Lists

About Lists

Lists can be used to create and maintain sets of data for use when creating rules. Lists allow you to easily reuse the same sets of data across multiple rules. Lists can be created on individual sites (Site Lists) as well as the corp as a whole (Corp Lists) to be easily used in multiple sites.

For example, you could create a list of prohibited countries that you don't do business with. You could then use this list in any rules that involve those countries, such as rules to track registration or login attempts originating from those countries. If a prohibited country changes, simply update the list instead of updating every rule that uses it.

Lists can consist of the following types of data:

- Countries
- IP addresses
- Strings
- Wildcards

> **Note:** Lists support CIDR notation for IP address ranges.

Creating a List

Create both Corp and Site lists using these steps.

Corp Lists

1. From the **Corp Rules** menu, select **Corp Lists**. The corp lists menu page appears.
2. Click **Add corp list**. The add corp list menu page appears.
3. From the **Type** menu, select the type of data the list will contain.
4. In the **Name** field, enter the name of the list.
5. Optionally, in the **Description (optional)** field, enter a description for the list.
6. In the **Entries** field, enter the items that will comprise the list. Each entry must be on its own line.
7. Click **Create corp list**.

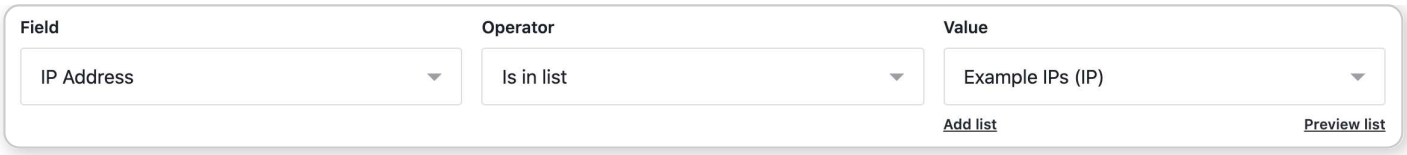
> **Note:** Only Owners can create, edit, and delete Corp Lists. This is because Corp Lists have the ability to manipulate traffic across every site and other user types can only manage Rules and Lists for sites they have access to.

Site Lists

1. From the **Site Rules** menu, select **Site Lists**. The site lists menu page appears.
2. Click **New list**. The new list menu page appears.
3. From the **Type** menu, select the type of data the list will contain.
4. In the **Name** field, enter the name of the list.
5. Optionally, in the **Description (optional)** field, enter a description for the list.
6. In the **Entries** field, enter the items that will comprise the list. Each entry must be on its own line.
7. Click **Create site list**.

Using a List

When creating a rule, select **Is in list** or **Is not in list** for the operator, then select the list from the value dropdown menu.



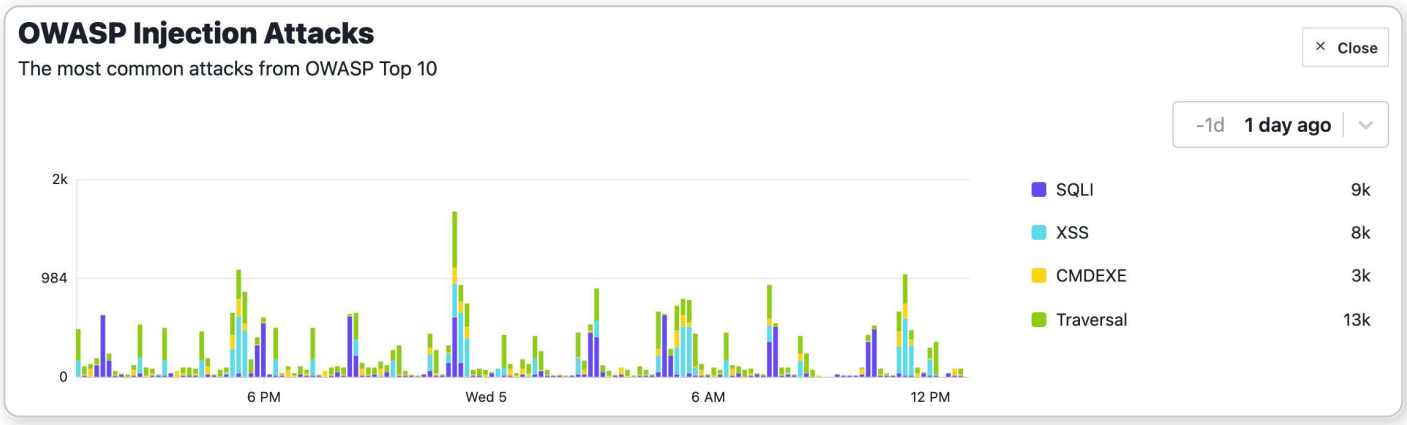
For more information about creating rules, see Rules.

Data Storage and Sampling

When our agent sends requests to our collectors, we store two types of data: **timeseries data** and **individual request data**.

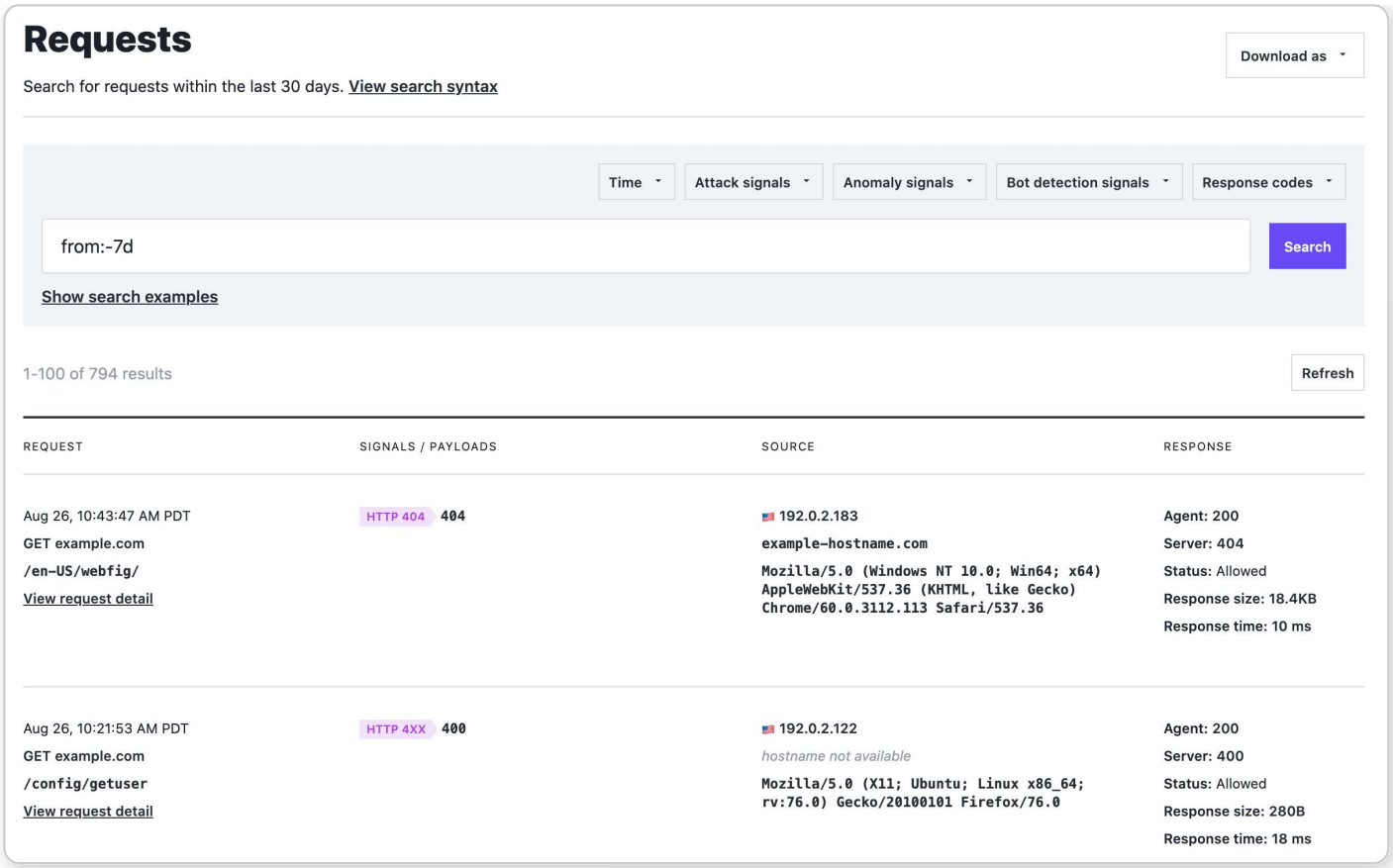
Timeseries data

Timeseries data counts the number of signals (e.g., XSS, SQLi, 404s) observed per minute, while individual request data includes individual records of anonymized requests. Timeseries data powers graphs visible throughout the product, as well as metrics such as tallies of request types.



