**Defender**

OpenZeppelin Defender provides a security operations (SecOps) platform for Ethereum with built-in best practices. Development teams implement Defender to ship faster and minimize security risks.

## Components

Automate your Ethereum operations to deliver high-quality products faster with lower risk to users. Learn more about each component by clicking on its card.

* Admin
* Relay
* Sentinel
* Autotask
* Advisor

# Admin

The Defender Admin service acts as an interface to **manage your smart contract project** through secure **multisig contracts** or **timelocks**. Defender Admin holds no control at all over your system, which is fully controlled by the keys of the signers.

## Use cases

Use Defender Admin whenever you need secure management of your smart contracts on-chain. Any administrative operation should not be unilaterally controlled by a single owner. Instead, set up a multi-signature wallet and rely on Defender Admin to run any of the following actions through it:

* **Upgrading a contract** to a new implementation
* **Tweaking a numerical parameter** in your protocol that affects its behavior
* Managing an **access control list** for restricted operations
* **Pausing** your contract in the event of an emergency

## Security considerations

Defender Admin acts exclusively as an interface to your contracts and multi-signature wallets. This means that you do not grant Defender any rights over your contracts by using Admin to manage them. All proposal approvals are signed client-side using the admin user private key through Metamask. The Defender Admin backend is only involved in storing proposal metadata and sharing the approval signatures when these are not stored on-chain. Ultimately, the multi-signature wallet contracts are the ones that verify these approvals and execute the proposed actions.

Defender Admin’s main contribution to security is then related to usability. First, it automates the process of crafting the transaction for a proposal to avoid manual errors. Second, it provides a clear interface for reviewing a proposal without having to manually decode the proposal hex data.

# Advisor

The Defender Advisor service contains a **knowledge base of security best practices** curated by the OpenZeppelin team. The best practices cover development, testing, monitoring and operations. Defender Advisor can be used as a checklist to prioritize efforts in implementing project security.

## Use cases

Each security best practice in Defender Advisor is rated for criticality and effort, and contains examples for implementation. Use Defender Advisor to:

* **Evaluate your security** versus established best practices
* **Prioritize** additional best practices that you can implement
* Improve **security training and awareness** within your development and operations teams

## Security Best Practices

The best practice articles in Defender Advisor are categorized by Development, Monitoring, Testing and Operations. Each article is also rated for the importance of the best practice either Critical, High or Normal. And each article is identified for the expected effort to implement which can be Large, Medium or Small. Using the controls on the column headers you can filter and sort the list of articles.

# Autotasks

The Defender Autotasks service allows you to **run code snippets** on a **regular basis**, **via webhooks**, or in **response to a transaction**. Thanks to tight integration to [Relay](https://docs.openzeppelin.com/defender/relay) and [Sentinels](https://docs.openzeppelin.com/defender/sentinel), you can use Autotasks to automate regular actions by easily sending transactions or reacting to events from your contracts.

## Use cases

Use Autotasks whenever you have a recurrent action you need to run on your Contracts. Since you can run arbitrary code snippets, you can trigger any transactions you need, checking whatever conditions you want, and gathering info from any external APIs you may need.

* Check your balance in contracts and **sweep funds to a wallet** upon reaching a threshold
* Update an **on-chain oracle** with information from an external API
* **Monitor your contracts** to verify their state or check that an off-chain data source is in-sync
* Poke your contracts to have them **transition to a new state** once a set of conditions is met
* Integrate with external **third party services** via webhooks
* Provide **gasless transactions** for your dapp users via meta-transactions

## What’s in an Autotask?

In a nutshell, an Autotask is a snippet of javascript code invoked at a regular interval, similar to a serverless function. And as a matter of fact, they are implemented with [Lambda functions](#Lamda).

When you create an Autotask, you provide a javascript snippet, choose a trigger for it to run, and optionally [link it to a Relayer](https://docs.openzeppelin.com/defender/autotasks#relayer-integration). Defender currently supports two types of triggers:

* **Schedule**: Choose a frequency to execute your Autotask, and Defender will make sure to invoke your function at the specified interval. Note that the specified interval is between two consecutive execution starts, *not* between the end of an Autotask and the beginning of the next one. Alternatively, you can specify when the Autotask should run using [cron expressions](https://crontab.cronhub.io/).
* **Webhook**: Defender will create a secret URL for your Autotask, and invoke it whenever an HTTP request is POSTed to that endpoint. You can regenerate this URL at any time. Defender will inject the HTTP request information into your Autotask, and return the Autotask run info along with any data you return from your code.

## Security considerations

Each Autotask is implemented as a separate AWS Lambda, ensuring strong separation among each individual Autotask and across Defender tenants.

Autotasks are restricted via Identity and Access Management to have zero access to the Defender internal infrastructure. The only exception is that an Autotask may access its linked Relayer, which is negotiated via temporary credentials injected by the Defender Autotask service upon each execution. Still, the Autotask can only call the Relayer exposed methods and has no direct access to the backing private key.

# Relay

The Defender Relay service allows you to send transactions via a regular HTTP API, and takes care of **private key secure storage**, **transaction signing**, **nonce management**, **gas pricing estimation**, and **resubmissions**. This way you don’t need to worry about securing your private keys in your backend scripts, or by monitoring your transactions to ensure they get mined.

