Case Study

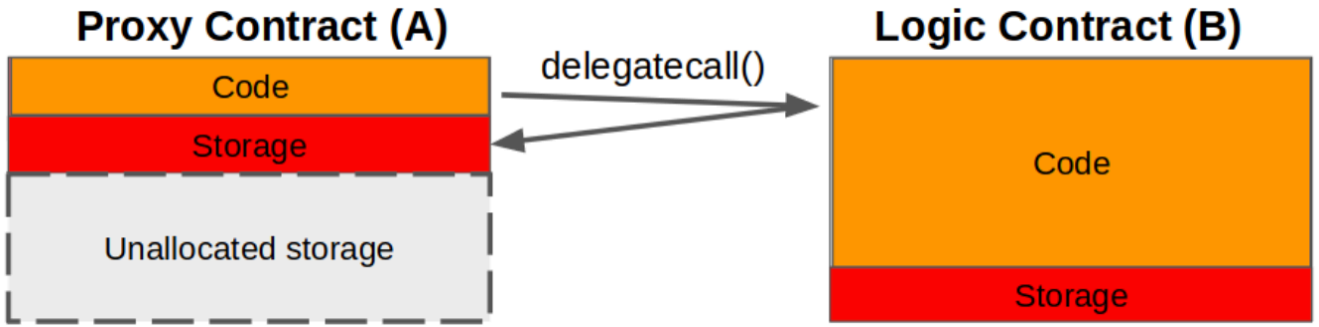
- Harvest Finance Uninitialized Proxies Bug($200k Bounty)

Proxies?

- Smart Contract를 업데이트 하려면?
  - code는 업데이트하면서,
  - storage state는 그대로 가져가야 함

Proxy와 DELEGATECALL 을 이용!

Proxy and DELEGATECALL



- Smart Contract Call에는 3가지 타입이 존재
  - CALL
  - STATICCALL
  - DELEGATECALL

CALL

Example 1:

EOA ------- CALL --------> Contract A -------- CALL --------> Contract B

| msg.sender == EOA address | msg.sender == Contract A address

| msg.value == EOA send value | msg.value == Contract A send value

| storage == Contract A | storage == Contract B

* EOA : Externally Owned Accounts

DELEGATECALL

Example 2:

EOA ---- CALL ------> Contract A ----- DELEGATECALL ------> Contract B

| msg.sender == EOA address | msg.sender == EOA address

| msg.value == EOA send value | msg.value == EOA send value

| storage == Contract A | storage == Contract A

* EOA : Externally Owned Accounts

Example 3:

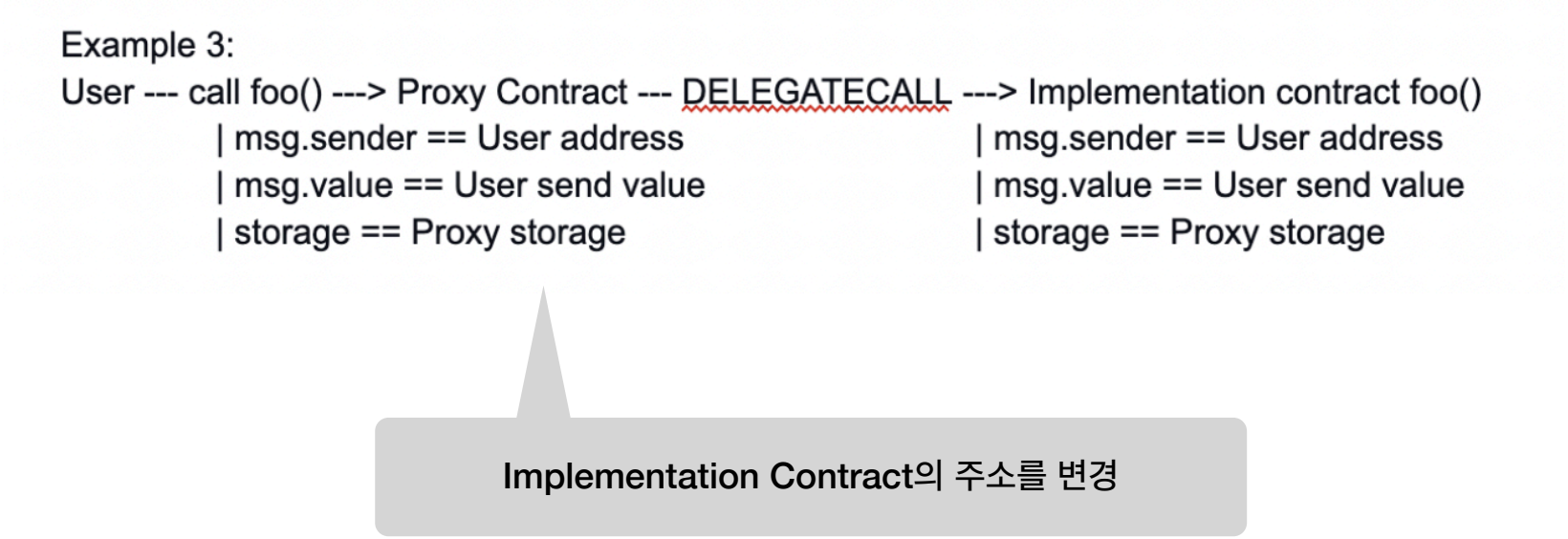
User --- call foo() ---> Proxy Contract --- DELEGATECALL ---> Implementation contract foo()