Individual request data

While all timeseries data is stored and available in the product, a representative sample of individual request data is stored. Individual request data provides detailed information about specific requests, such as the originating IP address and request parameters:



What data does Signal Sciences store?

We store all timeseries data sent to our collectors (powering graphs and metrics throughout the product).

We store individual request data based on the type of signals that requests are tagged with or the way that custom rules are configured. Storage categories include:

- **All:** all requests matching this storage category are stored and available for reference throughout the console.
- **Sampled:** a random sample of requests matching this storage category will be stored and available for reference throughout the console.
- **Timeseries only:** requests matching this storage category aren't stored. Timeseries data for all signals tagged to the request will be stored and visible.
- **Not stored:** requests matching this category aren't stored.

> **Note:** Timeseries-only data storage category is only available on agents 3.12 and above. Matching requests processed on earlier agents will be processed according to the Sampled data storage category.

Request signal type	Description	Storage category
Individual requests containing attack signals	Any requests containing 1 or more attack signals (e.g., SQLi, XSS)	All
Individual requests containing CVE signals	Any requests containing 1 or more CVE signals applied by virtual patching rules	All
Individual requests containing only anomaly signals	Requests that contain only anomaly signals (e.g., 404, Tor traffic) but no attack or CVE signals	Sampled
Individual requests containing custom signals	Requests containing custom signals but no attack or CVE signals. See Custom Signals for more information about creating and using signals.	Sampled
Individual requests containing only API or ATO templated rules signals, known as informational signals	Requests which are tagged with only a specific set of API or ATO templated rules signals, and no custom, anomaly, attack, or CVE signals	Timeseries only
Individual requests that aren't tagged with a signal	Requests containing no signals	Not stored

> **Note:** Any requests containing at least one attack or CVE signal will be stored,

> including requests that also have anomaly, informational, or custom signals.

Heroku Install

The Signal Sciences agent can be deployed with Heroku. The installation process is compatible with any of the language buildpacks.

Installation

1. Log in to Heroku.

```
heroku login
```

2. Add the Signal Sciences buildpack to your application settings.

```
heroku buildpacks:add --index 1 https://dl.signalsciences.net/sigsci-heroku-buildpack/
```

> **Note:** The Signal Sciences buildpack must run first or before your application's primary buildpack.

3. In your `Procfile` file, add `sigsci/bin/sigsci-start` so it precedes your existing start command:

```
web: sigsci/bin/sigsci-start YOUR-APPLICATION-START-COMMAND
```

Example:

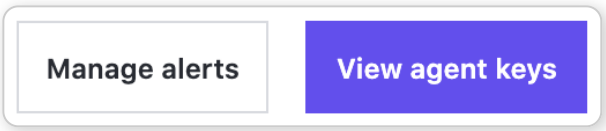
```
web: sigsci/bin/sigsci-start node index.js
```

4. Locate the **Agent Keys** for your Signal Sciences site:

   a. Log in to the Signal Sciences console.

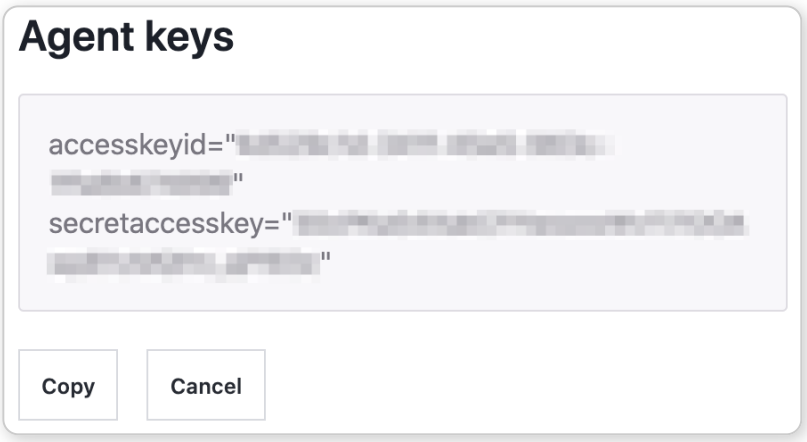
   b. Select a site if you have more than one site.

   c. Click **Agents** in the navigation bar. The agents page appears.



   d. Click **View agent keys**. The agent keys window appears.

   e. Copy the **Agent Access Key** and **Agent Secret Key**.



5. Add the Signal Sciences agent keys to your application's environment variables.

```
heroku config:set SIGSCI_ACCESSKEYID=access-key-goes-here
heroku config:set SIGSCI_SECRETACCESSKEY=secret-key-goes-here
```

6. Deploy your application. Heroku applications are typically deployed with the following commands:

```
git add .
git commit -m "my comment here"
git push heroku main
```

Configuration

- Each time you deploy your application, Heroku will automatically assign a new random name for the agent. An agent name for each deployment can be specified by setting the `SIGSCI_SERVER_HOSTNAME` environment variable:

```
heroku config:set SIGSCI_SERVER_HOSTNAME=agent-name
```

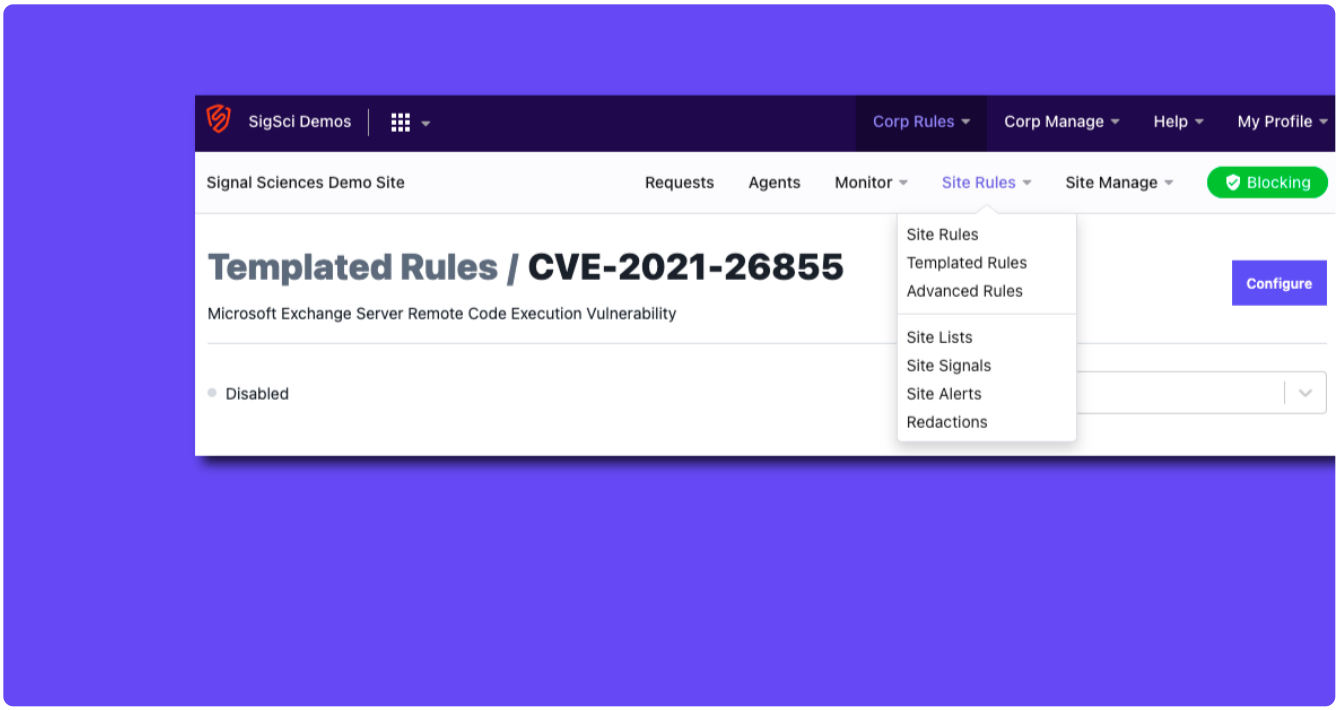
- Agent access logging can be enabled by setting the `SIGSCI_REVERSE_PROXY_ACCESSLOG` environment variable:

```
heroku config:set SIGSCI_REVERSE_PROXY_ACCESSLOG /tmp/sigsci_access.log
```

- The buildpack will install the latest version of the Signal Sciences agent by default. You can specify which agent version to install by setting the `SIGSCI_AGENT_VERSION` environment variable:

```
heroku config:set SIGSCI_AGENT_VERSION=1.15.3
```

Additional configuration options are listed on the agent configuration page.

