Ethereum Mainnet Smart Contracts Upgrades - December 2nd, 2023

Mónica Jin

1 Introduction

Smart contracts are deterministic programs deployed on blockchain networks. They facilitate interactions among multiple mutually distrusting parties without requiring intervention from a third party. Executed on the Ethereum Virtual Machine (EVM) and typically written in high-level programming languages like Solidity, Ethereum smart contracts are deployed by compiling them into EVM bytecode and including them in a transaction. Once deployed, a unique address is assigned to a smart contract. Similar to any transaction recorded in the blockchain, smart contracts exhibit immutability, preventing modifications after inclusion in the blockchain.

The characteristics of smart contracts, such as trustless execution, transparency, and immutability, have catalyzed the emergence of rapidly growing industries, including Decentralized Finance (DeFi) and NFT trading.

Despite their immutability, smart contracts can be designed to be upgradeable. While the contracts themselves remain immutable, it is possible to maintain the state/storage of the contract and alter its behavior. This feature of upgradeability is crucial for addressing post-deployment bugs and vulnerabilities. Moreover, it enables the addition of new features to the original contract while preserving its state.

The objective of this analysis is to offer insights into Ethereum smart contract upgrades. Understanding methods to detect smart contract upgrades is vital for comprehending the evolution of smart contracts and the adoption of upgradeable patterns. This analysis specifically concentrates on identifying the proxy contracts that underwent upgrades on the Ethereum mainnet on December 2nd, 2023.

Our findings reveal 19 smart contracts that were upgraded on the specified date. All relevant data, source code, and results for this research are accessible in the following repository: `https://github.com/Mokita-J/Ethereum-Smart-Contract-Upgrades`

2 Smart Contract Upgradeability

Traditionally, smart contracts are designed to be immutable and resistant to tampering once deployed. However, innovative software engineering techniques have emerged to introduce upgradeability through various methods. In this context, upgradeability denotes the capacity to modify a smart contract post-deployment while preserving its data and state. Several patterns [4, 3, 5] have been devised to facilitate smart contract upgradeability. These patterns offer structured approaches and methodologies to address the challenges associated with modifying deployed smart contracts,

ensuring their upgradability. Multiple patterns [4, 3, 5] have been developed for smart contract upgradeability.

2.1 Consensus Override

An approach to smart contract upgrades involves the concept of consensus override, which essentially entails altering the blockchain itself. The immutability of smart contracts is deeply rooted in the consensus protocol governing the blockchain. Smart contracts inherit their immutability from the consensus mechanisms adhered to by all miners within the blockchain network. In Ethereum, for instance, the consensus protocol relies on a majority of participants reaching agreement on the valid state and growth of the blockchain. If more than 51% of miners opt for a chain distinct from the original, this alternative chain becomes the prevailing one, enabling the replacement of the original smart contracts with updated versions.

2.2 Social Upgrade

An alternative method for smart contract upgrades involves a social approach, where the new version of the contract is deployed on a different address. The upgrade process includes migrating the state of the original contract into the newly deployed version, followed by notifying users to transition to the updated contract. This type of upgrade doesn't necessitate the implementation of additional features by the contract developer to support upgradeability. However, it does require careful engineering, particularly for contracts where developers need to devise mechanisms for transferring data seamlessly from the old contract to the new one.

2.3 CREATE2-based Metamorphosis

The execution of smart contracts in the EVM relies on a set of opcodes. The creation of a smart contract involves the use of the CREATE opcode, which computes a unique address for the contract based on the sender's address and nonce. The nonce, a unique number associated with an Ethereum account, increments with each transaction, ensuring sequential and non-reusable values.

The introduction of the CREATE2 and SELFDESTRUCT opcodes has presented an innovative avenue for smart contract deployment. CREATE2 enables the creation of a contract with an address defined by the developer, while SELFDESTRUCT terminates a contract and removes its code from the blockchain.

Smart contracts could be upgraded using a combination of these opcodes. The upgrade process involved invoking SELFDESTRUCT in a deployed contract, followed by creating a new contract on the "destructed" contract's address using CREATE2. However, it's crucial to note that this smart contract upgrade pattern is no longer supported by Ethereum, following the deprecation of SELFDESTRUCT on October 27th, 2022 [1].

