fastly

WHITEPAPER

Escape the Black Box of Security Visibility with Signals

Dive deep into Fastly's signals technology and learn how to gain deeper insights and create granular workflows from your application traffic.

Introduction

The Fastly Next-Gen WAF has been at the leading edge of security since 2014. Originally launched as Signal Sciences, we're pioneering a new paradigm in detection, blocking, and traffic analysis against application layer attacks for companies around the world.

A significant reason that our Next-Gen WAF (web application firewall) remains an award-winning solution in a decades-old, seemingly-stale industry is that we not only think about attack traffic in an entirely different way, but we also designed our solution to work with the way security practitioners naturally operate.

Legacy WAFs have historically placed their decisioning technology in a black box, preventing customers from understanding why traffic was blocked and limiting what they can do with it. Fastly lets security teams see and organize their traffic in an entirely different way. While legacy WAFs limit your vision and capabilities, our NGWAF (next generation WAF) shines a spotlight and lets you take the reins thanks to a feature we call **signals**.

This paper will cover four major aspects of signals: their origin, system signals, custom signals, and how signals can be utilized at the edge. This primer includes various types of signal examples, real-world use cases from our customers, and can serve as a guidepost for both experienced technologists wanting to learn more, and current customers who want to take their signals knowledge to the next level.

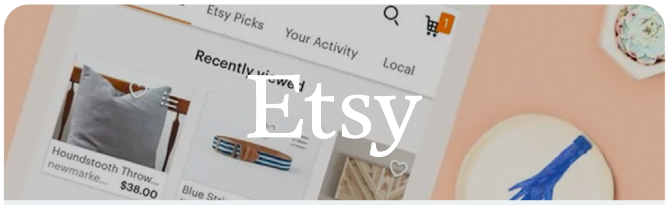
Table of Contents

1. Redefining Security Through Signals

As security practitioners, the founders at Signal Sciences knew firsthand that the WAF market was not keeping up with the pace of innovation. They saw their peers struggling to deploy manual solutions against sophisticated and innovative attackers. Security teams needed solutions that empowered them to be more agile, effective, and efficient against an ever-changing landscape.

After experiencing the faults of signature-based WAFs while at Etsy, the founders left and took an entirely different approach to building a new solution. One differentiating technology is SmartParse, which tokenizes each request coming through and looks at that hash to take a contextual understanding of the request. For example, regardless of whether the request is matching certain terms, is it in the format that would actually execute for cross-site scripting or SQL injection? This approach provides a much lower false-positive rate and is much faster than signature-based detection approaches.

Another differentiating technology and the focus of this paper is signals. A signals-based approach to web application security allows teams to gain more visibility and take greater control over their application traffic.



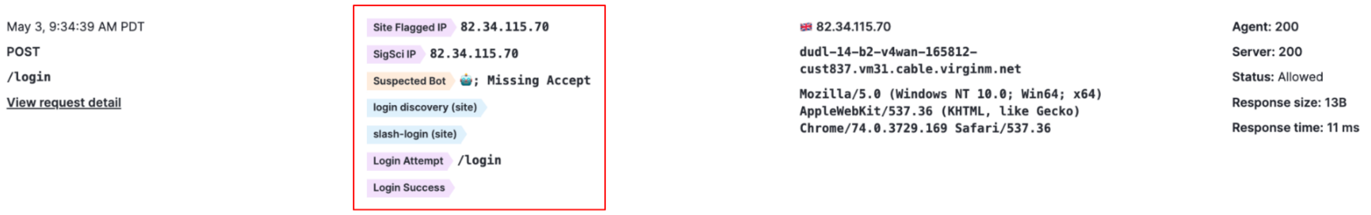
Etsy is a digital marketplace retailer that pioneered DevOps practices and digital transformation throughout its tech stack. Seeing a market need for a new way of securing applications and APIs, a group of security practitioners left Etsy and became the founders of Signal Sciences.

Defining Signals

A signal is a descriptive tag or label assigned to a traffic request as it's inspected. Requests are tagged with signals based on the logic of your active rules. Fastly's NGWAF has two types of signals:

1. **System signals**: common attack or anomalous web traffic requests that are labeled directly by Fastly, such as SQL Injection, XSS, or searchbot imposter.
2. **Custom signals**: web traffic requests that are created and labeled by customers per their unique application, such as homepage_search or 2023_blocklist_IPs.

