How to Exploit a DeFi Project

Xinyuan Sun[1], Shaokai Lin[2], Vilhelm Sjöberg[1], and Jay Jie[1]

[1] CertiK, New York NY 10018, USA
{sxysun, vilhelm.sjoberg, jay.jie}@certik.io
[2] University of California, Berkeley, Berkeley CA 94720, USA
shaokai@berkeley.edu

Abstract. The growing adoption of decentralized finance poses new security risks, as designing increasingly complex financial models is error-prone. We have witnessed numerous DeFi projects hacked (for tens of millions of dollars) because of unsound liquidation conditions, asset pricing, or position management, etc. To address these issues, we present a systematic way of finding vulnerabilities in DeFi projects based on automatically extracting financial models from smart contracts and reasoning about them symbolically using either a model checker or an interactive theorem prover. Specifically, we (i) formalized the concept of soundness in the financial model of a DeFi project which captures an interesting class of exploits (flash-loan attacks), and (ii) built a domain-specific language to automatically extract models from smart contracts and search possible exploits or prove their soundness. To demonstrate the capability of our approach, we model variants of most DeFi projects with a TVL (total value locked) larger than 20M USD (totaling about 8B USD TVL) and check their soundness. The result showed that we can automatically find both previous exploits and potential new flaws in DeFi.

Keywords: decentralized finance · model checking · formal verification

1 Overview

In the past year, there have been so many decentralized finance project breaches to the point that million-dollar hacks don't make the news anymore. We reviewed 31 exploit incidents happened to DeFi in 2020 and found that they in total caused a capital loss of 315.95M USD (Appendix A, Table 1), with the highest attack resulting 50M USD of lost funds. The inability to find and prevent those hacks resulted from the complexity of the DeFi "money Lego"—we not only have to consider contract implementation correctness but also, more importantly, the soundness of interleaved financial models. To quote an experienced auditor: "If you go into a DEX audit without a full understanding of how options or derivatives work, you are probably going about it all wrong."

Challenges. To systematically study DeFi exploits, we face two challenges:

(i) We lack a classification of past financial security-related exploits on DeFi projects. Earlier attack vectors like re-entrancy or access control only concern

simple semantics, and other functional correctness related hacks are well defined with respect to the specifications. However, there has been little work in the formalization of a DeFi exploit. People often argue if some exploit is a hack or an arbitrage, especially in communities where decentralization is valued (smart contract code is deemed as law for every trader to follow).

(ii) From the 31 DeFi attacks we studied, we found that 20 of them concern high-level financial model details (e.g., token bonding curve economics, AMM's self-balancing mechanism, etc,.) hidden in the footnotes of the project whitepaper or, worse, tens of thousands of lines of smart contract codes. The complex nature of DeFi protocols makes it infeasible for us to reason directly about the implementation. The truth is, current smart contract security solutions [3, 5, 4, 10] like pattern-based vulnerability matching or SMT-based verification, though well-studied, are not tailored to formally reason about assets and derivatives.

Attack Patterns. To address challenge (i), we studied 31 past "attacks" [3] on DeFi projects and found that they fall into three categories: unsound financial models, arbitrages, or insecure implementations. The arbitrage case is unavoidable as arbitrage is necessary for decentralized exchanges to align their prices with the real market price. The best thing we can do is to write a community arbitrage bot on-chain and distribute revenue to liquidity providers (who are usually losing money in arbitrage actions). If we consider priority gas auctions where miners extract the most value with a consensus-level advantage, then there is no way to stop this kind of "attacks" unless we devise a fairer auction mechanism [1]. Furthermore, arbitrage has the smallest impact out of the three categories in Table.1, with an average of 70k USD weekly profit and about 4M USD gain over one year [12], consisting less than 2% of the total capital loss. Insecure implementations of smart contracts (e.g., re-entrancy caused by inclusion of ERC777 tokens [6]) are already well-addressed by previous work [3, 5, 4, 10]. While these attacks can be caught by existing tools, they still happened because some projects didn't go through a proper audit, so these kind of attacks are more of an engineering problem.

