together to investigate solutions.

This attack is possible for both upward and downward adjustments to A. However, since potential losses are orders of magnitude greater for downward adjustments, I focus discussion on this class of attacks. The severity of these attacks is proportional to the magnitude of changes in A. As it turns out, the maximum loss as a share of token balances is bounded by $1-\sqrt[n+1]{\frac{A_{new}}{A_{old}}}$ where A_{old} is the initial parameter value, A_{new} is the updated parameter value, and n is the number of token types in the Curve contract. As an example, an update of yCurve occuring earlier in the same week changed A from $A_{old} = 2000$ to $A_{new} = 1000$. During this update, an attacker could have used the vulnerability to withdraw up to $1-\sqrt[4+1]{\frac{1000}{2000}} = 12.9\%$ of contract balances or approximately 77 million USD.

This attack is only possible to conduct during a scheduled update to the parameter A. Curve contracts are not vulnerable under normal operation. Consequently there is no need for urgent action to protect user funds. However, it is crucial that this vulnerability is patched before additional changes to A occur. Large downward adjustments to A are particularly dangerous. The Curve team is implementing improvements to procedures for updating A. These improvements should allow Curve contracts to update A in a safe manner going forward.

2 Means and Token Bonding Curve Shape

To understand the attack, it is necessary to have some knowledge of token bonding curves. I will explain some concepts, so that the reader can develop a conceptual understanding. I take a slightly different approach to the topic than what you might encounter elsewhere, focusing on the relation of token bonding curves to the mean of a set of variables.

In mathematics, a mean is "a quantity that has a value intermediate between those of the extreme members of some set." Thus, if x_1 is the smallest number in a set of n numbers and x_n the largest, a mean of this set would take on a value intermediate between x_1 and x_n. The two most common types of means are the arithmetic mean and the geometric mean. The arithmetic mean, call it $\bar{x}_{AM}$, is just the simple average of n numbers, i.e. $\bar{x}_{AM} = \frac{1}{n}\left(x_1 + x_2 + \cdots x_n\right)$. The geometric mean, call it $\bar{x}_{GM}$ is the nth root of the product of n numbers, i.e. $\bar{x}_{GM} = \left(x_1 x_2 \cdots x_n\right)^{\frac{1}{n}}$.

Means play a central role in token bonding curves. AMM contracts allow users to trade any combination of tokens, such that a mean of the AMM contract's token balances remains constant before and after a trade takes place. Different types of means are employed in different AMM designs. For Uniswap, an unweighted geometric mean is used. For Balancer, a weighted geometric mean. For mStable, an unweighted arithmetic mean.

Curve uses a weighted mean of the arithmetic mean and the geometric mean. I call this the Curve mean. The weighting of the Curve mean is determined by the so-called amplification parameter, A. As A increases towards infinity, Curve's mean converges to the arithmetic mean used by mStable. Conversely, if A is set to 0, Curve's mean becomes identical to the geometric mean used by Balancer and Uniswap. For intermediate values of A, Curve's token bonding curve will lie somewhere in the middle of these two extremes.

The three types of token bonding curves used by Uniswap, Curve, and mStable are illustrated in Figure 1.

Figure 1

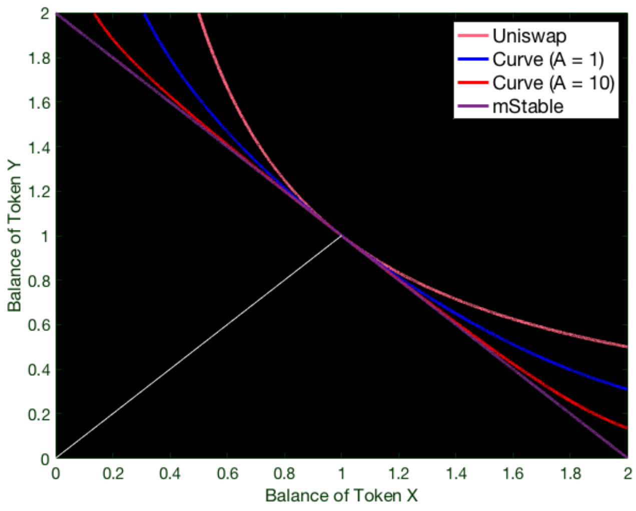


Figure 1 shows four token bonding curves that arise from holding the mean of token balances constant. At one extreme, Uniswap holds the geometric mean constant. This yields a very steep curvature. At the other extreme, mStable holds the arithmetic mean constant. This yields a straight line. The figure also shows examples for Curve. At the Curve parameter value, $A = 1$, a greater weight is placed on the geometric mean and the curve resembles Uniswap. At the Curve parameter value, $A = 10$, a greater weight is placed on the arithmetic mean and the curve more closely resembles mStable.