Signals can be generated into charts, queried in logs, incorporated into custom rules, and more. It gives you a unique way to see your traffic data and take action on requests based on your defined criteria.



May 3, 9:34:39 AM PDT
POST
/login
View request detail

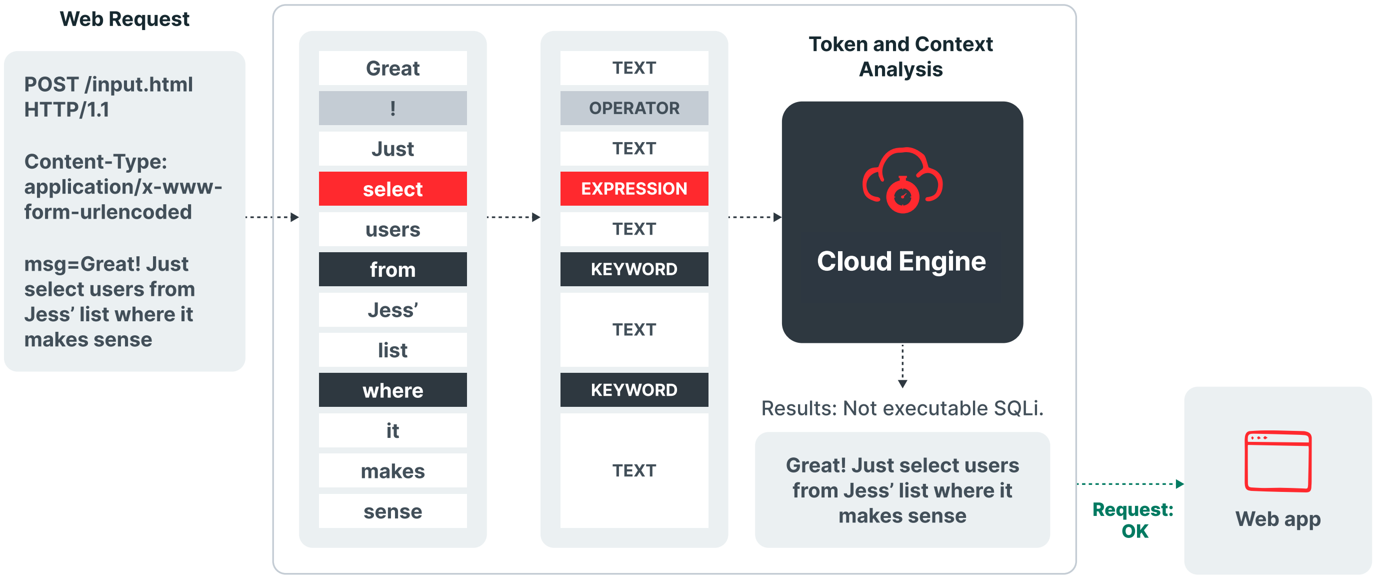
Site Flagged IP 82.34.115.70
SigSci IP 82.34.115.70
Suspected Bot ; Missing Accept
login discovery (site)
slash-login (site)
Login Attempt /login
Login Success

82.34.115.70
dudl-14-b2-v4wan-165812-cust837.vm31.cable.virginm.net
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36

Agent: 200
Server: 200
Status: Allowed
Response size: 13B
Response time: 11 ms

Examples of signals on an individual traffic request

Powering System Signals: Rejecting Regex with SmartParse


Web Request
POST /input.html HTTP/1.1
Content-Type: application/x-www-form-urlencoded
msg=Great! Just select users from Jess' list where it makes sense
Great
!
Just
select
users
from
Jess'
list
where
it
makes
sense
TEXT
OPERATOR
TEXT
EXPRESSION
TEXT
KEYWORD
TEXT
KEYWORD
TEXT
Token and Context Analysis
Cloud Engine
Results: Not executable SQLi.
Great! Just select users from Jess' list where it makes sense
Request: OK
Web app

SmartParse tokenizes web requests to provide more accurate detections

System signals, like SQL Injection or XSS, are automatically defined and maintained by our NGWAF. Failing to quickly and accurately detect these OWASP Top 10 types of attacks can leave your apps vulnerable by letting attacks through or blocking legitimate customers.