Scope. In this work we focus on unsound financial models, that is, we limit our scope to exploits where the attacker can mint, hold or burn assets (tokens, or positions representing a certain amount of token) at the expense of everyday traders and liquidity provider's interests, thus damaging the entire ecosystem's health. We emphasize on damaging the entire ecosystem's health because in cases like arbitrage or high slippage, *reasonably-knowledgeable* users should take responsibility for their own bad trades, while in a financial model flaw situation, the attacker can profit off users even if no one makes mistakes (even though some people might argue that by using flash-loans to manipulate the spot market for greater income in the derivatives constitutes an hack but is a necessary evil to cool the market down by deleveraging). From Fig.1, we can see that financial models have been consistently attacked more than insecure implementations all year, but exploits related to them only started to contribute a dominant amount

[3] see appendix A

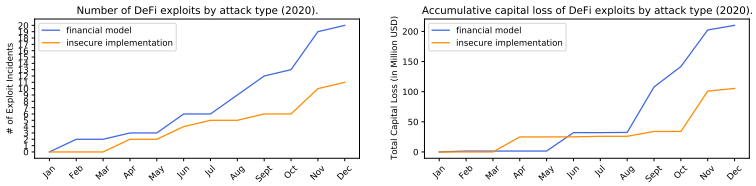

Fig. 1: DeFi exploit trends in 2020.

of capital losses since August, reaching a 66.48% at the end of year. This demonstrates that financial model exploits has been increasingly more damaging than insecure implementation exploits due to the mass adoption of DeFi since summer 2020. By *reasonably-knowledgeable*, we mean having knowledge of common sense finance. For example, users know that they can make a profit by arbitraging different spot prices at different exchanges, or that when using a self-balancing automated market maker, they should trade with the optimal price calculated from the reserves. But a reasonably-knowledgeable user would not be expected to know things that require detailed analysis of DeFi, e.g., that they can profit from manipulation of token inflation models using some exchanges' trading mechanism [11]. Additionally, we say a user is *maximally-rational* if they are reasonably-knowledgeable and rational, i.e., they trade in a way that maximizes their profits using their reasonable amount of knowledge. For example, if there is a spot arbitrage opportunity created by their own trades, they will arbitrage within the same transaction as their initial trade to front-run everybody else. Another example would be taking liquidation reward of an under-collaterized debt position on lending platforms (e.g., Compound).

Soundness. Formally, we model DeFi exploits in a model checking fashion. We say that the application starts with an initial state s_0 and transition functions $F = \{f_1, f_2 \dots f_n\}$, each abstracted from an actual state changing function in the project's smart contracts (e.g., `borrow` from AAVE [9]). If a state s_j is derivable from a state s_i by at most m arbitrary transactions which each can involve up to n contract calls, we write $s_i \Rightarrow_{m,n} s_j$. Note that each of the n contract calls within one transaction have a single caller, sequential execution semantics and deterministic ordering, while the m transactions have a non-deterministic ordering (decided by miners). If in those m transactions and n contract calls all users are *maximally-rational*, we write the transition relation as $\Rightarrow_{\mathcal{R}\ m,n}$. We define a state s_i as *consistent* if the state is the result of a series of interactions between multiple *maximally-rational* traders. Intuitively, *consistent* states can be understood as a state where spot and futures price of every trading pair is uniform across all DEXs and CEXs within a reasonable bound (e.g., less than the transaction fees). We say there is a hack of the financial model in a DeFi project if there exists such s_i, a *consistent* state derivable from the initial state s_0 by relation $\Rightarrow_{*,*}$, from where the attacker can take an arbitrary amount of interleaved (across different transactions) contract calls and reach a state s_j where the asset of the attacker at s_j is strictly greater than his/her asset at s_i

while everybody else acts maximally-rationally. Therefore, we define the ideal-soundness of a DeFi project as the absence of a hack: $\neg\exists s_i, s_j, h, s.t. s_0 \Rightarrow_{*,*} s_i \rightarrow \mathit{consistent}(s_i) \rightarrow s_i \Rightarrow_{\mathcal{R}\ *,*} s_j \rightarrow \Xi(h, s_i) < \Xi(h, s_j)$, where $\Xi(h, s_i)$ is the total value of address h's capital denoted in ETH, if liquidated, at state s_i.

