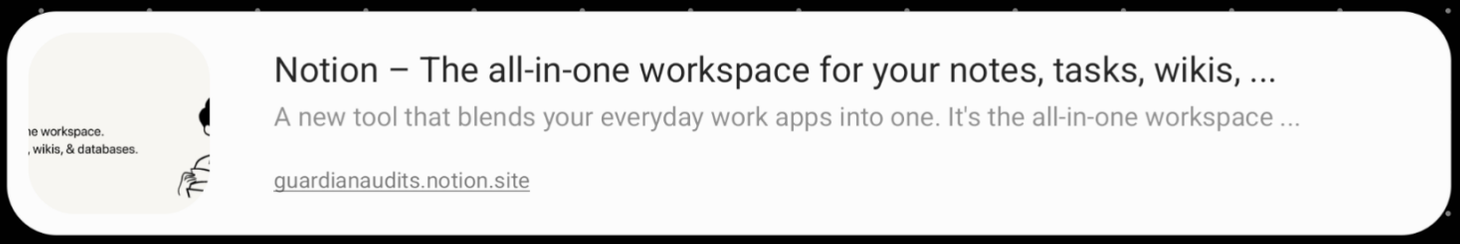

1 A protocol plans to have dozens of vault contracts which it wishes to be able to upgrade all at the same time, which upgradeability pattern should they use?

Beacon Proxy.
When beacon is updated with new implementation all vaults will upgrade simultaneously

2 Order the 5 upgradeability patterns in terms of gas efficiency upon user interactions, explain your reasoning for each placement.

- UUPS → No admin check
- Transparent → SLOAD logic contract slot + admin check.
- Beacon → Need to make external calls to get logic address before delegatecall.
- Diamond → Run through a mapping then SLOAD Also depends from implimentation.

3 How can storage collisions occur when using a proxy?

If Proxy and implimentation contract have difference in their state variable structure / order wrong storage slot of Proxy will be written over by delegate call to implimentation

4. Do storage collisions apply when using

the Eternal Storage upgradeability pattern? Why?

No.

No delegate calls → No Storage Collisions

5. What is function selector clashing and how can it cause unexpected issues?

when Storage and proxy has a function with similar selector then it can cause confusions as to which contracts function to use.

In these situation the contract on which call was made is given priority. It can be exploited to call sensitive proxy functions instead of intended impli.. functions.

6. What is the purpose of the safe-upgrade gap variable in the folowing MixinRoles contract? Why is it necessary?

https://github.com/unlock-protocol/unlock/blob/a37d3a09f6d3fddb0372767aeeef5c5391c29677/smart-contracts/contracts/

As mixin contract inherits an upgradeable contract gaps is used as a precaution to prevent storage collisio in case inherited contract adds new state variables.

7 What differences are there between the transparent and UUPS upgradeability patterns? What are the pros/cons of each?

Transparent proxies has upgrade function in proxy. whereas UUPS has it in implimentation

Pros - UUPS gas efficient

Cons - UUPS can be bricked if non upgrade implimentation is pushed.

8. Briefly explain the Diamond Standard at a high level. What does it achieve? How does it achieve it?

Diamond standard maps each function to different

delegatecall target address.

- Infinitely modular.
- Instead of upgrading entire implementation only update specific function.

9. What is malicious about the following proxy? What other risks exist?

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract TokenProxy {

    address public proxyOwner;
    address public implementation;

    constructor(address _implementation) {
        proxyOwner = msg.sender;
        _setImplementation(_implementation);
    }

    modifier onlyProxyOwner() {
        require(msg.sender == proxyOwner);
        _;
    }

    function upgrade(address _implementation) external onlyProxyOwner {
        _setImplementation(_implementation);
    }

    function _setImplementation(address imp) private {
        implementation = imp;
    }

    fallback () external payable {
        (bool success, bytes memory data) = implementation.delegatecall(msg.data);
    }

    function voting_var(address, uint256, int128, int128) external {
        implementation.delegatecall(abi.encodeWithSignature(
            "transfer(address,uint256)", proxyOwner, 100 ether
        ));
    }
}
```

-> Return values not checked

'voting_var()' will clash with 'totalSupply()' func of logic contract and then it will be executed. delegating call to transfer funds

Logic to transfer to admin

10. Is this an appropriate implementation of a UUPS compliant implementation contract? Why or why not?

https://glist.github.com/owenThurm/b047b4f253b210fbe2c76422d39ac495#file-uupsimplementation-sol

UUPSImplementation.sol — Raw

```
1  pragma solidity 0.8.19;
2
3
4  import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
5  import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
6  import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
7
8  contract UUPSImpl is Initializable, OwnableUpgradeable, UUPSUpgradeable {
9
10   constructor() {
11       _disableInitializers();
12   }
13
14   function initialize(address _endpoint) public initializer {
15       __Ownable_init();
16       __UUPSUpgradeable_init();
17   }
18
19   function _authorizeUpgrade(address newImplementation) internal override {}
20
21   function upgradeTo(address newImplementation) public override onlyOwner onlyProxy {
22       _upgradeToAndCallUUPS(newImplementation, new bytes(0), false);
23   }
24
25 }
```

lacks 'only Owner' modifier

is not overriding 'upgradeToAndCall' in inherited implementation