The power of system signals is only as good as our ability to accurately inspect and identify these requests: this is where SmartParse helps out.

Legacy WAFs have historically relied on regular expression (regex) signatures to power their inspection and decision-making processes. This means a WAF would inspect a request payload, and if parts of the payload match a signature of an attack or exploit, the WAF would automatically block it.

The "fuzziness" of regex rules - defining them perfectly to block bad requests and allow good ones - is an ongoing battle. For example, if an e-commerce customer wants to buy a British flag on an e-commerce site, for example, the search term Union Jack could be flagged as a SQLi attack and a legitimate user potentially blocked.

This leads to a lot of time spent tuning for false positives and ultimately lowers the trust that you may have for your WAF.

False positives are a timesink that can break applications and block legitimate users. Here are some innocuous phrases that we've seen flagged as an "attack" by regex-based WAFs.

- Is 300 lexus
- party NOT (baby and wedding)
- Oh and where is the key box?
- 20151114-Beta-Stop-Hunger-Now (31)-S.jpg
- Neurology & Sleep Specialist
- java lang courses
- Cozy and modern open space

SmartParse is the heart of our product because it informs how our system signals deliver information to you. It lays the foundation of trust that you have in our ability to deliver quick and accurate protection.

2. Diving into System Signals

System signals are powerful tools because they immediately identify the type of payload coming through the application. You can see individual requests on the console, then display these signals on your dashboard or generate reports to get a comprehensive look at your application's traffic trends. Let's explore system signals on a deeper level.

What are System Signals?

System signals are our "out of the box" signals that assign a tag based on inspecting a client's request. When requests are made to your web application, our NGWAF agent uses your active rules to identify which requests need to be tagged with a signal and then tags them with the appropriate signal.

We have two types of system signals:

- Attack
- Anomaly

With over three dozen system signals, our NGWAF has comprehensive coverage over the malicious traffic that matters most to your team. Below is a small sample of what we automatically define for you:

Sample listing of System Signals

Attacks	**Anomalies**
Log4J JNDI	Abnormal Path
GraphQL Max Depth	Datacenter Traffic
GraphQL Max Depth	Duplicate Header Names
Log4J JNDI	Malicious IP Traffic
GraphQL Max Depth	SearchBot Impostor

As you can see from the sample list, our NGWAF covers standard vectors like XSS and provides insight into potentially malicious IP traffic. But as tech stacks evolve and attack surfaces increase, our team also identifies and creates signals for more recent vulnerabilities such as **Log4Shell** and emerging technologies like **GraphQL** to ensure your coverage is up to date.

Signal Highlight: GraphQL

Developers are rapidly adopting GraphQL, an open-source standard query language, as an alternative to REST to meet the flexibility needed to maintain modern, high-growth APIs. Our NGWAF was one of the first WAFs to offer coverage through GraphQL Inspection, which detects, inspects, and blocks OWASP-style injection attacks, denial of service (DoS) attacks, and other vulnerabilities that target GraphQL APIs. We include GraphQL-specific signals in the console for customized protection based on user configuration.