## Use cases

Use a Defender Relayer every time you find yourself using a hot wallet in your code. Whenever you need to send a transaction from a script or your backend, you can connect to a Relayer instead to simplify key management and transaction submissions.

* **Poking your contracts** to trigger a state transition
* Update an **on-chain oracle** with external data
* Sending **meta-transactions** to build a gasless experience
* React to sign-ups in your app by **airdropping tokens** to your new users
* **Sweeping funds** from protocol contracts to secure wallets
* **Building bots** for interacting with smart contract protocols

## What’s in a Relayer?

Each Defender Relayer is an Ethereum account assigned exclusively to your team. Every time you create a new Relayer, Defender will create a new private key in a secure vault. Whenever you request Defender to send a transaction through that Relayer, the corresponding private key will be used for signing.

You can think of each Relayer as a queue for sending transactions, where all transactions sent through the same Relayer will be sent in order and from the same Ethereum account, controlled exclusively by your team.

### Pausing

You can pause a Relayer from the Defender website, to quickly respond to an emergency. This will cause the Relayer to reject any incoming requests to send a transaction, whether it is from an Autotask or via the API. However, keep in mind that any transactions already sent will not be cancelled. When you are ready to resume operations, just hit the Unpause button and your Relayer will go back to normal.

### Policies

You can limit a Relayer’s behavior by specifying policies. Currently, these are the different opt-in policies:

1. **Gas price cap**: specify a maximum gas price for every transaction sent by the Relayer. When this policy is enabled, Defender will overwrite the gasPrice or maxFeePerGas of any transaction that goes beyond the specified cap. Take into account that the gas price for a transaction is specified based on gas price oracles at the moment the Relayer actually sends the transaction to be mined, so this policy can be used as a protection on gas price surges.
2. **Receiver whitelist**: specify a list of authorized recipients for every transaction sent by the Relayer. Defender will reject and discard any transaction whose recipient address is not in the list.
3. **EIP1559 Pricing**: specify if the transactions the Relayer is sending should be EIP1559 by default or not. This applies whenever the Relayer send a transaction with the [speed](https://docs.openzeppelin.com/defender/relay-api-reference#send-transaction) parameter, or a non specified gasPrice or maxFeePerGas/maxPriorityFeePerGas. Note that this is only shown for EIP1559 compatible networks.

To configure your Relayer’s policies, go to your Relayer settings page, click the cog icon button, Manage policies. You will then see a form where you can opt to enable either of these two policies and tweak their parameters.

# Sentinel

The Defender Sentinel service offers 2 types of Sentinels, **Contract Sentinels** and **Forta Sentinels**. Contract Sentinels allow you to **monitor transactions** to a contract by **defining conditions** on events, functions, transaction parameters. Forta Sentinels allow you to **monitor Forta Alerts** by **defining conditions** on Forta Bots, contract addresses, alert IDs and severity. If a Sentinel matches a transaction or a Forta Alert based on your defined conditions it will notify you via email, slack, telegram, discord, [Autotasks](https://docs.openzeppelin.com/defender/autotasks), and more.

## Use cases

Use a Sentinel when you need to know about or respond to transactions and Forta Alerts involving your smart contracts, other smart contracts you need to monitor or Forta Bots. Your Sentinel will watch every transaction and Forta Alert and send the ones you care about to your notification methods of choice.

* **Monitor** your **sensitive functions** like transferOwnership, pause, or upgrade
* **Alert** on potentially dangerous transactions on your contracts
* **Respond** by executing logic when key events happen
* **Integrate** with your **existing tools** through slack, telegram, discord, email or custom **Autotask** integration
* **Know** when an unexpected volume of transactions or alerts occur

### When to use a Contract vs a Forta Sentinel

Contract Sentinels are particularly useful for monitoring transactions to a single contract. With the [Contract Conditions](https://docs.openzeppelin.com/defender/sentinel#specify-conditions) you can filter transactions by various transaction attributes i.e events, functions, params, gasPrice, value, etc.

Forta Sentinels are useful for monitoring one or more of your smart contracts using a more complex criteria, by hooking into the events and conditions defined by Forta Bot Developers. Forta Bots have greater flexibility for defining transaction conditions than a Contract Sentinel.

## What’s in a Contract Sentinel?

A Contract Sentinel watches for transactions related your chosen addresses, filters those transactions by any conditions that you specify, then notifies you when those transactions occur.

## What’s in a [Forta](#forta) Sentinel?

A Forta Sentinel watches Forta Alerts that come from a given Forta Bot or affect a given address, and filters those alerts by any conditions that you specify, then notifies you when those alerts occur.

## **Lambda -**

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. You can trigger Lambda from over 200 AWS services and software as a service (SaaS) applications, and only pay for what you use.

**Forta -**

On the far left there are the Forta developer tools that are used by detection bot developers to build and deploy bots to the Forta Network. These developer tools are described in more detail later on. Once deployed, the detection bots are run by Forta scan nodes. You can think of scan nodes as servers that provide capacity to the Forta Network. Scan nodes are responsible for running detection bots, providing them blockchain data and publishing any alerts. Detection bots use the blockchain data to detect some condition they are interested in, and can also make network calls to other APIs to combine richer data sources.

Refer - <https://docs.openzeppelin.com/defender/> - for defender docs