DeFi Auditing: Mechanisms, Effectiveness, and User Perceptions

Ding Feng[1], Rupert Hitsch[2], Kaihua Qin[3], Arthur Gervais[4], Roger Wattenhofer[2], Yaxing Yao[5], and Ye Wang[1]

[1] University of Macau, Macau, China
mc25944@um.edu.mo, wangye@um.edu.mo
[2] ETH Zurich, Zurich, Switzerland
hitsch.rupert@gmail.com, wattenhofer@ethz.ch
[3] Imperial College London, London, United Kingdom
kaihua.qin@imperial.ac.uk
[4] University College London, London, United Kingdom
a.gervais@ucl.ac.uk
[5] University of Maryland, Baltimore County, Baltimore, United States
yaxingyao@umbc.edu

Abstract. Decentralized Finance (DeFi), a blockchain-based financial ecosystem, suffers from smart contract vulnerabilities that led to a loss exceeding 3.24 billion USD by April 2022 [67]. To address this, blockchain firms audit DeFi applications, a process known as DeFi auditing. Our research aims to comprehend the mechanism and efficacy of DeFi auditing. We discovered its ability to detect vulnerabilities in smart contract logic and interactivity with other DeFi entities, but also noted its limitations in communication, transparency, remedial action implementation, and in preventing certain DeFi attacks. Moreover, our interview study delved into user perceptions of DeFi auditing, unmasking gaps in awareness, usage, and trust, and offering insights to address these issues.

Keywords: Decentralized finance · auditing · blockchain.

1 Introduction

Decentralized Finance (DeFi), an ecosystem built on blockchain's smart contracts, has experienced significant growth, with a Total Value Locked (TVL) surpassing 110 billion USD in 2022. Despite its expansion, DeFi is frequently targeted by cyber-attacks due to the lack of legal and industry standards and its inherent decentralization. The immutable nature of deployed DeFi project code, coupled with potential user vulnerabilities, worsens the scenario. An escalating number of auditing firms are striving to meet the demand for dependable safety assessments.

Nonetheless, DeFi projects continue to be compromised despite these audits, questioning these firms' reliability and credibility. Additionally, a universal standard for DeFi auditing is currently lacking. Consequently, our study explores the following crucial questions:

RQ1 How does DeFi auditing identify DeFi application security vulnerabilities?
RQ2 Which vulnerabilities persist despite DeFi auditing?
RQ3 How do DeFi users perceive the role of DeFi auditing in DeFi?

We analyze the mechanisms and effectiveness of DeFi auditing in our study by conducting mixed-method research across nine top DeFi auditing firms, 45 DeFi projects, and real-world DeFi incidents. Furthermore, we explore user perception through interviews, revealing that DeFi auditing steps are largely unknown to users, suggesting an understanding deficit in this field.

2 Data Collection

We curated a dataset of DeFi auditing through three main avenues: DeFi attacks, DeFi auditing firms and DeFi applications, and DeFi vulnerabilities.

2.1 DeFi Attacks

We examined real-world DeFi attacks between April 2018 and April 2022, using a dataset encompassing 181 attacks reported on platforms such as Rekt News[4], SlowMist[5], PeckShield[3], and Medium [2]. Each attack can be instigated by multiple vulnerabilities across several system layers, adding complexity to their logistics.

2.2 DeFi Auditing Firms and DeFi Applications

We selected nine prominent auditing firms, based on criteria like number of completed audits, reputation of audited DeFi projects, and development of popular security tools. Security information was collected from the firms' websites and social media.

Furthermore, 45 recent projects audited by these firms were selected to explore current DeFi auditing practices. Our dataset included the audit reports, official websites, and social media of these projects. Refer to Table 4 for detailed information.

2.3 DeFi Vulnerabilities and Taxonomy

We adopted Zhou et al.'s [67] taxonomy to categorize DeFi vulnerabilities, focusing on smart contract layer, protocol layer, and third-party layer vulnerabilities. We derived our vulnerability dataset from academic papers and 45 audit reports, resulting in 49 identified vulnerabilities. See Appendix A for a detailed list and explanation of these vulnerabilities.

3 DeFi Auditing Mechanisms

In this section, we elucidate how DeFi auditing discovers DeFi application security vulnerabilities (**RQ1**).

3.1 DeFi Auditing Workflow

Method To understand DeFi auditing mechanisms, we reviewed 45 audit reports from nine different firms (Table 4), as well as their official websites. We identified various auditing steps each firm took during their audit process. This was done by direct mentions or inferences made from reading audit reports in depth. Inclusion was also made for steps cited on the firms' websites and GitHub pages detailing their audit process. We assumed all tools developed by the firms were employed for auditing when their websites mentioned them.

Findings We find that DeFi audits can be divided into two distinct methodologies: tool-assisted analysis and manual analysis. Additionally, auditors recommend solutions for any problems identified through the utilization of these techniques. The statistics and further explanation of these audit steps are presented in Appendix C.