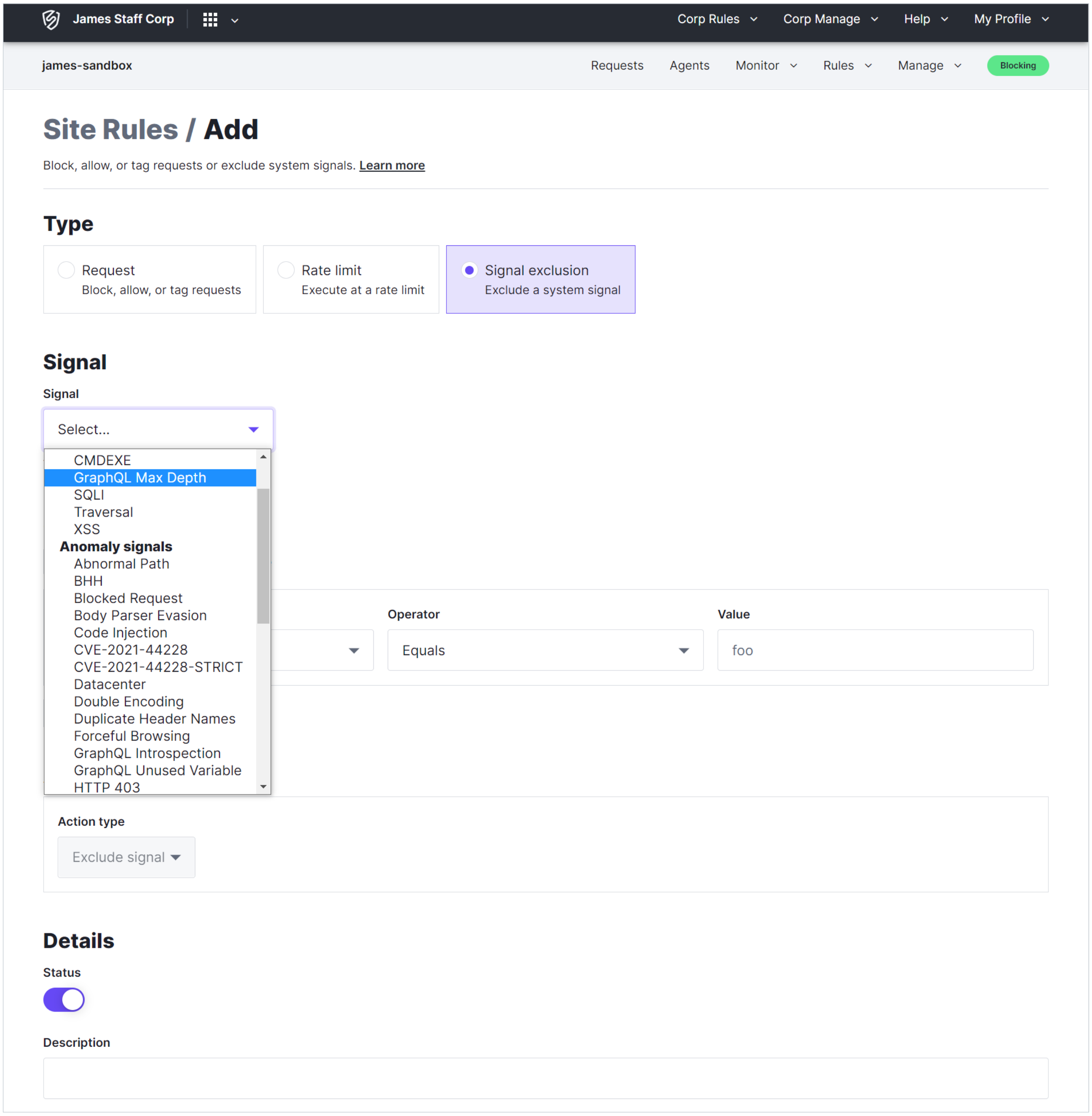
With these signals, you can define rules to route requests when certain thresholds or events happen. Some of these signals are included below.

GraphQL Signal	**Signal Description**
GraphQL Max Depth	A request has reached or exceeded the maximum depth allowed on the server for GraphQL API queries
GraphQL Introspection	An attempt to obtain the schema of a GraphQL API. The schema can be used to identify which resources are available, informing subsequent attacks.
GraphQL Undefined Variables	A request made to a GraphQL API containing more variables than expected by a function
GraphQL Duplicate Variables	A request that contains duplicated variables
GraphQL Missing Operation Name	A request has multiple GraphQL operations but does not define which operation to execute
GraphQL IDE	A request originating from a GraphQL Interactive Development Environment (IDE)

On the Fastly console, you can utilize GraphQL-specific signals to create a Request, Rate Limit or Exclusion rule to a Corp or Site.

Let’s take a look at an example of creating a Request Rule using the GraphQL Max Depth Signal to block traffic in order to avoid misuse or a potential DoS attack.

First, navigate to the site rules page, create signal exclusion rule. and select GraphQL Max Depth Signal from the signal drop down menu. In the below rule, we match on any requests that contain the "GraphQL Max Depth" signal and block them with the HTTP status code "406 Not Acceptable."



Site Rules / Add

Block, allow, or tag requests or exclude system signals. Learn more

Type

- Request — Block, allow, or tag requests
- Rate limit — Execute at a rate limit
- [✓ Signal exclusion — Exclude a system signal]

Signal

Signal: Select...