2.4 Data Logic separation

Smart contracts employing the Data Logic Separation upgrade pattern are explicitly designed with upgradeability in mind. This pattern involves the division of the smart contract program into at least two distinct contracts:

[1] https://eips.ethereum.org/EIPS/eip-6049

1. The Logic Contract: This contract defines the behavior of the smart contract;
2. The Storage Contract: This contract is responsible for storing the data and state of the program.

Commonly known as the proxy pattern, this design pattern enables the creation of upgradeable contracts. While not all proxy contracts inherently support upgradeability, they can be made upgradeable if the proxy includes a variable storing the address of the logic contract. This address variable can be modified through a function call, thus rendering the proxy contract upgradeable.

To execute the logic contract, a proxy contract utilizes the DELEGATECALL opcode [2]. This opcode invokes a method from another contract while utilizing the storage of the calling contract. The Data Logic Separation upgrade pattern exemplifies a strategic approach to achieving flexibility and upgradeability in smart contracts.

3 Upgrade Detection

Detecting smart contract upgrades is a complex task that is intricately tied to the unique implementation of each contract. Several challenges contribute to the difficulty of upgrade detection:

1. Opacity of Code: Smart contracts are permanently stored on the blockchain in bytecode format, which presents a challenge for human interpretation due to its low-level representation. Although decompilation tools for EVM bytecodes exist, they often suffer from information loss. Developers can provide human-readable source code, and platforms like Etherscan enable them to publish the source code through Smart Contract Verification.
2. Lack of Standardization: The absence of a standardized method for smart contract upgrades adds to the complexity. Additionally, smart contract function calls are recorded through transactions, where the data field contains the first 4 bytes of the hash of the called function's name and argument types, followed by the function's input values. Analyzing which functions have been called through transactions becomes challenging when the contract's source code is unavailable.

The detectability of each upgradeability pattern is outlined below:

- Consensus Override: If the blockchain is overridden, detecting a smart contract upgrade might be the least of our worries, considering the potential security breaches. Luckily, as of December 2nd, 2023, there were no global headlines announcing Ethereum's compromising, and unsurprisingly, no contract upgrades were detected for this case.
- Social Upgrade: This upgrade relies on social communication to disseminate the new smart contract address, making it non-detectable in the code.
- CREATE2-based Metamorphosis: Fröwis et al.[1] extensively studied this upgrade pattern, identifying 41 contracts between March 2019 and July 2021 that upgraded using this method. This analysis does not explore in depth this pattern.
- Data Logic Separation: The focus of this analysis is on detecting upgrades in upgradeable proxy contracts. The subsequent subsection details the methodology employed for upgrade detection.

[2] https://ethervm.io/#F4

3.1 Methodology

To pinpoint the smart contracts that underwent upgrades on December 2nd, 2023, this analysis relies on the recorded events within the blockchain for that specific date.

3.1.1 Events

Events, generated by transactions, furnish valuable insights into transaction executions. Programmed within the smart contract's source code, events facilitate communication between smart contracts and user interfaces. Each event not only specifies the executed function but also details its input values. The initial bytes of an event are computed using the Keccak-256 hash function, employing the function's name and its corresponding argument types as input.

Well-established libraries like OpenZeppelin utilize standardized events to document smart contract upgrades. In this study, the following events and their associated hash outputs are employed to detect potential upgrades:

- ProxyUpdated(address,address):
  `0xd32d24edea94f55e932d9a008afc425a8561462d1b1f57bc6e508e9a6b9509e1`
- Upgraded(address):
  `0xbc7cd75a20ee27fd9adebab32041f755214dbc6bffa90cc0225b39da2e5c2d3b`
- DiamondCut(FacetCut[],address,bytes):
  `0xbc7cd75a20ee27fd9adebab32041f755214dbc6bffa90cc0225b39da2e5c2d3b`

3.1.2 Pipelines

To compile the list of smart contracts upgraded on December 2nd, 2023, the processes outlined in Figure 1 were executed. Google BigQuery and Etherscan served as data sources for this endeavor. BigQuery played a role in extracting addresses from the Ethereum blockchain, while Etherscan facilitated the extraction of source code associated with these smart contract addresses. In terms of tools, Slither, coupled with the USCHunt [2] plugin, was instrumental in the analysis.