4 Effectiveness of DeFi Auditing

We scrutinize the efficiency of DeFi auditing via a detailed analysis, focusing on its merits and flaws in real-life situations to answer **RQ2**.

4.1 Audited vs Non-Audited Projects - Vulnerabilities and Attacks

Method We studied harmful DeFi attacks and classified them as audited or non-audited projects based on the occurrence of audits prior to the attack. The nature of exploited vulnerabilities and the handling of identified vulnerabilities in audits were analysed. We also investigated the rectification status of these vulnerabilities to discern if the exploits were due to negligence or inadequate rectification.

Findings The dataset included 189 exploited vulnerabilities, out of which 140 were from non-audited projects and 43 from audited ones. Most vulnerabilities were located at the smart contract and protocol layers. While audited projects had a slightly higher proportion of vulnerabilities at the smart contract layer, non-audited projects had a significantly higher proportion at the third-party layer. The efficacy of auditing in curbing third-party layer issues was statistically confirmed using a U test method. Refer to Figure 1 for detailed information.

DeFi Auditing Detection Capability We found that in 43 audited projects that were attacked, the audit reports mentioned the exploited vulnerabilities in 7 instances, searched for but did not detect the vulnerabilities in 11 instances, and did not mention the exploited vulnerabilities in 25 instances. The statistics of these three groups are presented in Table 6, Table 7 and Table 8.

Mentioned: Out of the 7 projects, two fixed the mentioned vulnerabilities, one partially resolved them, three did not address the vulnerabilities, and in one project the resolution status was unclear. This indicates the project owners' lack of urgency in addressing security risks.

Searched For, Not Detected: In 11 instances, auditors failed to detect the vulnerabilities they searched for, implying limitations in the detection mechanisms.

Not Mentioned: 25 attack causes weren't mentioned in audit reports, highlighting potential limitations in audit scope or transparency in communication.

5 DeFi Auditing: User Perceptions

This study explores users' perceptions of DeFi auditing, a security practice in the DeFi ecosystem. It examines how DeFi users understand the role of DeFi auditing in affecting their transactions and investments.

5.1 Methodology

Through Twitter, Discord, Telegram, and personal contacts, we recruited participants familiar with DeFi applications and smart contract auditing. From July to December 2022, we interviewed 12 users via Zoom, each lasting about an hour. Participants' experience with DeFi ranged from under 2 years to over 2 years. We recorded and transcribed interviews with permission, analyzed data using thematic analysis, and compiled results into a code book.

5.2 Findings

Necessity of DeFi Auditing Users viewed DeFi auditing as necessary for three reasons: it helps non-technical users understand potential security issues, enhances DeFi project security, and showcases a project's commitment to improving security.

Information Delivery of Auditing Some users found the technical nature of auditing findings challenging to comprehend, indicating a need for more accessible explanations. Others mentioned difficulties finding specific information within audit reports due to either overly abstract reporting or the absence of certain audit aspects.

Auditing Effectiveness Despite acknowledging the importance of DeFi auditing, most interviewees felt the auditing effectiveness was inadequate. Reasons include irregular quality among audit firms, difficulty covering all vulnerabilities technically, and a flawed auditing workflow focusing too much on technical issues without considering underlying business logic.

References

1. Market manipulation. https://www.investor.gov/introduction-investing/investing-basics/glossary/market-manipulation (date of access: August 1, 2022)
2. Medium. https://medium.com/ (date of access: September 5, 2022)
3. Peckshield. https://peckshield.medium.com/ (date of access: September 5, 2022)
4. rekt. https://rekt.news/ (date of access: September 5, 2022)
5. Slowmist hacked. https://hacked.slowmist.io/en/ (date of access: September 5, 2022)
6. What are liquidity pools in defi and how do they work? https://academy.binance.com/en/articles/what-are-liquidity-pools-in-defi (date of access: September 5, 2022)
7. What are cross-chain smart contracts? https://blog.chain.link/cross-chain-smart-contracts/ (date of access: September 5, 2022) (2022)
8. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on ethereum smart contracts (sok). In: Maffei, M., Ryan, M. (eds.) Principles of Security and Trust. pp. 164–186. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)
9. Beosin: Clip smart contract security audit. https://beosin.com/audits/CLIP_202205101825.pdf (date of access: June 13, 2022) (2022)
10. Beosin: Crafting smart contract security audit. https://beosin.com/audits/Crafting_202204141429.pdf (date of access: June 13, 2022) (2022)
11. Beosin: Masterchefv2 smart contract security audit. https://beosin.com/audits/MasterChefV2_202204211554.pdf (date of access: June 13, 2022) (2022)
12. Beosin: Nftaq721 smart contract security audit. https://beosin.com/audits/NFTAQ721_202204181629.pdf (date of access: June 13, 2022) (2022)
13. Beosin: Season swap smart contract security audit. https://beosin.com/audits/SeasonSwap_202204221127.pdf (date of access: June 13, 2022) (2022)
14. CertiK: Security assessment argo - audit - addendum. https://www.certik.com/projects/argo (date of access: June 13, 2022) (2022)
15. CertiK: Security assessment decaswap finance. https://www.certik.com/projects/decaswap-finance (date of access: June 13, 2022) (2022)
16. CertiK: Security assessment hunny swap. https://www.certik.com/projects/hunny-swap (date of access: June 13, 2022) (2022)
17. CertiK: Security assessment the space. https://www.certik.com/projects/the-space (date of access: June 13, 2022) (2022)
18. CertiK: Security assessment vicstep. https://www.certik.com/projects/vicstep (date of access: June 13, 2022) (2022)
19. Chen, H., Pendleton, M., Njilla, L., Xu, S.: A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. https://par.nsf.gov/servlets/purl/10197180 (date of access: September 7, 2022) (2020)
20. ConsenSys: Locking pragmas. https://consensys.github.io/smart-contract-best-practices/development-recommendations/solidity-specific/locking-pragmas/ (date of access: September 3, 2022)
21. ConsenSys: Gluwacoin erc-20 wrapper. https://consensys.net/diligence/audits/2021/10/gluwacoin-erc-20-wrapper/ (date of access: June 13, 2022) (2021)
22. ConsenSys: pstake finance. https://consensys.net/diligence/audits/2021/08/pstake-finance/ (date of access: June 13, 2022) (2021)