Operator: Equals

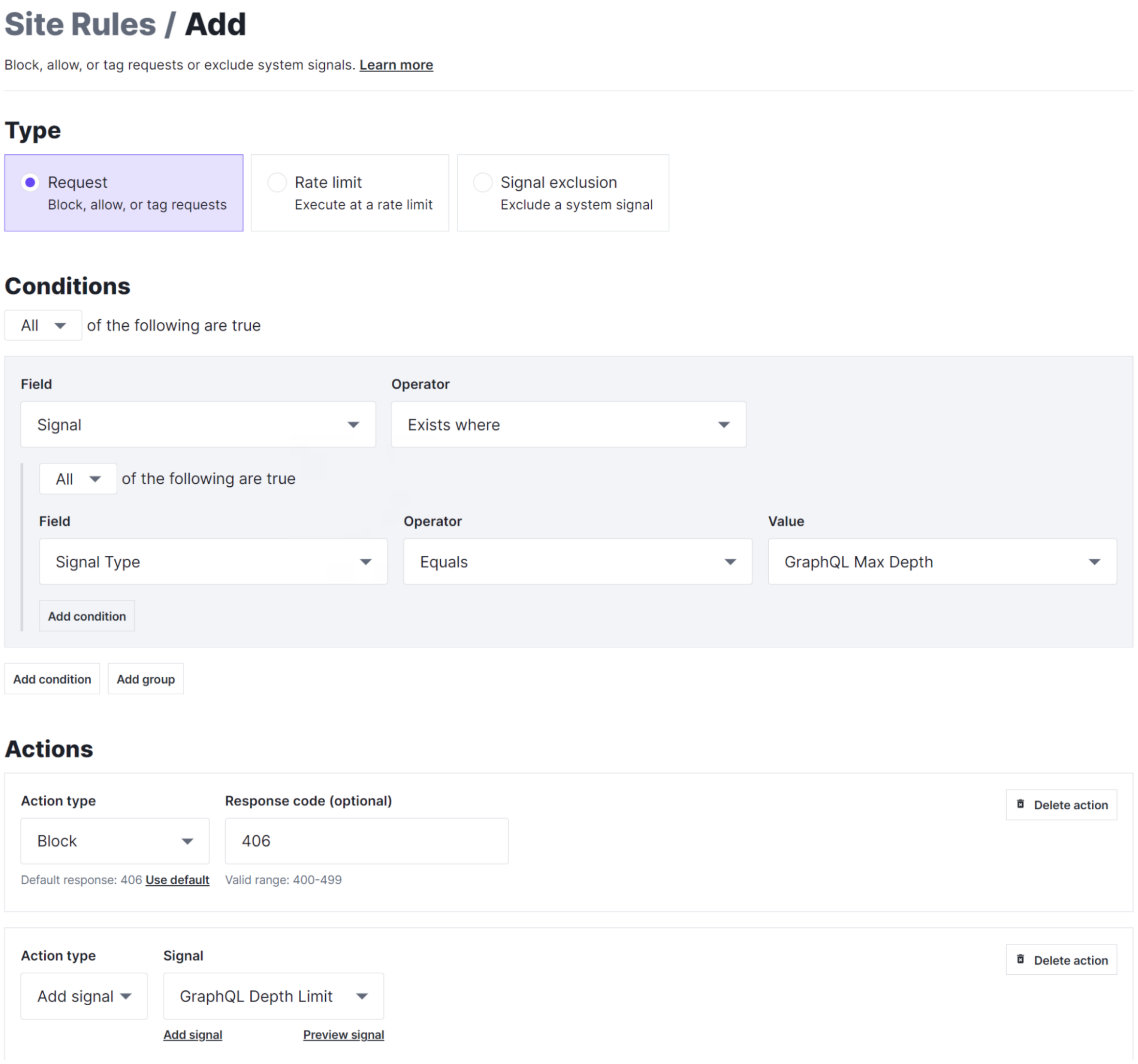
Value: foo

Action type: Exclude signal

Details

Status

Description


Site Rules / Add

Block, allow, or tag requests or exclude system signals. Learn more

Type

● Request — Block, allow, or tag requests

○ Rate limit — Execute at a rate limit

○ Signal exclusion — Exclude a system signal

Conditions

All of the following are true

Field	Operator	
Signal	Exists where	

All of the following are true

Field	Operator	Value
Signal Type	Equals	GraphQL Max Depth

Actions

Action type	Response code (optional)
Block	406
Default response: 406 Use default	Valid range: 400-499

Action type	Signal
Add signal	GraphQL Depth Limit

3. Custom Signals

Fastly's custom signals empower you to tag and organize your application traffic in a way that is tailored to your unique environment and use cases. You can tag interesting or anomalous traffic requests, making it easier to surface them to the broader team or make more precise blocking/allow decisions.