k-soundness. The ideal-soundness property is infeasible to express or check (for limitations of current automated reasoning techniques) in reality. For example, what kind of knowledge is reasonable for a DeFi user to be assumed to know is disputed, and the maximally-rational constraint on relation $\Rightarrow_{\mathcal{R}\ m,n}$ is hard to express. Therefore, we make three refinements to the previous definition: (1). we limit transaction transition from the sound state s_i to one, i.e., $\Rightarrow_{\mathcal{R}\ 1,*}$, so that no other users can interact with the contract except for the attacker. This means everyone is trivially maximally-rational as they do not operate. (Equivalently, you can imagine this as only considering "flash loan"-style attacks, in the sense that every operation within a flash-loan is deterministic and packed into one transaction). (2) we further limit the number of possible contract calls within that one transaction on s_i to a finite number k, i.e., $\Rightarrow_{\mathcal{R}\ 1,k}$. This enables us to use bounded model checking to traverse the state space symbolically so that we can prove there are no possible exploits within k steps. (3) we drop the requirement that state s_i has to be derivable from the initial state and we arbitrarily start from a state s_i with concrete values generated under the constraint *consistent*. Altogether, we define this changed property as k-soundness. We say a DeFi project is k-sound with respect to s_i if and only if given a concrete contract state s_i that is *consistent*, there exists no address that can statically (symbolically) find a way to increase its total asset within k interactions with the contract: $\neg\exists s_j, h, s.t. \mathit{consistent}(s_i) \rightarrow s_i \Rightarrow_{\mathcal{R}\ 1,k} s_j \rightarrow \Xi(h, s_i) < \Xi(h, s_j)$.

Model Extraction. We address challenge (ii) by providing a domain specific language to model assets in DeFi projects. We built the DSL using DeepSEA [8], which can automatically extract the financial model we annotated into (1) the UCLID5 [7] model checker and check for k-soundness, or (2) compile it to a Coq specification so that we can prove stronger correctness like ideal-soundness by manually. To make future usage easier, we provide a decentralized finance model library written in our DSL that covers all popular services (liquidity farming, interest-bearing tokens, automated market makers, etc,.). Moreover, since DeepSEA has a certified compiler backend, we can emit executable EVM/eWasm code for those asset-related functions after verifying their correctness. In the future, we could also extend the model checker into checking liveness properties like eventual governance to prevent failures like yam.finance [2].

Case Study. We used our language to model a past attack [4] of the bZx project which happened because of an incorrect liquidation check (the short position hacker opened did not liquidate despite under-collateralization). Specifically, we wrote simplified versions of the bZx and Uniswap contracts in DeepSEA and our DSL automatically extracted them into UCLID5 models, we checked against k-

[4] the bZx attack on Feb.15th, 2020, shown in appendix A

soundness ($k = 5$ in our setup) of the contract and it successfully gave us the exploit pattern.

Acknowledgement

We would like to acknowledge the contribution of many colleagues on various related projects at CertiK, especially Ronghui Gu, Dan She, Jialiang Chang, Junhong Chen and Zhaozhong Ni.