23. ConsenSys: Gamma. https://consensys.net/diligence/audits/2022/02/gamma/ (date of access: June 13, 2022) (2022)
24. ConsenSys: Notional protocol. https://consensys.net/diligence/audits/2022/03/notional-protocol-v2.1/ (date of access: June 13, 2022) (2022)
25. ConsenSys: Tribe dao - flywheel v2, xtribe, xerc4626. https://consensys.net/diligence/audits/2022/04/tribe-dao-flywheel-v2-xtribe-xerc4626/ (date of access: June 13, 2022) (2022)
26. Hacken: Smart contract code review and security analysis report for onechain. https://hacken.io/wp-content/uploads/2022/06/Summonersarena_01062022_SCAudit_Report.pdf (date of access: June 13, 2022) (2022)
27. Hacken: Smart contract code review and security analysis report for paribus. https://hacken.io/wp-content/uploads/2022/06/Paribus_25052022_SCAudit_Report_2.pdf (date of access: June 13, 2022) (2022)
28. Hacken: Smart contract code review and security analysis report for tencoins. https://hacken.io/wp-content/uploads/2022/06/Bolide-Strategy_08062022_SCAudit_Report_3.pdf (date of access: June 13, 2022) (2022)
29. Hacken: Smart contract code review and security analysis report for thenextwar. https://hacken.io/wp-content/uploads/2022/05/The_Next_War_02042022_SCAudit_Report_1.pdf (date of access: June 13, 2022) (2022)
30. Hacken: Smart contract code review and security analysis report for vynksafe. https://hacken.io/wp-content/uploads/2022/05/VYNKSAFE_03052022SCAudit_Report1.pdf (date of access: June 13, 2022) (2022)
31. Hertig, A.: What is a flash loan? https://www.coindesk.com/learn/2021/02/17/what-is-a-flash-loan/ (date of access: July 31, 2022)
32. Homoliak, I., Venugopalan, S., Reijsbergen, D., Hum, Q., Schumi, R., Szalachowski, P.: The security reference architecture for blockchains: Toward a standardized model for studying vulnerabilities, threats, and defenses. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9239372 (date of access: September 7, 2022) (2020)
33. Jakobsson, M., Myers, S.: Phishing and countermeasures: understanding the increasing problem of electronic identity theft. John Wiley & Sons (2006)
34. Jurzik, H.: 5 most common deployment mistakes. https://deploybot.com/blog/5-most-common-deployment-mistakes (date of access: September 5, 2022) (2021)
35. Kaleem, M., Mavridou, A., Laszka, A.: Vyper: A security comparison with solidity based on common vulnerabilities. https://arxiv.org/pdf/2003.07435.pdf (date of access: September 7, 2022) (2020)
36. 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 (2016), https://eprint.iacr.org/2016/633.pdf and https://github.com/enzymefinance/oyente (date of access: June 1, 2022)
37. Ng, F.: More than $4.7m stolen in uniswap fake token phishing attack. https://cointelegraph.com/news/more-than-4-7m-stolen-in-uniswap-fake-token-phishing-attack (date of access: September 5, 2022) (2022)
38. PeckShield: Smart contract audit report for arthswap masterchef. https://github.com/peckshield/publications/blob/master/audit_reports/PeckShield-Audit-Report-ArthSwap-MasterChef-v1.0.pdf (date of access: June 13, 2022) (2022)