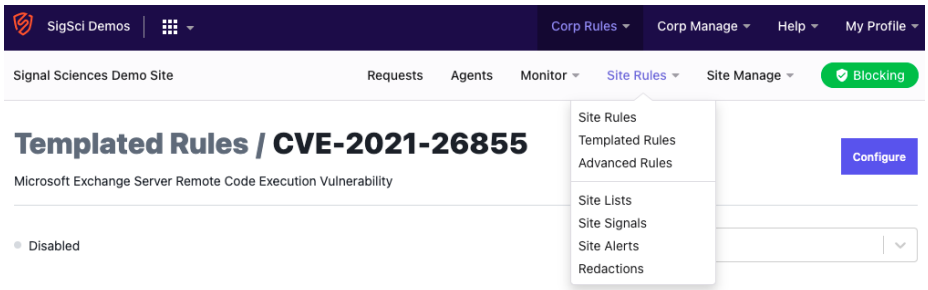


INFORMATION SECURITY, WEB APPLICATION SECURITY

Next-Gen WAF protection for recent Microsoft Exchange vulnerabilities

Protecting our customers

Our security research team has built and deployed a rule to protect Fastly's Signal Sciences Next-Gen WAF customers against the recently announced Microsoft Exchange Server vulnerabilities. The custom rule is available in the console under "Templated Rules".



We strongly suggest that customers using Signal Sciences Next-Gen WAF in front of their Microsoft Exchange servers enable this rule as soon as possible and configure it to block requests if the signal is observed. Additionally, follow all guidance from Microsoft to patch affected systems. The vulnerabilities in question are actively being exploited globally and have severe impact.

Patching Microsoft Exchange systems

We are seeing a large uptick in exploitation attempts in the wild. This is an evolving story and our teams are working continuously to ensure the rules are catching the latest attacks, but this should not be your only line of defense. We strongly recommend that you patch affected systems, perform incident response, and follow recommendations from Microsoft.

Exploit chain

The observed attacks on Microsoft Exchange systems chain together multiple CVEs (Common Vulnerabilities and Exposures) to carry out the attack. The impact of these attacks range from full system takeover through Remote Code Execution (RCE), as well as email inbox exfiltration and compromise. At a high level, the exploit chain is carried out as follows:

1. A Server-Side Request Forgery (SSRF) vulnerability in Microsoft Exchange Server identified as CVE-2021-26855 allows attackers to send HTTP requests to the exposed Exchange server and access other endpoints as the Exchange server itself. This is an unauthenticated step of the attack which makes the vulnerability exceptionally easy to exploit.
2. An insecure deserialization vulnerability identified by CVE-2021-26857 leverages the SYSTEM-level authentication obtained by the above SSRF attack to send specially-crafted SOAP payloads which are insecurely deserialized by the Unified Messaging Service. This gives the attacker the ability to run code as SYSTEM on the Exchange server.

3. After CVE-2021-26855 is successfully exploited, attackers can then utilize CVE-2021-27065 and CVE-2021-26858 to write arbitrary files to the Exchange server itself on any path. This code that is uploaded by the attacker is run as SYSTEM on the server. Lateral movement, malware implanting, data loss, escalation, and more can be carried out through these vulnerabilities.

By enabling the Signal Sciences Next-Gen WAF templated rule, the first step in the exploit chain cannot be carried out. If you would like to dig deeper into the technical details of this chain of attacks please see this post by the folks at Praetorian. To enable the templated rule, please refer to our documentation for details on how to enable templated rules.

Fastly Next-Gen WAF

Architecture and Deployment Overview

Web Application Firewall (WAF) | API security | Security

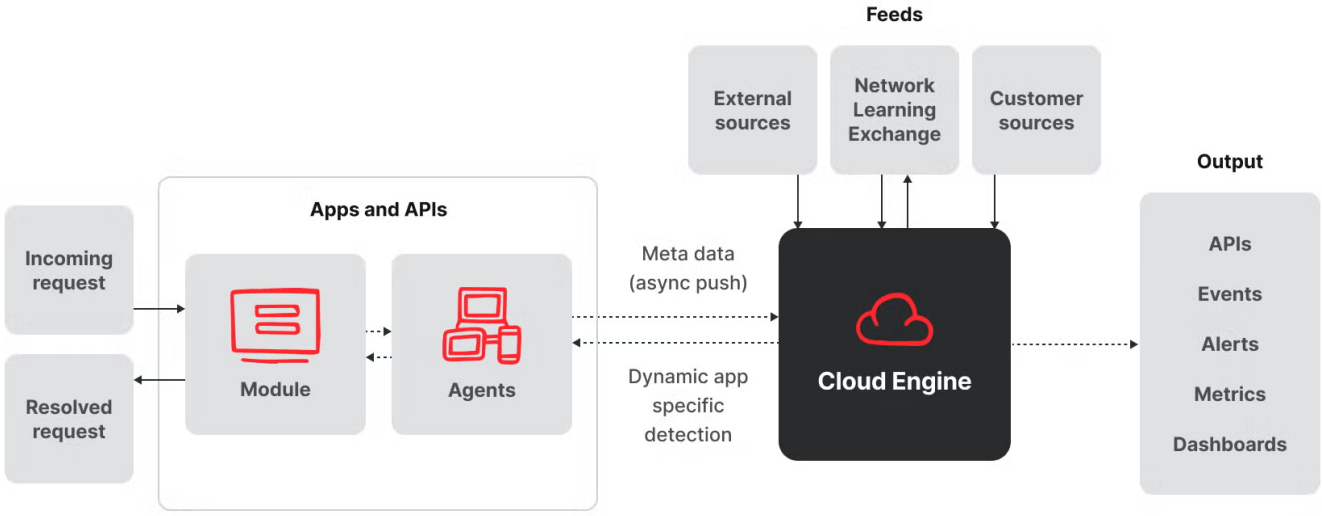
This datasheet provides detail into the highly performant, patented architecture of the Fastly Next-Gen WAF, as well as information on the wide array of deployment options available.

Unified web app and API security for any environment

Fastly offers the most flexibly deployed WAF on the market and can protect your apps and APIs wherever they are—in containers, on-premises, in the cloud, or at the edge—with one integrated solution. Gain comprehensive protection without sacrificing performance or requiring dedicated headcount: the Fastly Next-Gen WAF (powered by Signal Sciences) simply works out of the box and is so effective almost 90% of our customers run us in full blocking mode.

The Fastly Next-Gen WAF provides the proactive protection modern apps require while integrating into your DevOps and security toolchains for unparalleled visibility. Our flexible architecture can advance your application security strategy by providing developers, operations, and security teams insight into where and how your web applications and APIs are attacked.

Architecture overview



The Fastly Next-Gen WAF is a hybrid software as a service (SaaS) solution with three main components. This patented approach, developed by Signal Sciences, allows us to easily scale and protect even the highest volume applications and APIs without impacting performance.

Agents

Lightweight agents you deploy on your existing infrastructure to perform detection and decisioning against requests quickly and accurately.