When Fastly sees a traffic request come through the network, you can configure rules to take one or more of of signals:

1. **Block:** terminate the traffic request
2. **Allow**: permit the traffic request
3. **Tag:** add a custom signal

A custom signal allows you to surface specific types of requests and make them visible on your console. Furthermore, you can create and enforce rules based on any traffic request that contains that tag.

Let’s dive deeper into how you can utilize custom signals in your organization.

The Power of Custom Signals

Traditional WAFs were created to stop malicious traffic from reaching origin servers, which served its purpose well during the internet age of HTML and PNGs.

But Layer 7 traffic has exploded over the past decade, with dynamic content spanning applications, APIs, microservices, and more. Coupled with modern DevOps practices and CI/CD it becomes very difficult for a traditional WAF to keep up.

When a traditional WAF detects a bad request, the security team is presented with only two options: block or allow. Like mail sorting at the post office, you want to let envelopes and packages through and keep out the damaged or torn items that can break your machinery. But sometimes you want to do more than that - like track how many envelopes are coming from San Francisco, or how many packages have biohazard warnings.

Once this information is tagged correctly, it can be used to inform the policies that sort and route items through your mail facility.

By surfacing which packages are biohazardous, you can direct them to a safer part of the post office.

To make this capability less abstract, let's walk through how a company can use custom signals to reduce traffic noise by blocking web crawlers and tagging anonymous email domains.

Reducing Traffic Noise with Custom Signals

The owner of **acmecorp.com** has noticed an uptick in vulnerability scans coming from a variety of internet sources. One thing they all seem to have in common is that they are sending web requests to the IP of their website and not the proper hostname acmecorp.com.

We know that real users type in "acmecorp.com", not its IP address "192.168.1.10", indicating that the requests are actually coming from non-human traffic, like web crawlers.

These malicious web crawlers cause unnecessary noise, data transfers, and resource consumption, which can ultimately result in a higher cloud spend. Additionally, the owner noticed that fraudulent accounts were being created using disposable/anonymous email generators, which can lead to fraudulent transactions and reduced revenue.