39. PeckShield: Smart contract audit report for beamswap. https://github.com/peckshield/publications/blob/master/audit_reports/PeckShield-Audit-Report-Beamswap-v1.0.pdf (date of access: June 13, 2022) (2022)
40. PeckShield: Smart contract audit report for defiai. https://github.com/peckshield/publications/blob/master/audit_reports/PeckShield-Audit-Report-DeFiAI-v1.0.pdf (date of access: June 13, 2022) (2022)
41. PeckShield: Smart contract audit report for duet bond. https://github.com/peckshield/publications/blob/master/audit_reports/PeckShield-Audit-Report-Duet-Bond-v1.0.pdf (date of access: June 13, 2022) (2022)
42. PeckShield: Smart contract audit report for kaoyaswap. https://github.com/peckshield/publications/blob/master/audit_reports/PeckShield-Audit-Report-KaoyaSwap-v1.0.pdf (date of access: June 13, 2022) (2022)
43. Quantstamp: Quantstamp security assessment certificate capsulenft. https://certificate.quantstamp.com/full/capsule-nft (date of access: June 13, 2022) (2022)
44. Quantstamp: Quantstamp security assessment certificate nomad. https://certificate.quantstamp.com/full/nomad (date of access: June 13, 2022) (2022)
45. Quantstamp: Quantstamp security assessment certificate pine. https://certificate.quantstamp.com/full/pine (date of access: June 13, 2022) (2022)
46. Quantstamp: Quantstamp security assessment certificate playswoops. https://certificate.quantstamp.com/full/play-swoops (date of access: June 13, 2022) (2022)
47. Quantstamp: Quantstamp security assessment certificate rara. https://certificate.quantstamp.com/full/rara (date of access: June 13, 2022) (2022)
48. RuntimeVerification: Audit report alchemix v2. https://github.com/runtimeverification/publications/blob/main/reports/smart-contracts/Alchemix_v2.pdf (date of access: June 13, 2022) (2022)
49. RuntimeVerification: Audit report blockswap stakehouse. https://github.com/runtimeverification/publications/blob/main/reports/smart-contracts/Blockswap_Stakehouse.pdf (date of access: June 13, 2022) (2022)
50. RuntimeVerification: Security audit report algofi amm and nanoswap. https://github.com/runtimeverification/publications/blob/main/reports/smart-contracts/Algofi-dex-nanoswap.pdf (date of access: June 13, 2022) (2022)
51. RuntimeVerification: Security audit report atlendis protocol. https://github.com/runtimeverification/publications/blob/main/reports/smart-contracts/atlendis-audit-report.pdf (date of access: June 13, 2022) (2022)
52. RuntimeVerification: Security audit report exa finance smart contracts. https://github.com/runtimeverification/publications/blob/main/reports/smart-contracts/EXA_Finance.pdf (date of access: June 13, 2022) (2022)
53. SlowMist: Smart contract security audit report rotl. https://github.com/slowmist/Knowledge-Base/blob/master/open-report-V2/smart-contract/SlowMist%20Audit%20Report%20-%20ROTL_en-us.pdf (date of access: June 13, 2022) (2021)

54. SlowMist: Smart contract security audit report arowana. https://github.com/slowmist/Knowledge-Base/blob/master/open-report-V2/smart-contract/SlowMist%20Audit%20Report%20-%20Arowana_en-us.pdf (date of access: June 13, 2022) (2022)
55. SlowMist: Smart contract security audit report cheersup. https://github.com/slowmist/Knowledge-Base/blob/master/open-report-V2/smart-contract/SlowMist%20Audit%20Report%20-%20CheersUp_alpha8_en-us.pdf (date of access: June 13, 2022) (2022)
56. SlowMist: Smart contract security audit report laqiratoken. https://github.com/slowmist/Knowledge-Base/blob/master/open-report-V2/smart-contract/SlowMist%20Audit%20Report%20-%20LaqiraToken_en-us.pdf (date of access: June 13, 2022) (2022)
57. SlowMist: Smart contract security audit report starcrazy. https://github.com/slowmist/Knowledge-Base/blob/master/open-report-V2/smart-contract/SlowMist%20Audit%20Report%20-%20Starcrazy_en-us.pdf (date of access: June 13, 2022) (2022)
58. Stevens, R.: Inflationary and deflationary cryptocurrencies: What's the difference? https://www.coindesk.com/learn/inflationary-and-deflationary-cryptocurrencies-whats-the-difference/ (date of access: September 5, 2022)
59. Swende, M.H.: Eip-1884: Repricing for trie-size-dependent opcodes. https://eips.ethereum.org/EIPS/eip-1884#motivation (date of access: July 31, 2022)
60. Torres, C.F., Steichen, M., et al.: The art of the scam: Demystifying honeypots in ethereum smart contracts. https://www.usenix.org/system/files/sec19-torres.pdf (date of access: September 7, 2022) (2019)
61. TrailofBits: Advanced blockchain security assessment. https://github.com/trailofbits/publications/blob/master/reviews/AdvancedBlockchainQ12022.pdf (date of access: June 13, 2022) (2022)
62. TrailofBits: Degate security assessment. https://github.com/trailofbits/publications/blob/master/reviews/DeGate.pdf (date of access: June 13, 2022) (2022)
63. TrailofBits: Looksrare security assessment. https://github.com/trailofbits/publications/blob/master/reviews/LooksRare.pdf (date of access: June 13, 2022) (2022)
64. TrailofBits: Maple labs security assessment. https://github.com/trailofbits/publications/blob/master/reviews/MapleFinance.pdf (date of access: June 13, 2022) (2022)
65. TrailofBits: Perpetual protocol v2 security assessment. https://github.com/trailofbits/publications/blob/master/reviews/PerpetualProtocolV2.pdf (date of access: June 13, 2022) (2022)
66. Wang, Y., Zuest, P., Yao, Y., Lu, Z., Wattenhofer, R.: Impact and user perception of sandwich attacks in the defi ecosystem. In: CHI Conference on Human Factors in Computing Systems. pp. 1–15 (2022)
67. Zhou, L., Xiong, X., Ernstberger, J., Chaliasos, S., Wang, Z., Wang, Y., Qin, K., Wattenhofer, R., Song, D., Gervais, A.: Sok: Decentralized finance (defi) attacks. Cryptology ePrint Archive (2022)