Agents consist of a small daemon process and are designed to handle extremely heavy loads while making highly-performant and accurate detections and decisions locally. The agent also collects metadata about the malicious requests it has processed and shares that metadata with the Cloud Engine. We protect some of the highest volume sites on the Internet, where tens of thousands of agents collectively process trillions of production requests without impacting app or API performance. Agents block attacks before they hit applications or APIs and provide visibility into not only requests that come in but also server responses and anomalies that show how the application is behaving.

Modules

Optional but powerful component that pairs with our agents to enforce high performance and reliability.

Modules run on virtually any web server (NGINX, Apache, IIS, and more) or application language (.NET, Java, Python, PHP, .nodeJS, and more). The module is just a few hundred lines of code to ensure both reliability and extreme performance. Its sole job is to pass

requests through to the agent and receive and enforce decisions from the agent to allow the request through to the application or log/block it (depending on the mode set in the console).

Cloud Engine

Cloud-hosted analytics backend enriches the agent asynchronously with intelligence gathered from external and proprietary sources to make dynamic, application-specific detections.

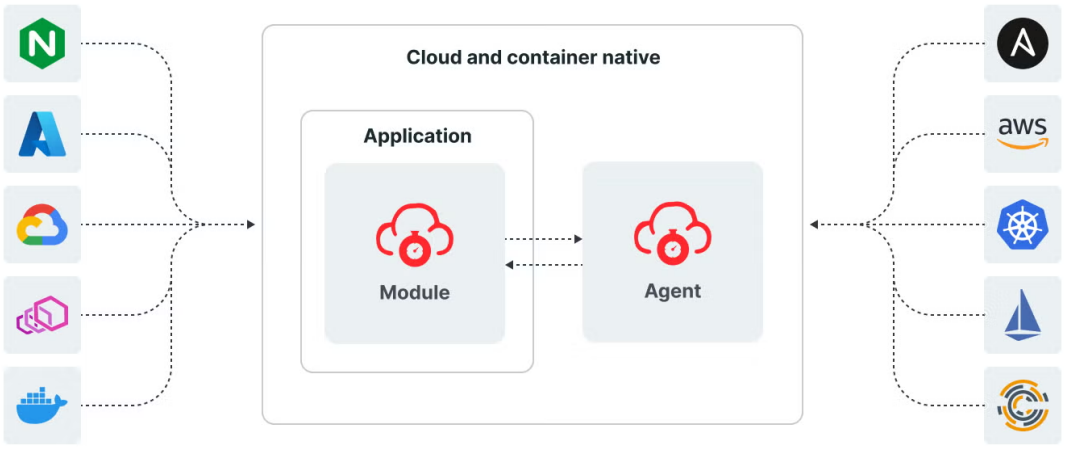
The Cloud Engine collects and analyzes anonymized attack data and telemetry from the many thousands of software agents across our customer base. The output from the Cloud Engine is used by the agent locally to perform better detection and make more aggressive blocking decisions. The agent decisioning is enhanced by our Network Learning Exchange (NLX) which shares confirmed malicious IP sources within the management console, alerting you to suspicious actors before they are a threat to your applications and APIs. Other feeds include external lists of malicious IPs and customers' custom IP lists, all of which provide additional request context that enriches the agent decisioning. This visibility and context is shared via our API and native integrations with the DevOps tools your team already uses, including Slack, PagerDuty, Jira, and more as well as security tools like Elastic and Palo Alto Networks Cortex XSOAR. Metrics and event reporting for your entire application footprint are also readily available via dashboards in a unified management console.

Deployment options

Native deployment options for data center, cloud, containers, and serverless.

Deployment Option #1: Cloud and container-native

Agent-module pair installs at your web server, API gateway, or at the app level within minutes. Our agent is infrastructure-agnostic which provides you the flexibility to deploy where you need it, without worrying about dependencies on underlying languages or frameworks.



Deploying in Kubernetes and service mesh

New application tools and frameworks, such as Kubernetes, are quickly moving companies into a DevOps-focused world. Companies now release code faster than ever before and Fastly offers flexible deployment options to fit within your container strategy with three “layers” where you can install our WAF in Kubernetes and four methods for how you deploy. Additionally, our native integrations with Envoy Proxy and Istio service meshes mean Fastly provides visibility into both north-south (client-server) and east-west (service-to-service) requests.

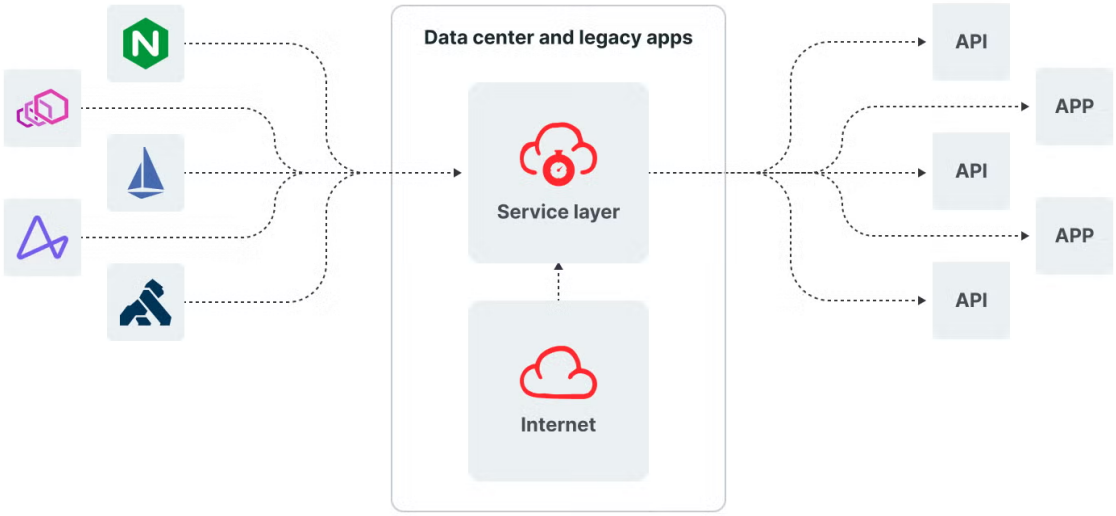
Install Method	Layer 1: Ingress Controller	Layer 2: Mid-Tier Service	Layer 3: App Tier
Agent + module in same app container	✓	✓	✓
Agent + module in different containers	✓	✓	✓
Agent in reverse proxy mode in same container as app	✓	✓	✓
Agent in reverse proxy in sidecar container	✓	✓	✓

Fastly fully supports deployments for:

kubernetes docker envoy Istio

Deployment Option #2: Data center and legacy application

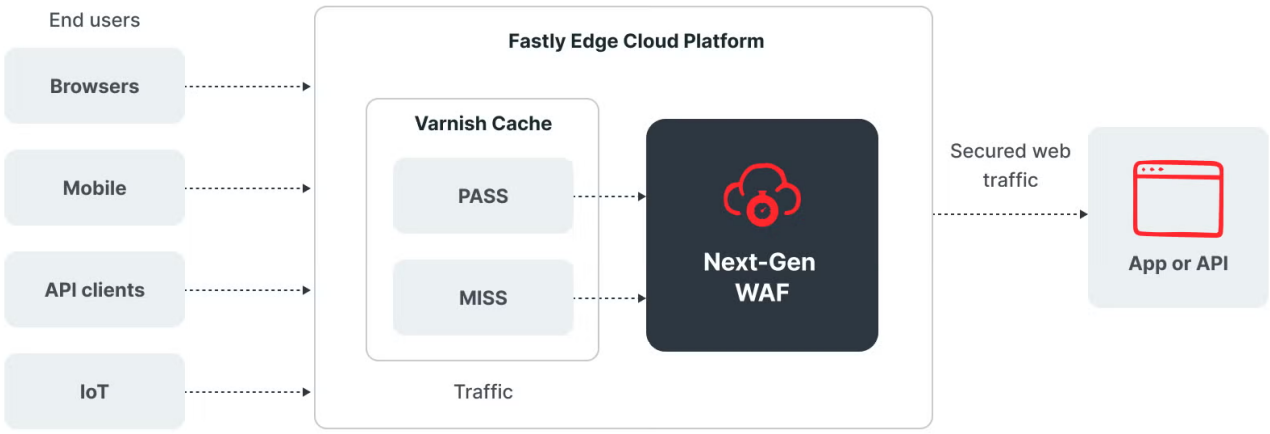
Customers who need protection for legacy applications or those deployed in data centers typically choose one of two deployment options: install the Fastly Next-Gen WAF to inspect traffic prior to web requests reaching the app or API endpoint, or install our agent in reverse proxy mode. For example, our module can be installed at the load balancer (A10 Networks, HAProxy, NGINX) or at the API gateway (Ambassador, Kong, Cloudentity). For customers with requirements that don't allow for installation at the load balancer or API gateway, our agent can be deployed in reverse proxy mode. Either deployment option provides the same level of visibility and actionable insights and alerts as our other deployment options with full feature parity.