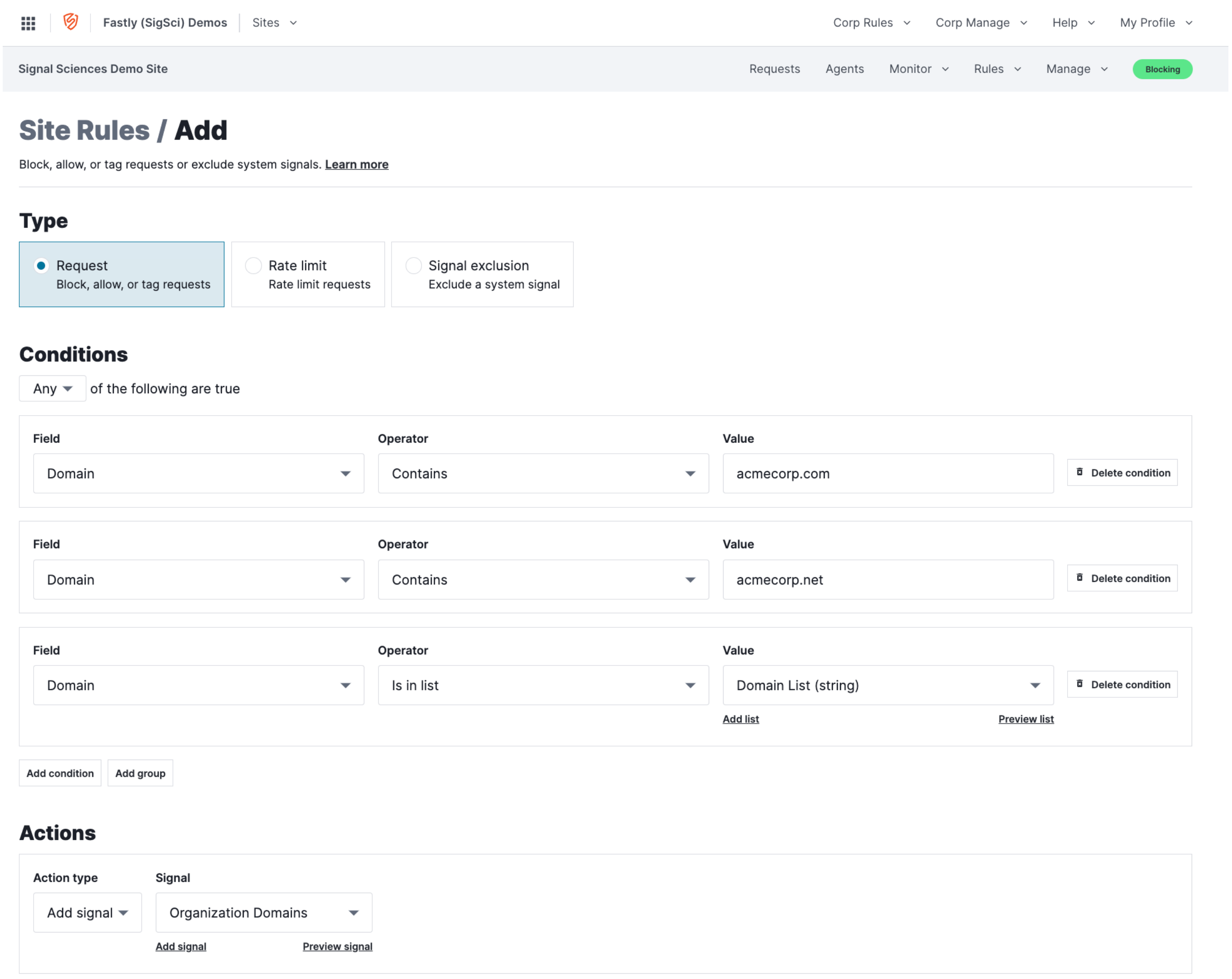
Our NGWAF’s custom signals can help by blocking web crawlers and tagging these anonymous email generators.

23% of security leaders think that keeping up with the volume of security alerts is their organization’s primary challenge.

- ESG research report: The rise of cloud-based security analytics and operations technologies

Custom Signal 1: Blocking Web Crawlers

First you can create a rule that tags any traffic that contains the domain acmecorp.com or acmecorp.net with the signal called Organization Domains.



Site Rules / Add

Block, allow, or tag requests or exclude system signals. Learn more

Type

● Request — Block, allow, or tag requests
○ Rate limit — Rate limit requests
○ Signal exclusion — Exclude a system signal

