OpenZeppelin

Service Monitoring and Emergency Response

with OpenZeppelin Defender

zpl.in/defender-workshop

Steven Landers
steven@openzeppelin.com

Workshop Agenda

- OpenZeppelin
- Defender
- Sentinels
- Real World Example
- Q & A

OpenZeppelin

Our mission is to protect the open economy

OpenZeppelin is a software company that provides **security audits** and **products** for decentralized systems.

Projects from any size -from new startups to established organizations- trust OpenZeppelin to build, inspect and connect to the open economy.

Security, Reliability and Risk Management

OpenZeppelin provides a complete suite of **security and reliability products** to build, manage, and inspect all aspects of software development and operations for Ethereum projects.

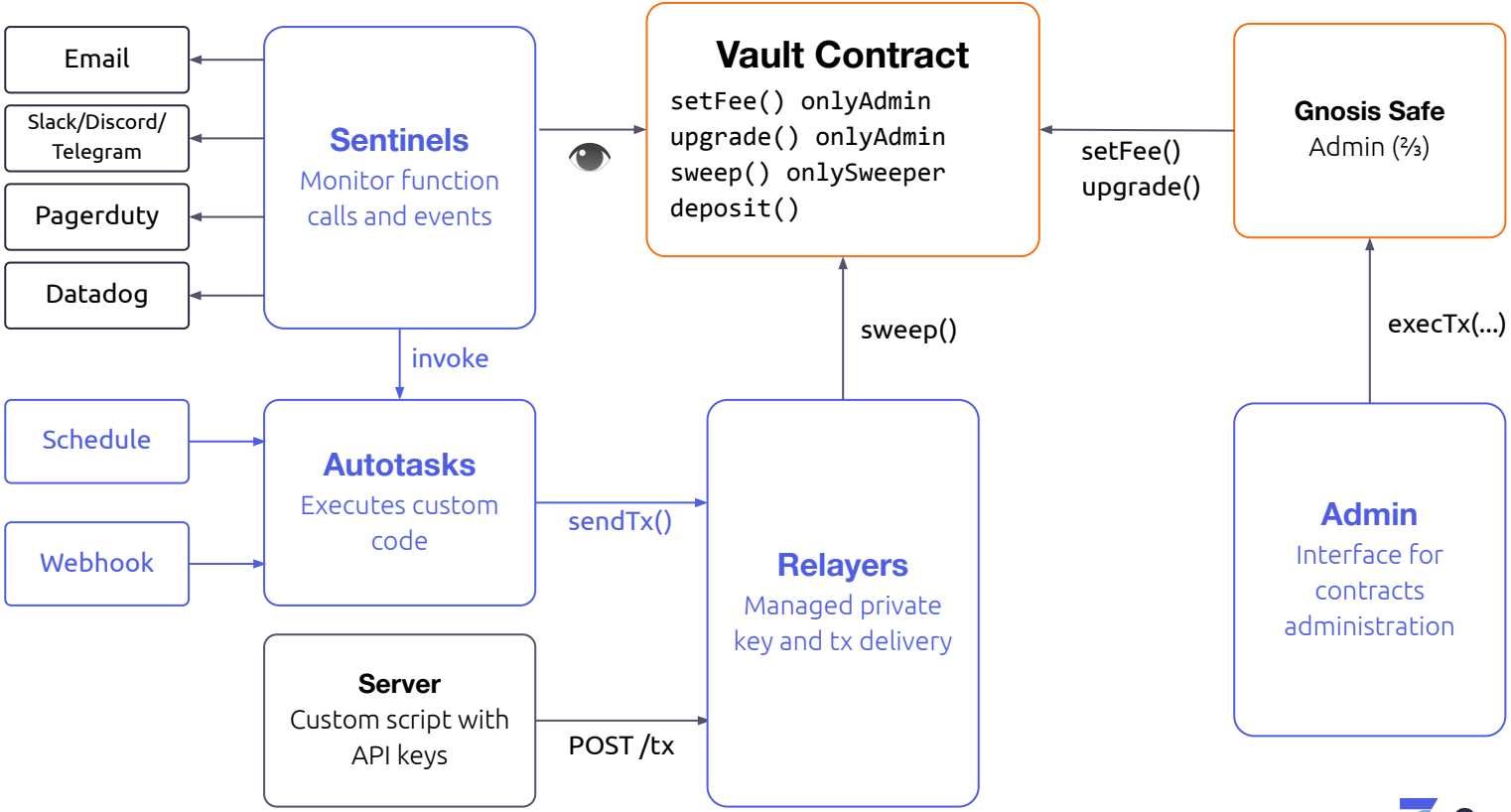
Defender

Operations Security (OpSec) Platform

Defender Features

- **Admin -** interface for contracts administration
- **Relayer -** secure hosted keys and reliable transaction delivery
- **Autotasks** - serverless code for automated tasks and off-chain logic