The first pipeline (P1) initiates the listing of smart contracts upgraded on the specified date, relying on emitted events. It commences by extracting all smart contract addresses that emitted one of the events detailed in Section 3.1.1. With these addresses in hand, the source code is extracted, enabling an analysis of each contract's upgradeability. Slither's USCHunt plugin sifts through the contracts, filtering out those that aren't upgradeable, eliminating false positives.

The second pipeline (P2) addresses potential false negatives from the previous pipeline—contracts not identified through common events. This pipeline extracts all smart contract addresses accessed via transactions on the specified date. Following a parallel process of source code extraction and upgradeable filtering, contracts containing the specific events in their code are excluded. If a contract features code related to any of the three events but none were emitted, it suggests no upgrade occurred.

The third pipeline (P3) focuses on listing potential CREATE2-based metamorphosis upgrades. It kicks off by extracting addresses of contracts created on the specified date and extracting their source code. Although incomplete, this pipeline provides an upper limit of contracts that this methodology may have overlooked, as it doesn't analyze whether the contracts adhere to the CREATE2-based pattern.

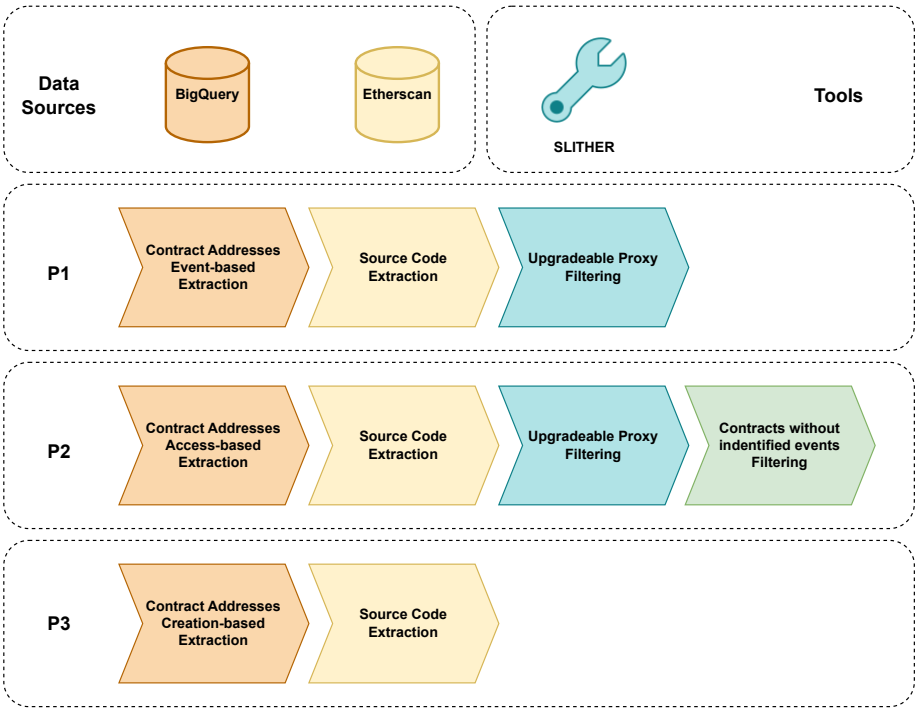

Figure 1: Smart Contracts upgrade methodology

4 Results

This section unveils the outcomes stemming from the implementation and execution of the pipelines.

The results gleaned from Pipeline 1 showcase an enumeration of all smart contracts upgraded on December 2nd, 2023. Among 591 contracts that emitted events detailed in Section 3.1.1, 27 had published source code on Etherscan and underwent subsequent analysis. Employing Slither, the analysis identified 19 upgradeable proxy contracts from this subset. Consequently, the deduction is that a minimum of 19 smart contract upgrades occurred on the Ethereum mainnet on the specified date.