A DeFi Vulnerability

A.1 Smart Contract Layer Vulnerabilities

1. Under-priced Opcodes: An imbalance between the gas price of executing an operation in the EVM and the resource consumption (CPU time, memory etc.) can be exploited, e.g. by flooding the network, leading to "denial of service" [59].
2. Outdated Compiler Version: The smart contract is compiled using an outdated compiler version, which might contain unresolved bugs and vulnerabilities [35, p.6].
3. Compiler version not fixed / Different Solidity versions used: The source code specified the compiler version with the caret operator "^", allowing it to be compiled with a future compiler version, which might not ensure backward compatibility, be buggy, or introduce syntax changes [20].
4. Call to the unknown: The smart contract calls other smart contracts that execute untrusted code, such as some of the primitives used in Solidity to invoke functions and to transfer ether, e.g. *call*, may have the side effect of invoking the fallback function of the callee or recipient. [8, p.169]
5. Reentrancy: The external callee's contact calls back a function in the caller's contract (that is, a circular call) before the caller's contract expires, allowing an attacker to bypass validation until the calling contract runs out of ether or the transaction runs out of gas [19, p.9].
6. Delegatecall / call injection: To facilitate code-reuse, the Ethereum Virtual Machine (EVM) provides an opcode, *delegatecall*, for inserting a callee contract's bytecode into the bytecode of the caller contract, [19, p.9] hence the malicious callee contract is called in the context of caller contract and can directly modify or manipulate its state variables.
7. Unchecked call return value / Unhandled or mishandled exception: Return values of functions may return important information about the program state, or success or failure of execution. A discrepancy exists in Solidity's handling of exceptions occurring in the execution of callee contracts: If the exception occurs directly referencing a callee contract's instance, the transaction is reverted and the exception is propagated to the caller, if the exception occurs by invoking *send*, *call*, or *delegatecall*, false is returned to the caller [8, p.170 f.].
8. Call stack depth limit: The EVM implementation limits the call stack's depth to 1024 frames and deliberately exceeding the call stack's depth limit causes instructions to fail. [19, p.13].
9. Locked or frozen assets: Allowing users to deposit their money to their contract accounts, but preventing them from spending their money from those accounts, effectively freezing their money. This can be accomplished by e.g. contracts not providing any function for spending money, relying on the money-spending function of another contract, e.g. a library, and the library being killed [19, p.9].

10. Integer overflow or underflow: The result of an arithmetic operation can fall outside of the range of a Solidity data type, causing e.g. unauthorized manipulation of balance or state variables [19, p.10].
11. Absence of coding logic or sanity check: Method arguments and return values that could cause malicious actions during the execution of the method are not checked or validated.
12. Short address: Manipulation of the data padding scheme to change function parameters (e.g. ether value) by providing an address, which is too short, as a function parameter.
13. Casting: The Solidity compiler does not check if types match in all cases and does not throw exceptions, hence the caller is not aware that code is executed in an unexpected manner [8, p.172].
14. Unbounded or gas costly operation: Each block has a "gas limit" field that specifies the maximum total amount of gas that can be consumed by the transactions in a block. When the amount of gas required for executing a contract exceeds the block gas limit, e.g. by looping over a large data structure, the transaction will not be executed [19, p.11].
15. Other arithmetic mistakes: Programming oversight when performing computer arithmetic, not unique to smart contracts.
16. Other non-arithmetic mistakes: Programming oversight not relating to computer arithmetic, from erroneous constructor names to faults in translating business logic into smart contract code.
17. Inconsistent or improper access control: The smart contract does not adequately authenticate access to restricted fields, allowing e.g. permissions to be overwritten or funds to be siphoned to external accounts.
18. Visibility error: Solidity provides four types of visibility to restrict access to a contract's functions, namely, public, external, internal, and private, which respectively says that a function can be called arbitrarily, only externally, only within the contract and derived contracts, or only within the contract. When visibility is incorrectly specified, it can permit unauthorized access [19, p.10].
19. Lackluster test coverage: The percentage of smart contract code, for which functionality tests are written, is too low, or the tests do not sufficiently test corner cases.
20. Other smart contract layer vulnerabilities: Other security vulnerabilities located in the smart contract layer according to the taxonomy laid out in Chapter 2.3.

