Sound Proxy Usage
Contents

1. What is a Proxy?
2. Transparent vs UUPS
3. Best Practices
What is a Proxy?
Contract P

use_code_at = <Address of Contract A>
Contract P

use_code_at = <Address of Contract A>

Contract A

<CODE>
Contract P

use_code_at = <Address of Contract A>

Contract A

number = 0

function getNumber() {
    return number
}

function setNumber(_number) {
    number = _number
}
setNumber(3)

Contract P

use_code_at = <Address of Contract A>
number = 3

Contract A

number = 0

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}
Contract A

number = 0

function getNumber() {
    return number
}

function setNumber(_number) {
    number = _number
}

Contract P

use_code_at = <Contract A>
number = 3

Contract Q

use_code_at = <Contract A>
Contract A

```
number = 0

function getNumber() {
    return number
}

function setNumber(_number) {
    number = _number
}
```

setNumber(7)

Contract P

```
use_code_at = <Contract A>
number = 3
```

Contract Q

```
use_code_at = <Contract A>
number = 7
```
setNumber(1)
CALL

- Callee code
- Callee storage
- Callee address
CALL
- Callee code
- Callee storage
- Callee address

DELEGATE CALL
- Callee code
- Caller storage
- Caller address
Contract A

number = 1

function getNumber() {
    return number
}

function setNumber(_number) {
    number = _number
}

Contract P

use_code_at = <Contract A>

number = 3
Contract A

number = 1

function getNumber() {
    return number
}

function setNumber(_number) {
    number = _number
}

Contract P

use_code_at = <Contract A>
number = 3
Contract A

```javascript
number = 1

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}
```

Contract B

```javascript
number = 0

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}

function getDoubleNumber() {
  return number * 2
}
```

Contract P

```javascript
use_code_at = <Contract B>
number = 3
```
Contract A

number = 1

function getNumber() {
    return number
}

function setNumber(_number) {
    number = _number
}

Contract B

number = 0

function getNumber() {
    return number
}

function setNumber(_number) {
    number = _number
}

function getDoubleNumber() {
    return number * 2
}

Contract P

use_code_at = <Contract B>
number = 3
Contract P

use_code_at = <Contract B>
number = 3

Contract B

number = 0

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}

function getDoubleNumber() {
  return number * 2
}
**Contract P**

```typescript
use_code_at = <Contract B>
number = 3
```

**Contract B**

```typescript
number = 0

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}

function getDoubleNumber() {
  return number * 2
}
```

GetDoubleNumber()  

6
Transparent vs UUPS
TRANSPARENT vs UUPS

Both use DelegateCall, but differ in how they manage upgrading process.

- **Transparent**
  - Upgrade logic handled in the proxy contract

- **UUPS**
  - Storage in proxy
  - Delegate call based
  - Uses logic contract
  - Uses implementation contract for upgrades
Upgrade logic handled in the proxy contract

proxy functionality requires a designated administrator

UUPS

Uses implementation contract for upgrades

implementation contract decides who is able to upgrade base on access control.
TRANSPARENT vs UUPS

Transparent

UUPS
TRANSPARENT vs UUPS

```cpp
/**
 * @dev If caller is the admin process the call internally, otherwise transparently fallback
 */

function _fallback() internal virtual overrides {
    if (msg.sender == _admin) {
        bytes memory ret;
        bytes4 selector = msg.sig;
        if (selector == ITransparentUpgradeableProxy.upgradeTo.selector) {
            ret = _dispatchUpgradeTo();
        } else if (selector == ITransparentUpgradeableProxy.upgradeToAndCall.selector) {
            ret = _dispatchUpgradeToAndCall();
        } else {
            revert ProxyDeniedAdminAccess();
        }
        assembly {
            return(add(ret, 0x20), mload(ret))
        }
    } else {
        super._fallback();
    }
}
/**
```
TRANSPARENT vs UUPS

```
/*
 * @custom:oz-upgrades-unsafe-allow-reachable delegatecall
 */
function upgradeToAndCall(address newImplementation, bytes memory data) public payable virtual onlyProxy {
    _authorizeUpgrade(newImplementation);
    upgradeToAndCallUUPS(newImplementation, data, true);
}
```
Best Practices
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.
1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.

```cpp
constructor() {
    _disableInitialization();
}
```
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.
3. Set up and initialize in one transaction.
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.
3. Set up and initialize in one transaction.
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.
3. Set up and initialize in one transaction.

Transaction 1
- action 1: deployment

Transaction 2
- action 2: initialisation
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.
3. Set up and initialize in one transaction.
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.
3. Set up and initialize in one transaction.
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.
3. Set up and initialize in one transaction.
4. Don’t use both Transparent and UUPS.
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use initializers, not constructors.
3. Set up and initialize in one transaction.
4. Don’t use both Transparent and UUPS.
5. Only use trusted code for implementation contracts.