- **Sentinels** - monitor transactions, trigger notifications & Autotasks
- **Advisor -** best-practices in securing your decentralized system

Defender Features



OpenZeppelin

Features that fit together

*A **Sentinel** can trigger an **Autotask** that can use a **Relayer** to respond to a transaction*

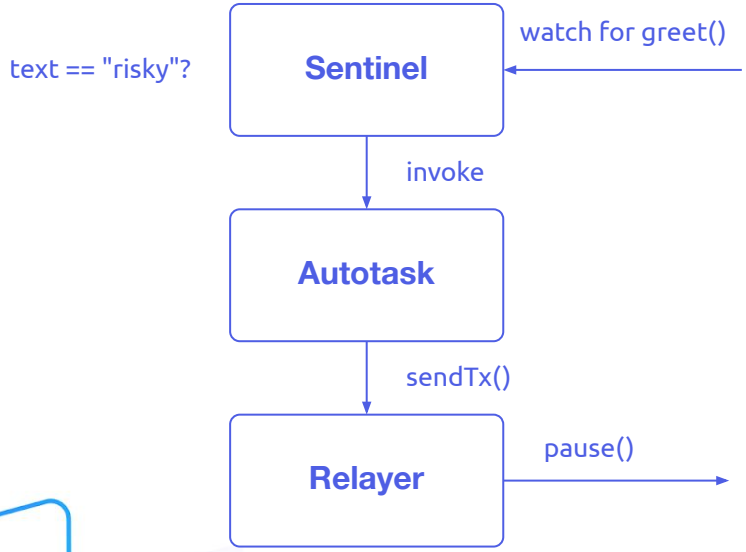
Monitoring Transactions

Introduction to Sentinels

Exercise: Greeter Example

Detect a “risky” Greeting

Greeter Example




```
pragma solidity >=0.7.0 <0.8.0;

contract Greeter {
    event Greeting(string text, address sender);
    //...props, modifiers...

    function greet(string memory text) notPaused public {
        emit Greeting(text, msg.sender);
    }

    function unpause() adminOnly public {
        paused = false;
    }

    function pause() adminOnly public {
        paused = true;
    }
}
```

Sentinel Notifications

Popular Platforms

Integrate with the tools you already use. Custom integrations are available via **Autotask** or email (many tools like PagerDuty offer an email interface)