3 Means and the Value of AMM Contract Holdings

Referring to Figure 1, we can see that all four curves intersect at a point along a 45-degree line from the origin of the graph. We can use the distance of this intersection point from the origin to quickly measure the value of an AMM contract's token portfolio. For example, if the distance of this intersection point from the origin increases by 20 percent, then, assuming no impermanent loss, the value of the AMM contract's holdings will have increased by 20 percent as well. This comparison works for all four of our bonding curve types. This propoerty is especially helpful when we consider Curve, which has the unique characteristic that the shape of its bonding curve changes when A is updated. For Curve, we can use the distance from the origin to measure the value of the contract's portfolio before and after a parameter update. Obviously, if this distance decreases markerdly after A is updated that would be a major cause for concern.

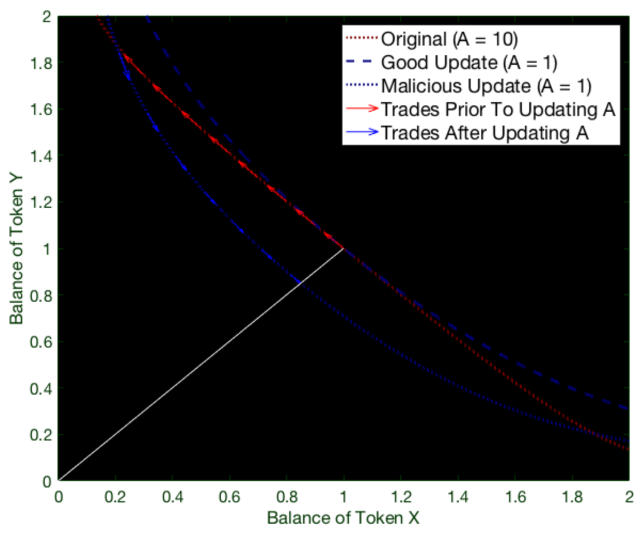
4 Break-Even Updates to A

Referring again to Figure 1, suppose that a Curve contract's token balances lie exactly at the intersection point along the 45-degree line. This occurs when all Curve tokens trade at a one-to-one price ratio. When A is updated from this starting point, there is no risk of monetary loss. For example, suppose that Curve moved the parameter setting $A_{old} = 10$ to $A_{new} = 1$, starting from this point. This update would not change the distance of the bonding curve from the origin. As a result, the parameter change would be completely innocuous and would not expose Curve LPs to risk of financial loss. Intuitively, if initial balances are not exactly at the intersection point, but close to this intersection point, the risk of loss remains small.

5 Loss-Making Updates to A

Now let's look at Figure 2. The figure illustrates how an attacker could potentially manipulate the initial conditions when A is updated to realize an enormous profit. For ease of illustration, I have depicted a change from $A_{old} = 10$ to $A_{new} = 1$, rather than Swerve's update from $A_{old} = 1000$ to $A_{new} = 100$. However, as it turns out, the severity of the vulnerability only depends on the ratio $\frac{A_{old}}{A_{new}}$ and thus the figure is an accurate depiction of Swerve's situation. Also, the attack illustrated in the figure only captures 15% of the contract's token inventory. A fully optimized attack would trade more extreme amounts and could capture up to 36.9% of token inventory.

Figure 2



Suppose that Curve contract balances are initially at the intersection point with the 45 degree line and that the initial parameter value is $A_{old} = 10$. Now suppose that an attacker sandwiches a parameter update between two malicious trades. During the first malicious trade, the attacker sells an extreme quantity of tokens to induce a severe inventory imbalance. Next, the attacker triggers an update to A from $A_{old} = 10$ and $A_{new} = 1$. Finally, the attacker buys back the tokens he sold at a much lower price. This action returns the contract to a perfectly balanced state along the 45 degree line. As depitcted here, the attack would deplete 15% of the AMM's token inventory.

6 Broken Smart Contract Logic

Is this attack actually possible? Surprisingly, the answer is yes. Curve contracts require changes in A to be scheduled days in advance and agreed upon via a decentralized on-chain governance process. However, once a change in A has been approved via governance and an activation deadline has passed, the contracts allow any caller to trigger the update. As a result, an attacker is free to flash loan a gargantuan quantity of stablecoin from Uniswap, sell it to Curve to trigger an extreme imbalance, trigger an update to A, and then buy the stablecoin back from Curve at an enormous profit. Fully optimized attacks are somewhat more involved, but there is no need to get into the details here. The simple attack I described above is enough to capture most of the potential profit.