| msg.sender == User address | msg.sender == User address

| msg.value == User send value | msg.value == User send value

| storage == Proxy storage | storage == Proxy storage

코드 업데이트


Example 3:
User --- call foo() ---> Proxy Contract --- DELEGATECALL ---> Implementation contract foo()
| msg.sender == User address
| msg.value == User send value
| storage == Proxy storage
| msg.sender == User address
| msg.value == User send value
| storage == Proxy storage
Implementation Contract의 주소를 변경

코드 업데이트

Example 3:
User --- call foo() ---> Proxy Contract --- DELEGATECALL ---> Implementation contract foo()
| msg.sender == User address | msg.sender == User address
| msg.value == User send value | msg.value == User send value
| storage == Proxy storage | storage == Proxy storage

새로운 Implementation Contract가 Deploy될 때 Constructor는 한번만 호출 -> Proxy Contract의 storage를 변경할 수 없음 -> 이를 위해 init / initialize 함수 존재 (OpenZeppelin)

Proxy Pattern의 문제

만약 같은 이름의 함수가 Proxy와 Implementation에 있다면 사용자가 호출 시 어느 Contract의 함수를 호출해야하나?

다른 이름의 함수라도 Contract가 다르면 동일 함수로 처리할 수도 있다 -> Solidity Bytecode 상에서 함수 identifier를 4bytes로 처리

Transparent Proxy Pattern

- msg.sender로 구분해서 처리
  - msg.sender가 proxy의 admin이면 무조건 proxy contract의 함수 호출
  - msg.sender가 그 외 주소면 무조건 Implementation contract의 함수 호출

msg.sender	owner()	upgradeTo()	transfer()
Admin	returns proxy owner	upgrades proxy	reverts
Other account	returns ERC20 owner	reverts	sends ERC20 transfer

문제는 caller가 admin인지 구분하는 코드가 항상 있어야 하고, proxy에는 추가적인 관리 코드가 필요 -> gas 효율적이지 않다!

Universal Upgradeable Proxy Standard

- TPP와의 차이
  - TPP는 Upgrade Logic이 Proxy에 존재
  - UUPS는 Upgrade Logic이 Implementation에 존재
    -> Implementation에서 다른 Contract(Proxy)의 Storage를 변경 가능
    - Upgrade 요청시 caller가 admin인지만 확인하면 됨

OpenZeppelin UUPS Uninitialized Proxies

- __Ownable_init()
  - 처음 호출한 사람이 owner로 설정
  - owner는 implementation contract의 upgradeToAndCall() 함수를 Proxy를 거치지 않고 직접 호출 가능
- UpgradeToAndCall() 함수 내부 동작
  - DELEGATECALL로 migration/initialization 함수를 호출
  - 내부적으로 SELFDESTRUCT 실행
    -> Proxy Contract가 empty contract를 호출하게 됨