Telegram

@Email

DISCORD

Autotask

slack

DATADOG

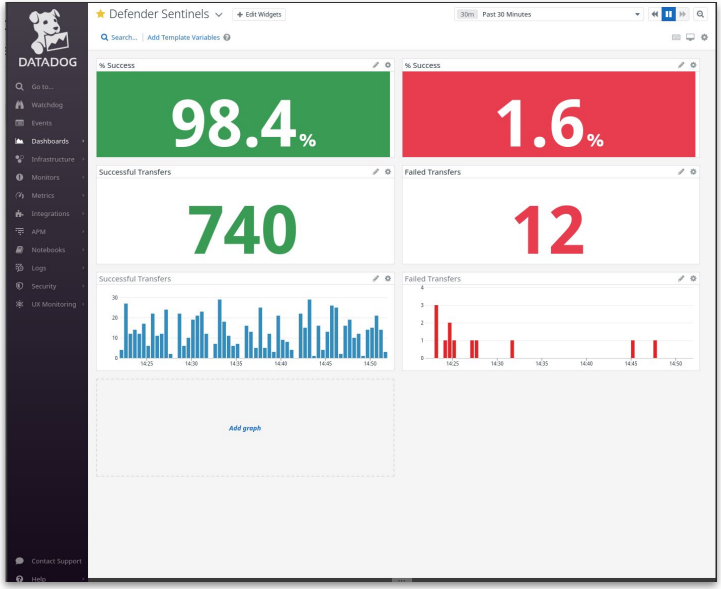
Sentinel Integrations

Advanced Analysis through Integrations

Use the tool for the job. Datadog is great at taking multiple metrics, relating them, and surfacing alerts.

Monitor **Successful** vs. **Failed** transactions, and alert when the Successful % dips below **95%** over a **15 minute** period

Datadog Integration (USDT success vs. failed)



Autotask Integration

Sentinels send events to Autotasks

Events contain key information about the transaction and the reasons for the Sentinel triggering.

https://docs.openzeppelin.com/defender/sentinel#event_schema

```
{
  "transaction": {
    ...eip-1474 receipt...
  },
  "blockHash": "0xab..123",
  "matchReasons": [
    {
      "type": "event",
      "signature": "Transfer(...)",
      "condition": "value > 5"
    }
  ],
  "sentinel": {
    "id": "44a7d5...31df5",
    "name": "Sentinel Name",
    "abi": [...],
    "address": "0xabc..123",
    "confirmBlocks": 1,
    "network": "mainnet"
  }
}
```

What should I monitor?

- Administrative or Sensitive Actions
- Spikes in Transaction Volume
- Spikes in Failed Transactions
- Significant Gas Price Changes
- Loss of System Funds
- Anomalous Transactions

Other Scenarios

- Alert if `setFee(uint256)` is not consistent with Oracle
- Alert if transaction sends >1000 ETH to my contract
- Integrate a custom webhook when `Purchase( )` is emitted with details
- Send an email to security for any call to `transferOwnership(address)`
- Send a slack for every transaction involving a sensitive address
- Others….

Consider an Audit

*Consider a formal **audit** to surface key monitoring recommendations.*

https://openzeppelin.com/request/

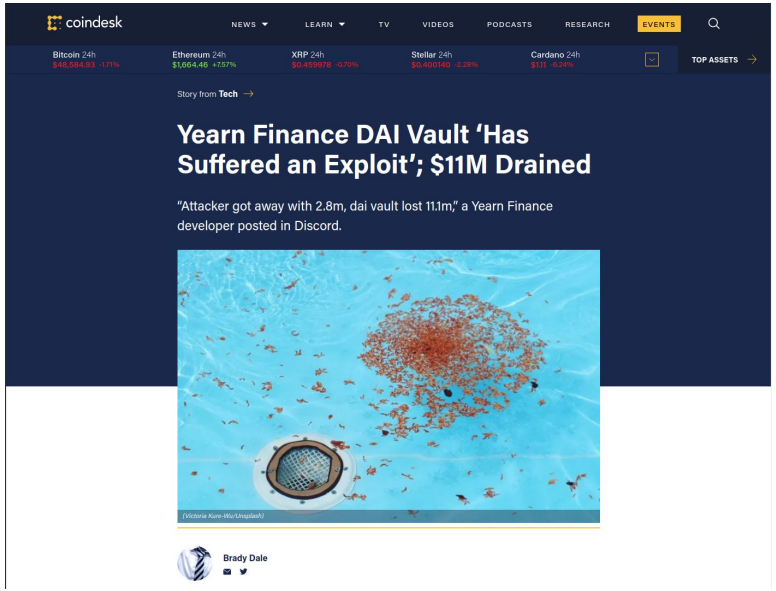
Yearn 2021/2/4 Exploit

What Happened

Yearn Exploit - 2021/2/4

- Vault Lost **$11.1m**
- Attacker gained **$2.8m**
- Mitigated after **38 Minutes**
- **11 Transactions**