A.2 Protocol Layer Vulnerabilities

1. Front-running: The result of certain transactions depends on the order in which they are executed. Since transactions are publicly broadcast to the network, a malevolent Externally Owned Account (EOA) can offer a higher gas price to have its transactions included in blocks sooner than others'. Moreover, a malicious miner can always pick up its own transactions irrespective of the gas price [19, p.13].

2. Back-running: The result of certain transactions depends on the order in which they are executed. Since transactions are publicly broadcast to the network, a malevolent EOA can offer a lower gas price to have its transactions included in blocks later than others'. Further, a malicious miner can always pick up its own transactions irrespective of the gas price [19, p.13].
3. Sandwiching: An attacker observes a non-executed transaction purchasing an asset and quickly front-runs it by purchasing that asset for a low price. As the supply of the twice-bought asset is low, its value increases by the design of the bonding curve. The attacker then back-runs by selling the asset shortly after the victim transaction to make a profit [66, p.1].
4. Other transaction order dependency: The result of certain transactions depends on the order in which they are executed, which can be manipulated by means other than front-/back-running or a combination thereof.
5. Transaction / strategy replay: Transactions are often validated with digital signatures. If a digital signature does not depend on due information, e.g. a nonce, a malicious entity could use a digital signature multiple times, e.g. to withdraw extra payments.
6. Randomness: Relying on the pseudo-random or predictable data, such as the block number or timestamp, as a source of randomness, e.g. for encryption, is not safe, since it can allow a prediction of the next "random" number [19, p.13f.].
7. Other block state dependency: Other means by which manipulable elements of the block state are depended upon, e.g. a contract may use the block timestamp, which can be manipulated as a triggering condition to execute some critical operations [36, p.4].
8. On-chain oracle manipulation: To circumvent relying on third parties for pricing assets, some DeFi applications solely retrieve prices from on-chain oracles, which can be manipulated, e.g. with flash loans [31].
9. Governance flash borrow or purchase: Protocols that implement decentralized governance may employ governance tokens, holders of which can propose and vote to change the protocol. Through flash loans [31], entities may temporarily amass a large enough share of tokens to unilaterally change the protocol.
10. Fake token: Real tokens have certain token standards to abide by and fraudulent token creators build in mechanisms to steal token owners' money, not adhering to these standards. Fake tokens may pose as legitimate tokens, though they are not of equal value, for instance, in attempts to extract information about unsuspecting users [37].
11. Token standard incompatibility: Token standards like ERC20 are Ethereum's technical standards for the implementation of cryptocurrency tokens. They define a standard list of rules that tokens should follow within the larger Ethereum ecosystem, allowing developers to predict exactly how the tokens will interact. Smart contracts are missing return values or missing functionality specified in the standard, potentially leading to undefined behavior. Alternatively, smart contract code is not compatible with special types of tokens, e.g. deflationary tokens [58].

12. Other unsafe DeFi protocol dependency: Other DeFi protocol risks that may occur due to external dependencies.
13. Unfair slippage protection: When performing trades on a blockchain, where asset prices adapt according to the order in which trades are executed, users may wish to only execute a trade if the price variation does not exceed a previously specified tolerance ("slippage"). Applications may not execute trades according to the user-defined slippage.
14. Unfair liquidity providing: Liquidity providers may manipulate the supply of liquidity to control the price of currencies in a liquidity pool [6], e.g. through variants of sandwich attacks.
15. Other unsafe DeFi protocol interaction: Other ways in which interacting with a DeFi protocol could be unsafe.
16. Other protocol layer vulnerabilities: Other security vulnerabilities located in the protocol layer according to the taxonomy laid out in Chapter 2.3.