7 Fixing the Smart Contract Logic for Changes in A

Currently, there are two production versions of Curve contracts. For unpatched older edition contracts, the attack described in the previous section works exactly as described. For newer contracts, there is still a potential vulnerability, albeit much less severe. I'll describe recommended changes for older contracts first.

7.1 Fixing Older Curve Contracts

In older Curve contracts, changes in A occur in one big discrete step. Moreover, the contract logic allows an attacker to perform trades at different values of A within a single transaction. In particular, the attacker can use his initial trades to force an extreme inventory imbalance, then trigger the change in A, then perform more trades at the updated value of A. This allows the attacker to perform an entire attack involving hundreds of millions in exchanged volume with zero risk, aside from the gas cost, bad mojo and likely prison sentence.

To fix this, I recommend that older contracts be updated so that only a trusted multi-signature account could activate an update to A. Furthermore, activation of A should require a token balance check to confirm that token balances have not changed significantly from the timepoint when the parameter update transaction was broadcast. This balance check prevents a possible attack by a rogue miner. In particular, a rogue miner could reorder transactions such that he performs a large trade before A is updated, and then another large trade after A is updated.

The balance check prevents activation of a change to A when the contract is an unexpectedly unbalanced state. This is sufficient to protect Curve LPs from the exploit.

7.2 Fixing Newer Curve Contracts

In newer Curve contracts, changes in A occur gradually in a series of small discrete steps before each trade is performed. My understanding is that only a single step adjustment is permitted per block. Furthermore, the contracts require a scheduled step adjustment to occur before any trades are performed. This is sufficient to protect against a normal attacker, but is not necessarily secure against a rogue miner. In particular, a rogue miner could mint two blocks in a row and insert malicious trades in both blocks. This would allow the miner to perform an initial trade in the first block at a high value of A and a final trade in the second block at a lower value of A. To make things worse, the rogue miner has an extended window over which to attempt these attacks. As long as A is still in the process of updating, the rogue miner can continue to make attempts at mining a two block sequence.

To protect these newer contracts, I recommend reducing the step size in A to no more than 0.1% per block. Why does a small step size help? This brings in a factor that I haven't introduced yet. The Curve contract charges a fee. Due to this feee. any trade will cause the token bonding curve to move very slightly away from the origin. This also applies to the attacker's mammoth trades and this makes attacks slightly less profitable. If a change in A is sufficiently small, the gains from a fully optimized attack are more than offset by fees the attacker pays to the contract. As a result, it is no longer possible for an attacker to profit by sandwiching a change in A between two trades.[1]

8 Lessons for Security Audits and Smart Contract Design

Design of this attack required a deep understanding of token bonding curves. It may not be realistic for smart contract auditors to discover a vulnerability that makes use of such highly specialized knowledge. The Curve contracts have been audited multiple times. As I write this, the Swerve contracts have just passed yet another audit. In my view, it would be really useful to have a general auditing procedure that could detect vulnerabilities of this type via brute force probing rather than theory. To detect this type of vulnerabiity, I would recommend token bonding curve audits incorporate simulation of arbitrary two step trading procedures. In these procedures, the auditor would run a random trade against the contract, trigger a smart contract action, then run another random trade. Here, the smart contract action would be activating an update to A. For this vulnerability, this fuzz testing procedure would uncover scenarios where the contract suffers catostrophic loss. Auditors could then investigate further to understand underlying causes.

For smart contract designers, it is helpful to appreciate the limitations of audits. When contracts allow for a complex series of interactions to be performed all at once, comprehensive fuzz testing becomes infeasible. The problem is that there are simply too many possible combinations of user interactions to probe every possibility. Thus, it can be helpful to limit the number and variety of actions a user can take within a short time span. The idea here is to avoid creating a smart contract that is so complex that it becomes infeasible to audit via a brute force approach.

I would like to thank the Curve team for payment of a very generous bug bounty as reward for reporting this vulnerability. Also, a big shout out to my 0x buddy Greg Hysen for digging through Curve's code to help me understand the smart contract logic for updating A.

[1] A caveat here is that the miner can always recover a profit if they can mine a sufficient number of blocks in a row. However, a core security assumption of Ethereum is that it is not possible for a miner to do this. While there are ways of addressing this concern, in my opinion it is not really necessary to go that far.