Deployment Option #3: At the edge

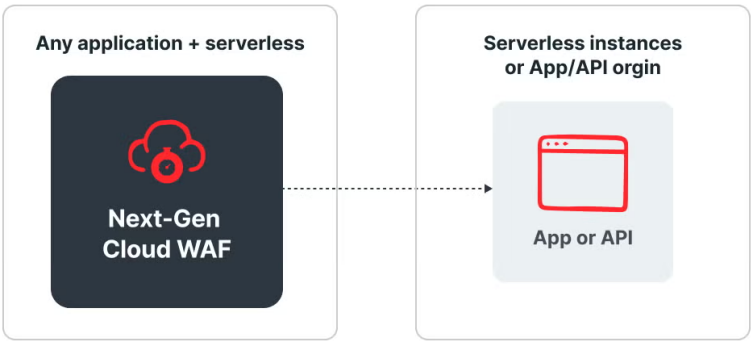
The Fastly Next-Gen WAF is available on the Fastly Edge Cloud Network, allowing customers to enforce security controls as part of Fastly delivery services. The edge cloud deployment option is seamlessly integrated with Fastly's caching layer, Varnish.

This provides protection and acceleration closer to users and shields origin systems from abusive attack traffic while delivering world-class performance. Our edge deployment is ideal for customers who are unable to install software on existing infrastructure and for those who want to take advantage of the performance benefits of Fastly's global content delivery network (CDN). This deployment option also offers additional features including Layer 3 and 4 always-on DDoS protection and TLS management.



Deployment Option #4: Cloud WAF

Cloud WAF empowers you to quickly and easily protect web applications, APIs, microservices, and serverless applications—without installing software on your infrastructure. Once deployed, a simple DNS change to point application traffic to Cloud WAF is all that's needed to enable the visibility and protection of the Fastly Next-Gen WAF for your applications. All web requests are redirected to our cloud enforcement layer where bad requests are detected and blocked. All good, legitimate traffic is then forwarded to your application origin server. Cloud WAF is ideal for customers wanting to add an easy-to-manage WAF without making upstream changes to their CDN layer.



Protection that's committed to data privacy

Many leading financial services firms, healthcare companies, and others with strict data privacy requirements all utilize Fastly's next-gen WAF because of our strong architecture built for data privacy. All sensitive data is handled entirely within the customer environment and only sanitized and redacted portions of requests that are marked as attacks or anomalies are then sent to the Fastly Cloud Engine.

Once the agent identifies a potential attack or anomaly in a request, a set of fully customizable redactions are applied locally and then the agent sends only the redacted individual parameter of the request which contains the attack payload, as well as a few other non-sensitive or benign portions of the request, such as client IP, user agent, URI, etc. Our backend only collects the response's metadata e.g. response codes, sizes, and times. We provide customers the ability to fully customize redaction policies and fields as needed. For additional protection, Fastly automatically enforces redaction of common sensitive data types—such as passwords, keys, GUIDs, and any type of PII or PHI—before the request is sent to our backend.

Betterment

"It works straight out of the box, scales automatically, and does a great job at providing visibility while securing the

> application.”
>
> Anson Gomes
> Lead Security Engineer

DevOps and security toolchain integrations

The best path to success for effective application and API protection is to provide the same baseline of security data to development, operations, and security teams in the tools they’re already using. Fastly works with the industry’s best tools and platforms to provide real-time alerting into your DevOps and security toolchains and to ensure it’s easy for your teams to leverage our production security telemetry within your organization’s current tools and processes for further investigation and analysis.

Out-of-the-box technology integrations help teams make or continue their transition to modern development models and architectures. Our single-click integrations include the most common development and operations alerting engines, chat-ops, project management, and incident tracking systems.

Technology and platform integrations

Run the Fastly Next-Gen WAF anywhere

Web servers

Microsoft IIS · NGINX · APACHE · Apache Tomcat

IAAS

Google Cloud · aws · Microsoft Azure

PAAS

HEROKU IBM Cloud Red Hat VMware Tanzu Microsoft Azure

Containers

OPENSHIFT Apache MESOS kubernetes docker envoy Istio

Config Management

CHEF Progress puppet SALTSTACK Red Hat Ansible Terraform

Feed integrations & partners

Send and receive data from the Fastly Next-Gen WAF

DevOps toolchain

DATADOG Jira Software Microsoft Teams Opsgenie

PagerDuty slack PivotalTracker VictorOps

SIEM/SOAR

DATADOG elastic McAfee

ArcSight NETWITNESS cisco splunk> Stack

CORTEX XSOAR Radar sumo logic PagerDuty

Fastly Next-Gen WAF

GraphQL Inspection

API security | Security | Web Application Firewall (WAF)

The Fastly Next-Gen WAF provides the ability to parse GraphQL requests, enabling visibility and protection against GraphQL attacks.

Visibility and protection for your GraphQL APIs

The evolution of API development has played a key role in the explosive growth of dynamic applications. Organizations are seeking more efficient ways to speed up release cycles, and this velocity is changing the CI/CD pipeline, from how releases are organized (from waterfall to agile development, for example) all the way down to the architecture level.

While REST and SOAP have played a major role in how APIs are written, GraphQL is quickly becoming a developer favorite for its efficiency, speed, and specificity.

The Fastly Next-Gen WAF (powered by Signal Sciences) provides advanced protection for your applications, APIs, and microservices, wherever they live, from a single unified solution. With the addition of GraphQL Inspection, we have expanded our coverage over your APIs, no matter what architecture or specification your developers use.

While REST has set the standard for API development over the past two decades, organizations are quickly adopting GraphQL as a more efficient way to make calls

that power their applications.

Setting a new standard for API protection

The strength of GraphQL over REST is that it allows the caller to request the exact information they need without returning any extraneous data. And this is done with a single call instead of making multiple roundtrips to the backend which decreases overall server strain.The increased adoption of the open-source query language GraphQL demonstrates how developers are moving away from REST-based APIs to a specification that is faster and more efficient. While this provides a better experience for both developers and the end-users, GraphQL is still subject to OWASP API Top 10 attacks, as well as GraphQL-specific exploits.

The Fastly Next-Gen WAF offers coverage that blocks OWASP-style injection attacks, denial of service attacks, and other vulnerabilities that can target GraphQL APIs. GraphQL Inspection gives your teams the ability to adopt newer technologies without taking on additional application security risk. With almost 90% of customers in full blocking mode, we provide comprehensive API protection that doesn't break your application or block legitimate requests.

Key benefits

Increased API attack visibility and coverage

As your organization increases its application footprint, the Fastly Next-Gen WAF provides scalable and comprehensive protection across all of your Layer 7 assets with full feature parity. Teams can apply their existing WAF rules to GraphQL requests and block attacks, or create custom rules to specifically handle GraphQL traffic.

Protection where your developers are

Help your developers to deploy applications using the tools they want, without slowing down release cycles. GraphQL Inspection allows organizations to write APIs safely while giving development teams the freedom and flexibility to work within the languages that fit into their workflows.

Operational efficiency

GraphQL Inspection enables you to reap the benefits of GraphQL's efficiency improvements without putting your applications at risk.

Blocking GraphQL attacks

GraphQL Inspection for the Fastly Next-Gen WAF offers several ways for you to take advantage of our patented approach to security:

Automation

As a part of our out-of-the-box solution, GraphQL Inspection will parse GraphQL requests and inspect the contents of the request within context. If an OWASP-style attack is present, we will block it automatically without any additional setup and configuration required.