References

1. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. arXiv preprint arXiv:1904.05234 (2019)
2. George Georgiev: Yam finance crashes over 90%, founder admits his failure (2020), https://cryptopotato.com/yam-finance-crashes-over-90-founder-admits-his-failure/
3. Hajdu, Á., Jovanović, D.: solc-verify: A modular verifier for solidity smart contracts. arXiv preprint arXiv:1907.04262 (2019)
4. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 254–269. ACM (2016)
5. Permenev, A., Dimitrov, D., Tsankov, P., Drachsler-Cohen, D., Vechev, M.: Verx: Safety verification of smart contracts. Security and Privacy **2020** (2019)
6. Riley, D.: $25m in cryptocurrency stolen in hack of lendf.me and uniswap (2020), https://siliconangle.com/2020/04/19/25m-cryptocurrency-stolen-hack-lendf-uniswap/
7. Seshia, S.A., Subramanyan, P.: Uclid5: Integrating modeling, verification, synthesis and learning. In: 2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE). pp. 1–10 (Oct 2018). https://doi.org/10.1109/MEMCOD.2018.8556946
8. Sjöberg, V., Sang, Y., Weng, S.c., Shao, Z.: Deepsea: a language for certified system software. Proceedings of the ACM on Programming Languages **3**(OOPSLA), 1–27 (2019)
9. Team, A.: Aave developers doc (2020), https://docs.aave.com/developers/
10. Wang, Y., Lahiri, S., Chen, S., Pan, R., Dillig, I., Born, C., Naseer, I., Ferles, K.: Formal verification of workflow policies for smart contracts in azure blockchain. In: Verified Software: Theories, Tools and Experiments. Springer (September 2019)
11. Williams, M.: Rising defi protocol balancer loses $500,000 to hacker in pool exploit (updated) (2020), https://www.bitcoininsider.org/article/89413/rising-defi-protocol-balancer-loses-500000-hacker-pool-exploit-updated
12. Zhou, L., Qin, K., Cully, A., Livshits, B., Gervais, A.: On the just-in-time discovery of profit-generating transactions in defi protocols. arXiv preprint arXiv:2103.02228 (2021)

A Past Exploits on DeFi Projects

Table 1: Exploits on DeFi projects happened in 2020

Platform	Attack Type[1]	Reason	Loss[2]	Date
bZx	financial model	liquidation check	830k	2020.2.15
bZx	financial model	oracle synchronization	642k	2020.2.18
Uniswap	insecure implementation	ERC777 re-entrancy	220k	2020.4.18
Lendf.Me	insecure implementation	ERC777 re-entrancy	24.7M	2020.4.22
Hegic	financial model	frozen funds	29k	2020.4.23
Bancor	insecure implementation	access control	131.9k	2020.6.19
Atomic Loans	insecure implementation	front-running	N/A	2020.6.24
Balancer	financial model	deflation model	500k	2020.6.29
Balancer	financial model	reserve synchronization	2.7k	2020.6.29
Synthetix	financial model	oracle synchronization	30M	2019.6.30
VETH	insecure implementation	access control	900k	2020.7.1
Uniswap BZRX	arbitrage	bonding curve	531k	2020.7.14
Opyn	financial model	option exercise	371k	2020.8.4
Uniswap & Curve	arbitrage	spot arbitrage	43k	2020.8.10
NUGS	financial model	token inflation	N/A	2020.8.12
Yam.finance	financial model	token inflation	60M	2020.8.12
SYFI	financial model	exchange rate	250k	2020.9.10
bZx	insecure implementation	double dipping	8M	2020.9.14
Soda	financial model	loan liquidation	160k	2020.9.20
Eminence	financial model	burning mechanism	15M	2020.9.29
Harvest.finance	financial model	interest model	33.8M	2020.10.26
Akropolis	insecure implementation	re-entrancy	20M	2020.11.12
Value DeFI	financial model	oracle manipulation	7.4M	2020.11.14
Cheese Bank	financial model	oracle manipulation	3.3M	2020.11.16
OUSD	insecure implementation	re-entrancy	7M	2020.11.17
88mph	financial model	token burning	N/A	2020.11.19
Pickle.finance	insecure implementation	whitelist	20M	2020.11.22
Compound	financial model	oracle manipulation	N/A	2020.11.26
Pickle.finance	insecure implementation	whitelist	20M	2020.11.22
Sushiswap	financial model	token conversion	15k	2020.11.30
Saffron Finance	financial model	frozen funds	50M	2020.11.30
Warp finance	financial model	oracle manipulation	7.7M	2020.12.18
Cover Protocol	insecure implementation	outdated cache	4.4M	2020.12.28
total	N/A	N/A	315.95M	N/A

[1] We do not include centralization issues (rug pulls)
[2] Capital loss calculated using corresponding cryptocurrencies' prices at time of exploit