https://www.coindesk.com/yearn-finance-dai-vault-exploit



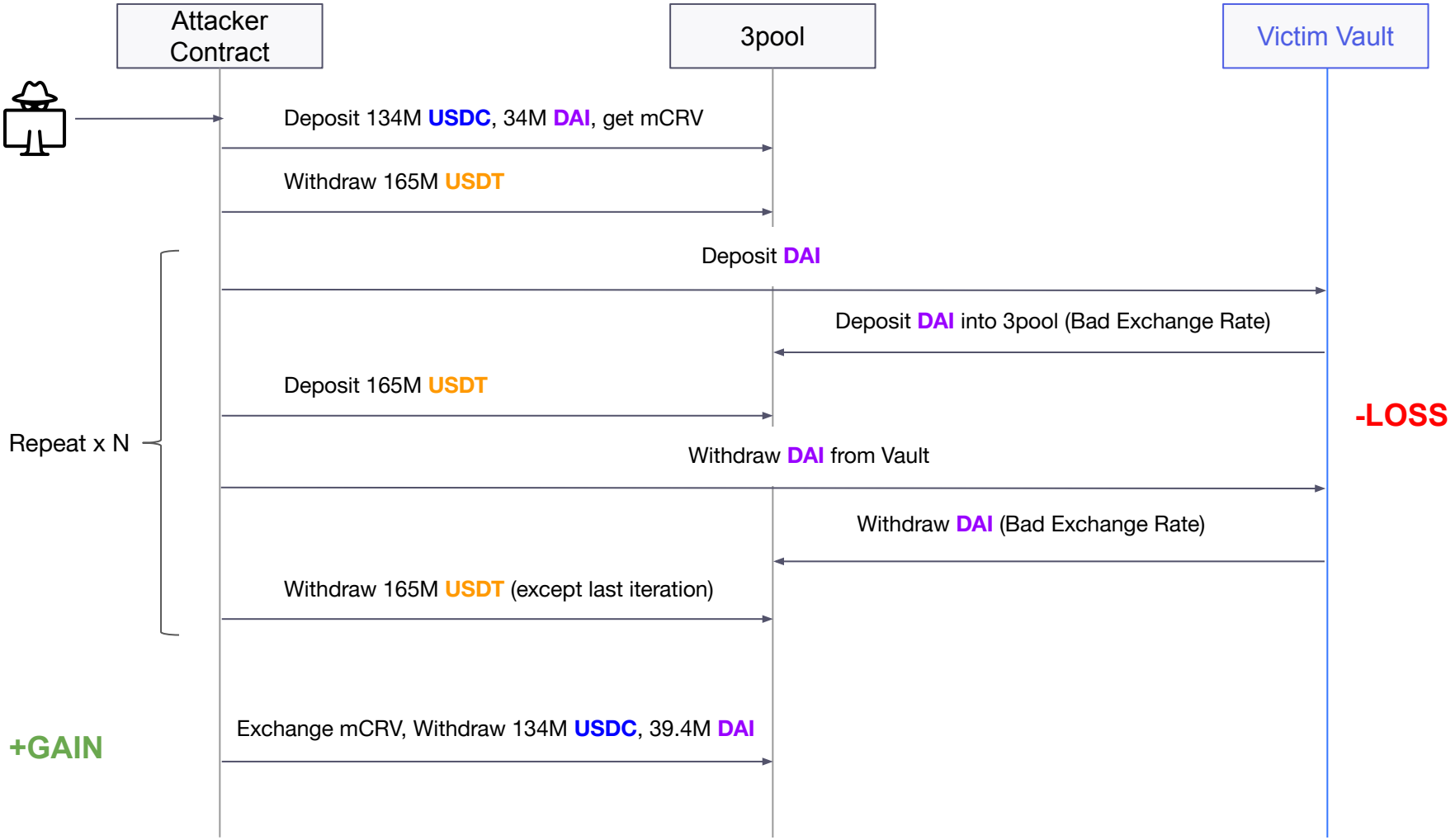
Yearn Exploit - 2021/2/4

"At a high level, the exploiter was able to profit through the following steps:

1. *Debalance the exchange rate between stablecoins in Curve's 3CRV pool.*
2. *Make the yDAI vault deposit into the pool at an unfavorable exchange rate.*
3. *Reverse the imbalance caused in step 1.*

This pattern was repeated in a series of ***11 transactions executed over 38 minutes*** *before being mitigated."*

Yearn Disclosure
https://github.com/yearn/yearn-security/blob/master/disclosures/2021-02-04.md


Attacker Contract
3pool
Victim Vault
Deposit 134M USDC, 34M DAI, get mCRV
Withdraw 165M USDT
Deposit DAI
Deposit DAI into 3pool (Bad Exchange Rate)
Deposit 165M USDT
-LOSS
Repeat x N
Withdraw DAI from Vault
Withdraw DAI (Bad Exchange Rate)
Withdraw 165M USDT (except last iteration)
Exchange mCRV, Withdraw 134M USDC, 39.4M DAI
+GAIN

Detecting Anomalies

Basic sanity checks are often enough to detect that something isn't quite right.

What could have detected this?

- **Yearn v1 yDAI vault**
- **Large Complex Transaction**
  - `gasUsed > X`
  - `earn(), withdraw(), deposit()`
- **Loss of Funds**
  - `balance()`
  - Previous Block vs. This Block

https://github.com/yearn/yearn-security/blob/master/disclosures/2021-02-04.md

At 21:45 (UTC), Andre Cronje notices the **complex transaction pattern** of a contract that is interacting with **Yearn vaults**. Yearn's security team is called into action, and what eventually is determined to be an active exploit on Yearn's v1 yDAI vault, is mitigated 11 minutes later.