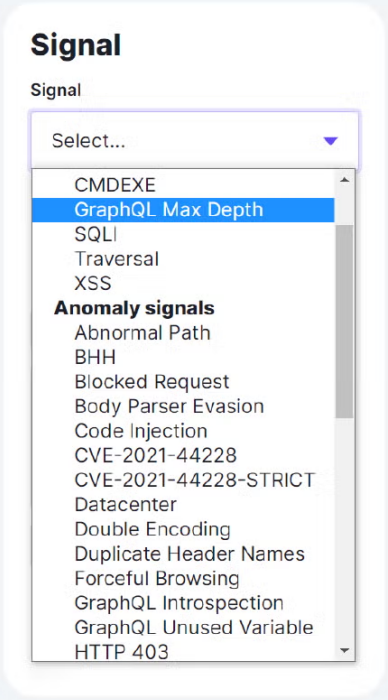
Custom Signals

Fastly provides GraphQL-specific signals in the console for customized protection based on user configuration. With these signals, you can define rules to route requests when certain thresholds or events happen. Some of these signals are included below.

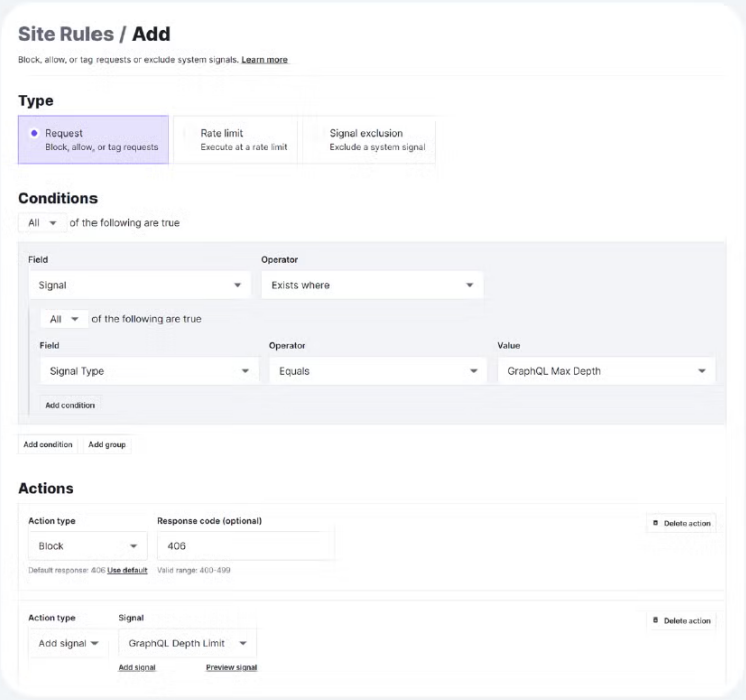
A signal is a descriptive tag about a request. It makes certain characteristics of the request visible and apparent – this includes what types of attack payloads a request contains or whether the request was blocked.

Templated Rules

Includes a templated rule around requests for a GraphQL IDE. GraphQL attacks are now visible via the console.



Image[1]



Image[2]

Common GraphQL vulnerabilities and Next-Gen WAF anomaly signal examples

OWASP API Security Top 10

- Injection Attacks
  - SQL Injection
  - Command Injection
  - Server-side Request Forgery
- Denial of Service
- Broken Access Controls

GraphQL-specific

- Max Depth - complex queries that crash servers
- Introspection - public data exposure through queries
- Unused variables - sign of an attack within request

Edge deployment

The Edge deployment method allows you to add the Next-Gen WAF as an edge security service onto Fastly's Edge Cloud platform without needing to make any modifications to your own hosting environment.

Limitations and considerations

Keep in mind the following limitations when working with the edge deployment method:

- Adding the Next-Gen WAF via the edge deployment method to an existing Fastly service counts against the service chain limit.
- This feature works with backends defined in VCL services using the API, CLI, or web interface. Backend definitions defined manually in VCL or snippets can be supported by redefining them using the API, CLI, or web interface. This will offer validation and enable a number of features not available to VCL-defined backends, including shielding. Learn more about defining backends in our integration documentation.
- We automatically support VCL directors as long as they are defined using the Fastly API.

Deploying at the edge

To deploy at the edge, you will need a corp and at least one site to protect. Setup involves making calls to the Signal Sciences API. These API calls will add privileged dynamic VCL snippets to your service that enable inspection.

Creating the edge security service

Create a new edge security service by calling the edge deployment API endpoint. This API call creates a new edge security service associated with your corp and site. You will need to replace `${corpName}` and `${siteName}` with those of the corp and site you are adding the edge security service to. Your `${corpname}` and `${siteName}` are both present in the address of your Next-Gen WAF console, such as `https://dashboard.signalsciences.net/corps/${corpName}/sites/${siteName}`.

```
$ curl -H "x-api-user:${SIGSCI_EMAIL}" -H "x-api-token:${SIGSCI_TOK"
-H "Content-Type: application/json" -X PUT \
"https://dashboard.signalsciences.net/api/v0/corps/${corpName}/sites/${
```

Run this API call again for each site you want to deploy on.

If successful, you will receive an HTTP 200 response with a blank response body (`{}`). Once this is returned, wait 1-2 minutes to allow the edge resources to be created.

To confirm the Compute instance resources associated with the site have been created, query the `edgeDeployment` endpoint again using the following request:

```
$ curl -H "x-api-user:${SIGSCI_EMAIL}" -H "x-api-token:${SIGSCI_TOK"
-H "Content-Type: application/json" \
"https://dashboard.signalsciences.net/api/v0/corps/${corpName}/sites/${
```

The query should now return the appropriate Compute instance associated with the Next-Gen WAF site in the URL path with no services attached. To attach the appropriate service, refer to Mapping to the Fastly service.

```
{"AgentHostName":"se--${corpName}--{SiteUID}.edgecompute.app","Services
```

Mapping to the Fastly service

To map your corp and site to an existing Fastly service and synchronize the origins, follow these steps:

1. Using the curl command line tool, call the PUT edgeDeployment/${fastlySID} API endpoint in a terminal application:

   Unix-based | Windows-based

   ```
   $ curl -H "x-api-user:${SIGSCI_EMAIL}" -H "x-api-token:${SIG
   -H "Fastly-Key: ${FASTLY_KEY}" -H 'Content-Type: application/jso
   ```

```
"https://dashboard.signalsciences.net/api/v0/corps/${corpName}/si
```

This API call will create and activate a new service version with dynamic VCL snippets automatically added to the service. By default, the service will be activated and set to 0% traffic ramping. You can override those defaults by providing parameters in the JSON body:

- `activateVersion` - activate Fastly service version after clone. Possible values are `true` or `false` (unquoted). If not specified, defaults to `true`.
- `percentEnabled` - percentage of traffic to send to the Next-Gen WAF. Possible values are integers values `0` to `100` (unquoted). If not specified, defaults to `0`. This can be adjusted later. Check out Traffic ramping for details.

For example, to disable initial activation and set initial traffic ramping to 10%, add the curl parameter `-d '{"activateVersion": false, "percentEnabled": 10}'` to the usual call:

```
$ curl -H "x-api-user:${SIGSCI_EMAIL}" -H "x-api-token:${SIGSCI
-H "Fastly-Key: ${FASTLY_KEY}" -H 'Content-Type: application/json'
-d '{"activateVersion": false, "percentEnabled": 10}' \
"https://dashboard.signalsciences.net/api/v0/corps/${corpName}/site
```

This API call requires the Fastly-Key header for authentication. The Fastly API key must have write access to the Fastly service ID. This API call will create and activate a new service version with dynamic VCL snippets automatically added to the service.

2. Optionally, follow these steps again for each additional Fastly service that you want to deploy on.

   If your origins change, you will need to call the PUT edgeDeployment/${fastlySID}/backends API endpoint again to resynchronize the backends.

Re-mapping a Fastly service to a new site

To re-assign the Fastly service to a new site, follow these steps:

1. Using the curl command line tool, call the DELETE edgeDeployment/${fastlySID} API endpoint in a terminal application:

```
$ curl -v -H "x-api-user: ${SIGSCI_EMAIL}" -H "x-api-token: ${S
-H "Fastly-Key: ${FASTLY_KEY}" -H 'Content-Type: application/json'
"https://dashboard.signalsciences.net/api/v0/corps/${corpName}/site
```

This API call requires the Fastly-Key header for authentication. The Fastly API key must have write access to the Fastly service ID. This API call removes all backends from the edge deployment connected to the Fastly service and detaches the Fastly service from the edge deployment.