Pipeline 2's results enumerate potential proxy smart contract upgrades that evaded detection by Pipeline 1. On the date in question, 18,422 contracts were accessed, and among these, 2,304 proxy source codes were available. Slither categorized 652 of these contract codes as upgradeable. Notably, 593 of these upgradeable contracts featured one of the listed events, leaving 59 without any event inclusion. Consequently, Pipeline 2, at its maximum, identifies 59 potential smart contract upgrades overlooked by Pipeline 1, either due to insufficient event consideration or a lack of emitted upgrade events.

Pipeline 3 delineates potential smart contracts created to update their predecessors. Detecting 763 contract creations on the specified date implies that the methodology might have missed a maximum of 763 contracts upgraded using the CREATE2-based metamorphosis.

5 Discussion

While successfully identifying 19 smart contracts upgraded on December 2nd, 2023, my approach encounters limitations when it comes to detecting contracts without publicly available source code on Etherscan. Since smart contracts are stored in the blockchain in bytecode format, our upgrade detection method relies on a static code analysis technique for Solidity and Vyper programming languages. Consequently, if a contract's code is not publicly accessible in the considered data sources, this method may not yield conclusive results.

Identifying two threats to the validity of this study, the first revolves around potential issues in the code implementation. Bugs in the executed code could introduce discrepancies between the methodology description and its actual implementation. To mitigate this threat, both the source code used and the collected data are available in the associated GitHub repository. The second threat relates to the event assumptions. This study relies on three event hash outputs to identify potential updates, computed using Keccak-256 with the function's name and argument types. While hash collisions (two different inputs producing the same output) are unlikely, they remain a theoretical possibility. Additionally, the executed behavior of the function is not explicitly recorded in the blockchain's logs, leaving room for identified events to potentially have meanings other than assumed, such as parallel interpretations unrelated to smart contract upgrades.

This work presents avenues for further exploration. While upgradeable proxy contracts take center stage in this analysis, other upgradeable smart contract patterns warrant exploration. Although briefly introducing CREATE2-based upgradeable contracts and quantifying potential upgrades, there is potential for further analysis of available source code to detect upgradeability in these contracts. Future work can delve deeper into unexplored upgradeable patterns, enhancing our understanding of the diverse landscape of smart contract upgrades.

6 Conclusion

While smart contracts are typically considered immutable entities on the blockchain, advancements in software engineering have introduced the concept of upgradeability. This capability proves invaluable for rectifying bugs and introducing new features post-deployment. In an effort to shed light on the dynamic landscape of smart contract upgrades, this analysis focuses on listing proxy smart contracts that underwent upgrades on December 2nd on the Ethereum mainnet. Leveraging event-based data collection and static code analysis, the results unveil 19 proxy contracts that underwent upgrades on that specific date.

References

[1] Michael Fröwis and Rainer Böhme. Not all code are create2 equal. In Shin'ichiro Matsuo, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez Hernandez, Sam Werner, Thomas Haines, Aleksander Essex, Andrea Bracciali, and Massimiliano Sala, editors, *Financial Cryptography and Data Security. FC 2022 International Workshops*, pages 516–538, Cham, 2023. Springer International Publishing.

[2] William Edward Bodell III, Sajad Meisami, and Yue Duan. Proxy hunting: Understanding and characterizing proxy-based upgradeable smart contracts in blockchains. In Joseph A. Calandrino

and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 1829–1846. USENIX Association, 2023.

[3] Sajad Meisami and William Edward Bodell III. A comprehensive survey of upgradeable smart contract patterns. *CoRR*, abs/2304.03405, 2023.

[4] Ilham Qasse, Mohammad Hamdaqa, and Björn Jónsson. Smart contract upgradeability on the ethereum blockchain platform: An exploratory study. *arXiv preprint arXiv:2304.06568*, 2023.

[5] Mehdi Salehi, Jeremy Clark, and Mohammad Mannan. Not so immutable: Upgradeability of smart contracts on ethereum. *arXiv preprint arXiv:2206.00716*, 2022.