A.3 Third-Party Layer Vulnerabilities

1. Compromised private key / wallet: The attacker can gain control of the private key or wallet and is able to extract all money, ransoms the account, or the attacker disables wallet usage.
2. Weak password: A third party can hack into an ether account and steal funds by simply correctly guessing a weak password.
3. Deployment mistake: Deployment of smart contracts suffers from risks of regular software deployment, e.g. deploying code manually (inconsistently), not tracking code changes properly between different versions [34], but also suffer deployment challenges related to interoperability between different blockchains [7].
4. Malicious oracle updater: Entities responsible for updating real-time data feeds may benefit from providing manipulated information, often to the detriment of users, e.g. in prediction markets, where people can trade contracts that pay based on the outcomes of unknown future events [32, p.22].
5. Malicious data source: Centralized data feeds provide arbitrary data from a single centralized source, and they build on existing blockchain platforms. Centralized data feeds rely on a trusted party that may misbehave or accidentally produce wrong data, harming users in the process [32, p.22].
6. External market manipulation: Adversaries can manipulate the market price of assets off-chain by disseminating false or misleading information about a company, engaging in a series of transactions to make it appear that the security is being traded more actively, or rigging prices, quotes, or trades to manipulate the perception of demand [1].
7. Backdoor / Honeypot: Honeypots are smart contracts that have an obvious flaw in their design, where users a priori commit a certain amount of ether to a contract, thus allowing ether to be extracted from the contract. However, if a user were to attempt to exploit this apparent vulnerability, it would open a second, as-yet-unknown trapdoor for him, preventing him from successfully ejecting ether. [60, p.3]. The attacker takes the money that the victim lost in the exploitation attempt.

8. Insider activities: Since DeFi is less regulated than CeFi, acting on confidential or non-public information for one's benefit or collusion is more common.
9. Phishing attack: Phishing is a form of social engineering in which an attacker sends deceptive messages or deploys malicious software designed to cajole people into disclosing sensitive information to the perpetrator. As a result, unwanted funds are transferred to the attacker [33, p.1].
10. Authority control or breach of promise: Creators or administrators of DeFi projects, such as coins or platforms, may act maliciously by breaking implicit or explicit commitments of trustworthiness to users.
11. Faulty wallet provider: Third-party wallet providers, which DeFi projects rely on, may include, deliberately or negligently faulty, exploitable code in the provided wallets.
12. Faulty API / RPC: The JSON-RPC of an Ethereum client exposes various APIs for EOAs to communicate with the Ethereum network. For security reasons, an attacker could access his client remotely via a JSON request, so the interface should not be reachable from the internet [19, p.17].
13. Other third-party layer vulnerabilities: Other security vulnerabilities located in the third-party layer according to the taxonomy laid out in Chapter 2.3.

Table 1: Smart contract layer vulnerabilities grouped by their respective causes. Explanations of each vulnerability can be found in Appendix A.1.

Vulnerability Cause	Vulnerability Type
State transition design / implementation error	Under-priced opcodes
	Oudated compiler version
	Compiler version not fixed / Different Solidity versions used
Untrusted callee	Call to the unknown
	Reentrancy
	Delegatecall / call injection
Coding mistake	Unchecked call return value / Unhandled or mishandled exception
	Call-stack depth limit
	Locked or frozen assets
	Integer overflow or underflow
	Absence of coding logic or sanity check
	Short address
	Casting
	Unbounded or gas costly operation
	Other arithmetic mistakes
	Other non-arithmetic mistakes
Access control error	Inconsistent or improper access control
	Visibility error
Sub-par code maintenance	Lackluster test coverage
Other	Other smart contract layer vulnerabilities

Table 2: Protocol layer vulnerabilities grouped by their respective causes. Explanations of each vulnerability can be found in Appendix A.2.

Vulnerability Cause	Vulnerability Type
Transaction order dependency error	Front-running
	Back-running
	Sandwiching
	Other transaction order dependency
Replayable design error	Transaction / strategy replay
Block state dependency error	Randomness
	Other block state dependency
Unsafe DeFi protocol dependency	On-chain oracle manipulation
	Governance flash borrow or purchase
	Fake token
	Token standard incompatibility
	Other unsafe DeFi protocol dependency
Unsafe DeFi protocol interaction	Unfair slippage protection
	Unfair liquidity providing
	Other unsafe DeFi protocol interaction
Other	Other protocol layer vulnerabilities

Table 3: Third-party layer vulnerabilities grouped by their respective causes. Explanations of each vulnerability can be found in Appendix A.3.

Vulnerability Cause	Vulnerability Type
Faulty operation	Compromised private key / wallet
	Weak password
	Deployment mistake
Off-chain oracle manipulation	Malicious oracle updater
	Malicious data source
	External market manipulation
Greedy project owners or other internet entities	Backdoor / Honeypot
	Insider activities
	Phishing attack
	Authority control or breach of promise
Faulty blockchain service provider	Faulty wallet provider
	Faulty API / RPC
Other	Other third-party layer vulnerabilities

B Data Collection

Table 4: The sampled DeFi applications of each audit company.