2. Using the curl command line tool, call the `PUT edgeDeployment` API endpoint in a terminal application with the new `${siteName}` to create a new edge security service. For example:

```
$ curl -H "x-api-user:${SIGSCI_EMAIL}" -H "x-api-token:${SIGSCI
-H 'Content-Type: application/json' -X PUT \
"https://dashboard.signalsciences.net/api/v0/corps/${corpName}/site
```

You can verify if a compute instance resource was successfully created by the above step by referring to Create the edge security service.

3. Using the curl command line tool, call the PUT edgeDeployment/{fastlySID} API endpoint in a terminal application to map the existing Fastly service to the new `${siteName}`. For example:

```
$ curl -H "x-api-user:${SIGSCI_EMAIL}" -H "x-api-token:${SIGSCI
-H "Fastly-Key: ${FASTLY_KEY}" -H 'Content-Type: application/json'
"https://dashboard.signalsciences.net/api/v0/corps/${corpName}/site
```

This API call will activate a new service version by updating the existing Next-Gen WAF VCL dynamic snippet with the new edge security service ID.

Synchronizing origins

IMPORTANT

Failure to synchronize origins may result in your traffic not being inspected properly. Requests sent to a backend that does not exist in the edge security service will be

served a `503 Unknown wasm backend` error. You can correct this issue by running an API call to properly sync origins after any changes.

Some conditions cause origin syncing to occur automatically:

- Site configuration changes
- Agent mode changes (e.g., blocking, not blocking)
- Enabling or disabling IP Anonymization
- Rule changes (e.g., request rules, signal exclusion rules, CVE rules)
- Rule list changes (only if the list is being used by a rule)
- IP addresses flagged

If you change your origins in the Fastly Console, you will need to take additional action to synchronize your changes using an API call. The API call makes sure origin changes applied in the Fastly Console are reflected in the edge security service. For example:

```
$ curl -v -H "x-api-user:${SIGSCI_EMAIL}" -H "x-api-token:${SIGSCI_
-H "Fastly-Key: $FASTLY_KEY" -H "Content-Type:application/json" -X PUT
"https://dashboard.signalsciences.net/api/v0/corps/${corpName}/sites/${
```

WAF execution

Once both API calls are completed, your service will automatically be set up with dynamic VCL snippets that control the execution of the Next-Gen WAF. A new service version will be created and activated containing the additional VCL snippets.

The edge security service runs in the `vcl_miss` and `vcl_pass` subroutines. Execution priority is set to a high value to enable compatibility with any other VCL snippets that may be in use.

Traffic ramping

You can control the amount of traffic inspected by the edge security service using the `Enabled` dictionary key. This value is available in the `Edge_Security` dictionary and is automatically created when you attach a delivery service.

The default value is 0, with numbers greater than zero representing a percentage of the traffic being inspected. This means that unless you change the value of the `Edge_Security` Edge dictionary, your WAF will be enabled but won't inspect any traffic. If the value is set to 100, all traffic (100%) will be passed through the edge security service. If the value is less than 100, a random sample of the specified percentage will be sent through the edge security service.

> **NOTE**
>
> The `Edge_Security` Edge dictionary no longer uses The `DISABLED` field. To control blocking and logging behavior of an edge security service or turn off agent configurations entirely, use the web interface instead.

Health checks

The edge security service includes a health check inside the `edge_security` function. Using the `backend.health` property, this health check will skip security processing entirely if the edge security service is unhealthy for any reason. The edge security service is modeled as an origin using the backend type and uses the same health check feature.

The health check works by sending a periodic probe every 15 seconds and checks for an HTTP status code 200 as an expected response. Should a check indicate an unhealthy service, all security processing will be skipped until the service becomes healthy again. It may take up to 60 seconds for all security processing to be skipped.

Determining if you already use health check logic

You can check if your service already uses health check logic by inspecting the value of the `x-sigsci-edgemodule` HTTP header, which is added to the request prior to being sent to the edge security service. If the value is greater than or equal to `1.6.0`, then your VCL includes the health check logic.

Enabling and testing health check logic

To enable the health check for an existing service, make sure your VCL is updated to the latest version by re-running the steps in our instructions for mapping to the Fastly service. Then, test the health check logic by toggling the Agent mode to **Off**. This will simulate an unhealthy state for the edge security service and processing will be skipped.

Kubernetes Agent + Module

Integrating the Next-Gen WAF agent

You can install the Next-Gen WAF agent as a sidecar into each pod or as a service for some specialized needs.

We recommend installing the Next-Gen WAF agent in Kubernetes by integrating the `sigsci-agent` into a pod as a sidecar. This means adding the `sigsci-agent` as an additional container to the Kubernetes pod. As a sidecar, the agent will scale with the app/service in the pod instead of having to do this separately. However, in some situations, it may make more sense to install the `sigsci-agent` container as a service and scale it separately from the application.

You can use the `preStop` container hook to slow the pod's shutdown and ensure drain timeouts are met.

```
preStop:
  exec:
    command:
      - sleep
      - "30"
```

By default, the agent prioritizes quick start up and performance readiness for preliminary inspection. However, quick startup isn't always desirable if you only want the agent to inspect traffic after loading your rules and configuration data. If you want to delay agent startup, consider configuring a startup probe.

Getting and updating the agent container image

An official `signalsciences/sigsci-agent` container image is available on Docker Hub.

Alternatively, if you want to build your own image or need to customize the image, then follow the sigsci-agent build instructions.

These instructions reference the `latest` version of the agent with `imagePullPolicy: Always`, which will pull the latest agent version even if one already exist locally. This is so the documentation does not fall out of date and anyone using this will not have an agent that stays stagnant. However, this may not suit your needs if you need to keep installations consistent or on a specific version of the agent. In these cases, you will need to specify an agent version. Images on Docker Hub are tagged with their versions and a list of versions is available.

Whether you choose to use the `latest` image or a specific version, there are a few items to consider to keep the agent up-to-date.

Using the `latest` container image

If you do choose to use the `latest` image, then you will want to consider how you will keep the agent up to date.

- If you have used the `imagePullPolicy: Always` option, then the latest image will be pulled on each startup and your agent will continue to get updates.
- Alternatively, you may instead choose to manually update the local cache by periodically forcing a pull instead of always pulling on startup:

  ```
  $ docker pull signalsciences/sigsci-agent:latest
  ```

  Then, use `latest` with `imagePullPolicy: Never` set in the configuration so pulls are never done on startup (only manually as above):

  ```
  - name: sigsci-agent
      image: signalsciences/sigsci-agent:latest
      imagePullPolicy: Never
      ...
  ```

Using a versioned container image

To use a specific version of the agent, replace `latest` with the agent version (represented here by `x.xx.x`). You may also want to change `imagePullPolicy: IfNotPresent` in this case as the image should not change.

```
- name: sigsci-agent
    image: signalsciences/sigsci-agent:x.xx.x
```

```
    imagePullPolicy: IfNotPresent
    ...
```

This will pull the specified agent version and cache it locally. If you use this method, then we recommend you parameterize the agent image, using Helm or similar, so it is easier to update the agent images later on.

Using a custom tag for the container image

It is also possible to apply a custom tag to a local agent image. To do this, pull the agent image (by version or use `latest`), apply a custom tag, then use that custom tag in the configuration. You will need to specify `imagePullPolicy: Never` so local images are only updated manually. After doing so, you will need to periodically update the local image to keep the agent up-to-date.

For example:

```
$ docker pull signalsciences/sigsci-agent:latest
$ docker tag signalsciences/sigsci-agent:latest signalsciences/sigsci-ag
```

Then use this image tag in the configuration:

```
- name: sigsci-agent
    image: signalsciences/sigsci-agent:testing
    imagePullPolicy: Never
...
```

Configuring the agent container

Agent configuration is normally done via the environment. Most configuration options are available as environment variables. Environment variables names have the configuration option name all capitalized, prefixed with `SIGSCI_` and any dashes (-) changed to underscores (_). For example, the max-procs option would become the `SIGSCI_MAX_PROCS` environment variable. For more details on what options are available, see the Agent Configuration documentation.