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.2;

import “@openzeppelin/contracts-
upgradeable/token/ERC20/ERC20Upgradeable.sol”;
import “@openzeppelin/contracts-
upgradeable/proxy/utils/Initializable.sol”;
import “@openzeppelin/contracts-
upgradeable/access/OwnableUpgradeable.sol”;
import “@openzeppelin/contracts-
upgradeable/proxy/utils/UUPSUpgradeable.sol”;

contract MyToken is Initializable, ERC20Upgradeable,
OwnableUpgradeable, UUPSUpgradeable {
  function initialize() initializer public {
    __ERC20_init(“MyToken”, “MTK”);
    __Ownable_init();
    __UUPSUpgradeable_init();
  }

function _authorizeUpgrade(address newImplementation)
    internal
    onlyOwner
    override
  {}

}
```

OpenZeppelin UUPS Uninitialized Proxies

Attack Step-by-step Guide

1. 공격자는 implementation contract의 initialize() 함수를 호출, owner가 된다
2. 공격자는 selfdestruct() 함수가 있는 malicious contract를 배포한다
3. 공격자는 implementation contract의 upgradeToAndCall() 함수를 owner로 호출하여, malicious contract를 가리키게 한다
4. upgradeToAndCall() 함수 내부에서 DELEGATECALL로 malicious contract를 호출, selfdestruct로 인해 implementation contract가 destroy 된다
5. Proxy contract가 destroy된 implementation contract를 가리킨다(useless)

OpenZeppelin UUPS Uninitialized Proxies

Security Advisory

Security advisory: Initialize UUPS implementation contracts

General Announcements

spalladino OpenZeppelin Team 108 5 Sep 10

Update September 14th: We have published a security advisory 7 and released Contracts v4.3.2 19 with a hotfix for this vulnerability. Still, it is now recommended you always initialize your implementation contracts as explained below.

Update September 16th: We have now published a post-mortem with a lot of detail at UUPSUpgradeable Vulnerability Post-mortem 61.

Due to a vulnerability in OpenZeppelin Contracts v4.1.0 through v4.3.1 7, all projects using the UUPS proxy pattern should `initialize` their implementation contracts.

To help mitigate this situation, we have already executed transactions to initialize over 150 implementation contracts from multiple projects we identified across Mainnet, Polygon, xDAI, Binance, and Avalanche. These transactions were sent from address `0x37E8d216c3f6c79eC695FBD0cB9842e62fB84370` via a batching contract at `0x310fAC62C976d8F6FDFA34332a56EA1a05493b5b` on all networks.

Harvest Vulnerability

- 3개의 uninitialized proxy implementation을 제보
  - https://contract-library.com/contracts/Ethereum/392A5C02635DCDBD4C945785CE530A9A69DDA6C2
  - https://contract-library.com/contracts/Ethereum/47228860C3EBEB999AE6CC43286FD94DF264A143
  - https://contract-library.com/contracts/Ethereum/E41E27CD5C99BF93466FCE3F797CF038EFC3C37D
- No published source code, only bytecode is available
- Proxy는 source code 공개
  - OpenZeppelin UUPS 패턴과 다르게 upgrade 로직이 Implementation에 없음

Harvest Vulnerability

Upgrade

```
function upgrade() external {
  (bool should, address newImplementation) =
IUniVaultV1(address(this)).shouldUpgrade();
  require(should, “Upgrade not scheduled”);
  _upgradeTo(newImplementation);
 …
```

- shouldUpgrade
  - Implementation에서 처리
  - UUPS 패턴과 동일한 문제 발생
- Attack 시나리오
  - Implementation contract를 initialize
  - hard-worker storage field를 SELFDESTRUCT를 호출하는 Contract로 설정
  - doHardWork 호출 -> Implementation을 Destroy

Harvest Vulnerability

How to find

- OpenZeppelin의 UUPS uninitialized implememtation 취약점을 일반화
  - Uninitialized contract X가 untrusted caller로부터 caller가 선택한 주소로 DELEGATECALL을 호출
  - contract X에 initializer(init, initialize) 존재
  - Initializer가 한번도 호출된 적 없음 -> 과거 Transaction 탐색
  - (Contract Y가 X를 DELEGATECALL 호출한 적 있음)

Reference

- https://medium.com/immunefi/harvest-finance-uninitialized-proxies-bug-fix-postmortem-ea5c0f7af96b
- https://blog.openzeppelin.com/the-transparent-proxy-pattern/
- https://docs.openzeppelin.com/contracts/4.x/api/proxy#transparent-vs-uups
- https://forum.openzeppelin.com/t/security-advisory-initialize-uups-implementation-contracts/15301