Company	Sampled DeFi Projects
CertiK	Vicstep [18], The Space [17], Argo [14] Hunny Swap [16], Decaswap Finance [15]
ConsenSys	Fei Labs [25], pSTAKE Finance [22], Gluwacoin [21] Gamma [23], Notional Protocol [24]
Trail of Bits	Advanced Blockchain [61], DeGate [62], Looks Rare [63] Maple Labs [64], Perpetual Protocol V2 [65]
Beosin	Alpha Quark [12] Clip [9], SeasonSwap [13] MasterChefV2 [11], Crafting [10]
SlowMist	ROTL [53], LaqiraToken [56], CheersUp [55] Arowana [54], Starcrazy [57]
PeckShield	DeFiAI [40], KaoyaSwap [42], ArthSwap [38] Beamswap [39], Duet Bond [41]
Quantstamp	PlaySwoops [46], CapsuleNFT [43], Pine [45] Rara [47], Nomad [44]
Hacken	Paribus [27], TheNextWar [29], VYNKSAFE [30] Onechain [26], Bolide [28]
Runtime Verification	Algofi [50], EXA Finance [52], Blockswap Stakehouse [49] Atlendis Protocol [51], Alchemix v2 [48]

C Auditing Mechanism Findings

Table 5: Core steps of DeFi auditing. We include companies if they consider the specific step during their audits. A: CertiK, B: ConsenSys, C:Trail of Bits, D: Beosin, E: SlowMist, F: PeckShield, G: Quantstamp, H: Hacken, I: Runtime Verification.

Category	Auditing Step	Companies
Tool-Assisted Analysis	Static Analysis	A, B, C, D, E, F, G, H, I
	Symbolic Execution	B, C, G, H, I
	Fuzzing	B, C, D, E, H
	Formal Verification	A, B, C, D, I
Manual Analysis	Vulnerability Verification and Testing	A, B, C, D, E, F, G, H, I
	Advanced DeFi Scrutiny	A, B, C, D, E, F, G, H, I
	Semantic Consistency Analysis	A, B, D, F, G, H, I
Suggest Remediations	Vulnerability Remediation	A, B, C, D, E, F, G, H, I
	Coding Practice Recommendation	A, B, C, D, E, F, G, H, I

Tool-assisted analysis aims to identify potential vulnerabilities in smart contract codes through automated scrutiny, negating the need for an in-depth analysis of the protocol's business logic. Various tools are used to achieve this, with firms like Quantstamp and Trail of Bits explicitly mentioning their usage of specific tools such as Slither and Echidna respectively in their reports.

Manual Analysis is often employed to expand the scope and increase the accuracy of vulnerabilities identified. This step is crucial as automated tools may generate false positives. After collecting potential vulnerabilities identified by tools, auditors manually verify them, perform code testing, and conduct advanced DeFi scrutiny and semantic consistency analysis.

Remediation Suggestion provides solutions for each identified vulnerability. Audit reports contain detailed descriptions of all vulnerabilities found, including their severity, type, example exploit, and remediation recommendations. They also provide suggestions for coding practices improvement and compliance with

industry standards. The reports also state whether the vulnerabilities have been resolved after auditing, with statuses categorized as Resolved, Partially resolved, Acknowledged, or No Information provided.

D Auditing Effectiveness Findings

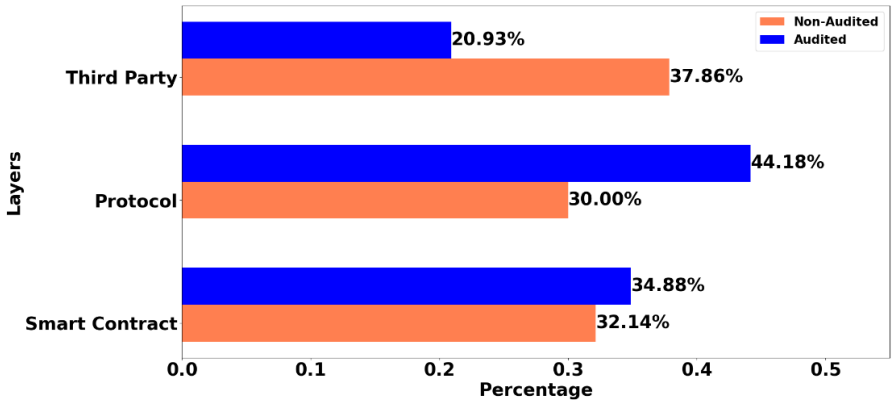

Fig. 1: Percentage distribution of 43 exploited vulnerabilities across different layers.

Table 6: The frequency of appearance of auditing firms in Not Mentioned Group

Audit Company	Number of cases
CertiK	8
Haechi	2
Solidity	2
OpenZepplin	2
ZOKYO	2
Harborn	1
Onmniscia	1
Arcadia	1
Cleanunicorn	1
InterFi Network	1
Consensys	1
Hacken	1
Solidified	1
Nick Johnson(Personal)	1

Table 7: The frequency of appearance of auditing firms in Searched For Group

Audit Company	Number of cases
CertiK	5
PeckShield	2
TechRate	1
Harborn	1
Quantstamp	1
Personal Audit	1

Table 8: The category of vulnerablities in Searched For Group

Cause of attack	Number of cases
Absence of code logic or sanity check	5
Call to untrusted contract	3
Reentrancy	1
Oracle manipulation	1
Fake Token	1