The `sigsci-agent` container has a few required options you will need to configure:

- Agent credentials (**Agent Access Key** and **Agent Secret Key**).

- A volume to write temporary files.

Agent credentials

You must set the `sigsci-agent` credentials with two environment variables:

- **SIGSCI_ACCESSKEYID**: The **Agent Access Key** identifies which site in the Next-Gen WAF console the agent is configured for.
- **SIGSCI_SECRETACCESSKEY**: The **Agent Secret Key** is the shared secret key to authenticate and authorize the agent.

Because of the sensitive nature of these values, we recommend you use the built in `secrets` functionality of Kubernetes. With this configuration, the agent will pull the values from the secrets data instead of reading hardcoded values into the deployment configuration. This also makes any desired agent credential rotation easier to manage by having to change them in only one place.

Use the `valueFrom` option instead of the `value` option to use the `secrets` functionality. For example:

```
env:
  - name: SIGSCI_ACCESSKEYID
    valueFrom:
      secretKeyRef:
        # Update my-site-name-here to the correct site name or sim
        name: sigsci.my-site-name-here
        key: accesskeyid
  - name: SIGSCI_SECRETACCESSKEY
    valueFrom:
      secretKeyRef:
        # Update my-site-name-here to the correct site name or sim
        name: sigsci.my-site-name-here
        key: secretaccesskey
```

The `secrets` functionality keeps secrets in various stores in Kubernetes. This guide uses the generic secret store in its examples, but you can use any equivalent store. Add the Agent secrets to the generic secret store using YAML similar to the following example:

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: sigsci.my-site-name-here
stringData:
  accesskeyid: 12345678-abcd-1234-abcd-1234567890ab
  secretaccesskey: abcdefg_hijklmn_opqrstuvwxy_z0123456789ABCD
```

You can also create these secrets from the command line using `kubectl` such as with the following example:

```
$ kubectl create secret generic sigsci.my-site-name-here \
  --from-literal=accesskeyid=12345678-abcd-1234-abcd-1234567890ab \
  --from-literal=secretaccesskey=abcdefg_hijklmn_opqrstuvwxy_z012345678
```

Additional information about Kubernetes `secrets` functionality can be found in the Kubernetes documentation.

Agent temporary volume

For added security, we recommended you execute the `sigsci-agent` container with the root filesystem mounted as read only. However, the agent still needs to write some temporary files such as the socket file for RPC communication and some periodically updated files such as geolocation data.

To accomplish this with a read only root filesystem, there needs to be a writeable volume mounted. This writeable volume can also be shared to expose the RPC socket file to other containers in the same pod.

You can create a writeable volume using the built in `emptyDir` volume type. This is typically configured in the `volumes` section of a deployment, as shown in the following example:

```
volumes:
  - name: sigsci-tmp
    emptyDir: {}
```

Containers will then mount this volume at `/sigsci/tmp`:

```
volumeMounts:
```

```
    - name: sigsci-tmp
      mountPath: /sigsci/tmp
```

The default in the official agent container image is to have the temporary volume mounted at `/sigsci/tmp`. If you need a different directory for the agent container, then you will need to configure the following options from their defaults to match the new mount location:

- `rpc-address` defaults to `/sigsci/tmp/sigsci.sock`
- `shared-cache-dir` defaults to `/sigsci/tmp/cache`

Next-Gen WAF agent with a web application and Next-Gen WAF module installed

This deployment example configures the example `helloworld` application to use the `sigsci-agent` via RPC and deploys the `sigsci-agent` container as a sidecar to process these RPC requests.

To configure Next-Gen WAF with this deployment type you must:

- Modify your application to add the appropriate Next-Gen WAF module, configured it to communicate with a `sigsci-agent` via RPC.
- Add the sigsci-agent container to the pod, configured in RPC mode.
- Add an `emptyDir{}` volume as a place for the `sigsci-agent` to write temporary data and share the RPC address.

Modifying and configuring the application container

The `helloworld` example is a language based module (Golang) that has already been modified to enable communication to the `sigsci-agent` via RPC if configured to do so. This configuration is done via arguments passed to the `helloworld` example application as follows:

- Listening address (defaults to `localhost:8000`).
- Optional Next-Gen WAF agent RPC address (default is to not use the `sigsci-agent`). Other language based modules are similar. Web server based modules must have the Next-Gen WAF module added to the container.

For this `helloworld` application to work with the `sigsci-agent` it must have the `sigsci-agent` address configured as the second program argument and the `sigsci-tmp`

volume mounted so it can write to the socket file:

```
...
      containers:
      # Example helloworld app running on port 8000 against sigsci-
      - name: helloworld
        image: signalsciences/example-helloworld:latest
        imagePullPolicy: IfNotPresent
        args:
          # Address for the app to listen on
          - localhost:8000
          # Address sigsci-agent RPC is listening on
          - /sigsci/tmp/sigsci.sock
        ports:
        - containerPort: 8000
        volumeMounts:
        # Shared mount with sigsci-agent container where the socket
        - name: sigsci-tmp
          mountPath: /sigsci/tmp
```

Adding and configuring the agent container as a sidecar

The `sigsci-agent` container will default to RPC mode with a Unix Domain Socket (UDS) file at `/sigsci/tmp/sigsci.sock`. You must have a temp volume mounted at `/sigsci/tmp` to capture this socket file and share it with the pod. You will need to configure the `sigsci-agent` to communicate via this UDS socket. You will also need to modify the deployment YAML from the example above by adding a second argument to specify the `sigsci-agent` RPC address of `/sigsci/tmp/sigsci.sock`.

> **NOTE**
>
> It is possible to use a TCP based listener for the `sigsci-agent` RPC, but we do not recommend this for performance reasons. If you need TCP (or UDS is not available, such as in Windows), then you can specity the RPC address as `ip:port` or `host:port` instead of a UDS path. In this case, the volume does not need to be shared with the app, but it does need to be created for the `sigsci-agent` container to have a place to write temporary data such as geodata.

Adding the `sigsci-agent` container as a sidecar:

```
...
      containers:
      # Example helloworld app running on port 8000 against sigsci-
      - name: helloworld
        image: signalsciences/example-helloworld:latest
        imagePullPolicy: IfNotPresent
        args:
          # Address for the app to listen on
          - localhost:8000
          # Address sigsci-agent RPC is listening on
          - /sigsci/tmp/sigsci.sock
        ports:
        - containerPort: 8000
        volumeMounts:
        # Shared mount with sigsci-agent container where the socket
        - name: sigsci-tmp
          mountPath: /sigsci/tmp
      # Next-Gen WAF agent running in default RPC mode
      - name: sigsci-agent
        image: signalsciences/sigsci-agent:latest
        imagePullPolicy: Always
        env:
        - name: SIGSCI_ACCESSKEYID
          valueFrom:
            secretKeyRef:
              # This secret needs added (see docs on sigsci secrets
              name: sigsci.my-site-name-here
              key: accesskeyid
        - name: SIGSCI_SECRETACCESSKEY
          valueFrom:
            secretKeyRef:
              # This secret needs added (see docs on sigsci secrets
              name: sigsci.my-site-name-here
              key: secretaccesskey
        # If required (default is /sigsci/tmp/sigsci.sock for the c
        #- name: SIGSCI_RPC_ADDRESS
        #  value: /path/to/socket for UDS OR host:port if TCP
        securityContext:
          # The sigsci-agent container should run with its root fil
          readOnlyRootFilesystem: true
```

```
        volumeMounts:
        # Default volume mount location for sigsci-agent writeable
        # NOTE: Also change `SIGSCI_SHARED_CACHE_DIR` (default `/si
        #       if mountPath is changed, but best not to change.
        - name: sigsci-tmp
          mountPath: /sigsci/tmp
```

> **NOTE**
>
> The above `sigsci-agent` configuration assumes that `sigsci` secrets were added to the system section above.

Adding the agent temp volume definition to the deployment

Finally, you will need to define the agent temp volume for use by the other containers in the pod using the built in `emptyDir: {}` volume type.

```
...
      volumes:
      # Define a volume where sigsci-agent will write temp data and
      # which is required with the root filesystem is mounted read
      - name: sigsci-tmp
        emptyDir: {}
```