Exercise Summary

1. Watch the **Vault address** for function calls
2. Trigger when a transaction is found with **gasUsed > 7M**
    a. Sentinel sends a **Slack Alert**
    b. Invoke an **Autotask**
        i. **Autotask** determines whether there is a loss
        ii. **Autotask** sends separate **Slack Alert** (webhook) if loss exceeds threshold
        iii. (Could have sent mitigation transaction)

```
// Autotask receives event from sentinel
exports.handler = async function(payload) {
  const provider = new DefenderRelayProvider(payload);
  const evt = payload.request.body;

  // init vault contract
  const block = await provider.getBlock(evt.blockHash)
  const yVault = new ethers.Contract(evt.sentinel.abi, evt.sentinel.address, provider);

  // get balance for this block and previous block
  const currentBlockBalance = await yVault.balance({blockTag: block.number});
  const prevBlockBalance = await yVault.balance({blockTag: block.number-1})

  // send alert if loss exceeds threshold
  const delta = currentBlockBalance.sub(prevBlockBalance);
  const exceedsThreshold = delta.lte(threshold);

  if(exceedsThreshold) {
    // call webhook to slack
    await sendLossAlert(payload, evt, delta);
  }

  // return for autotask logging
  return {currentBlockBalance, prevBlockBalance, delta, exceedsThreshold }
}
```

Exercise: Detect a Real Exploit

Detect the Yearn Exploit

defender.openzeppelin.com
docs.openzeppelin.com
forum.openzeppelin.com

Thank you!

Learn more

openzeppelin.com/**defender**
forum.openzeppelin.com
docs.openzeppelin.com

Contact

steven@openzeppelin.com