Conditions

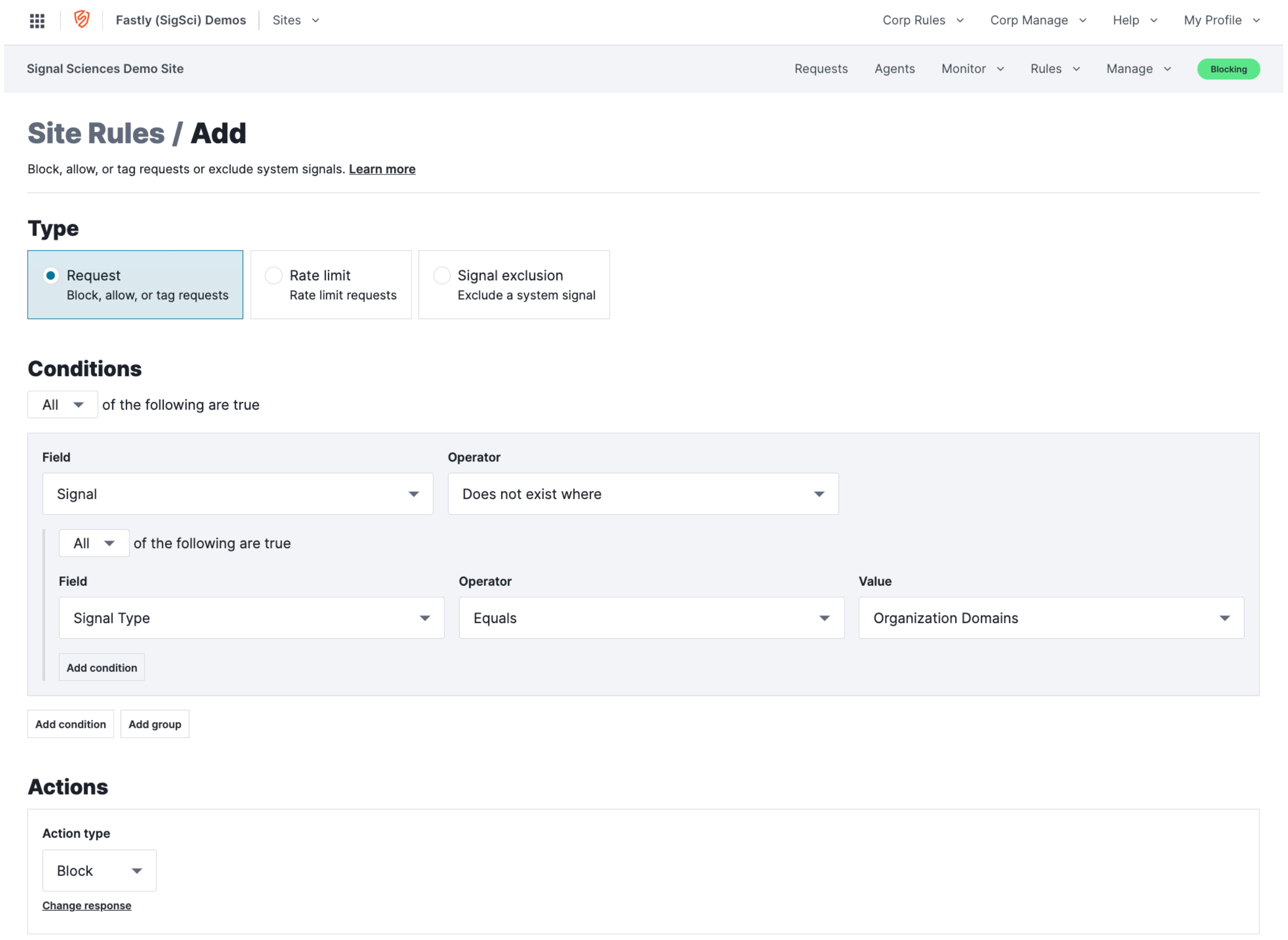
Any of the following are true

Field	Operator	Value
Domain	Contains	acmecorp.com
Domain	Contains	acmecorp.net
Domain	Is in list	Domain List (string)

Actions

Action type	Signal
Add signal	Organization Domains

Next, you can add a rule that states that any request that does not contain the signal Organization Domains would be blocked.



Site Rules / Add

Block, allow, or tag requests or exclude system signals. Learn more

Type

[✓ Request — Block, allow, or tag requests] Rate limit — Rate limit requests; Signal exclusion — Exclude a system signal

Conditions

All of the following are true

Field: Signal — Operator: Does not exist where

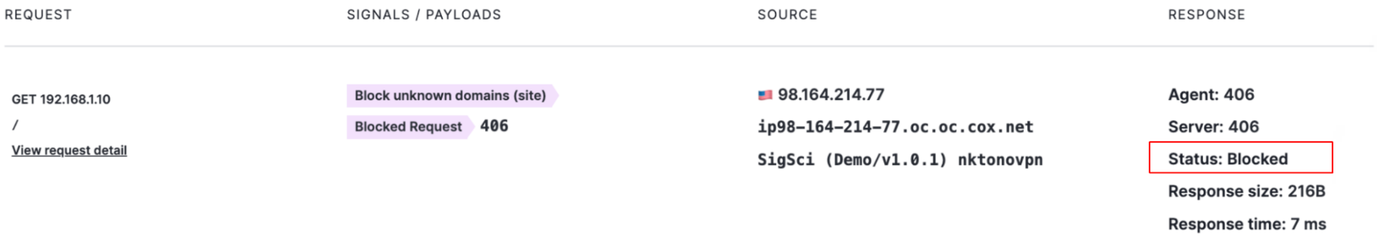
All of the following are true

Field	Operator	Value
Signal Type	Equals	Organization Domains

Actions

Action type: Block

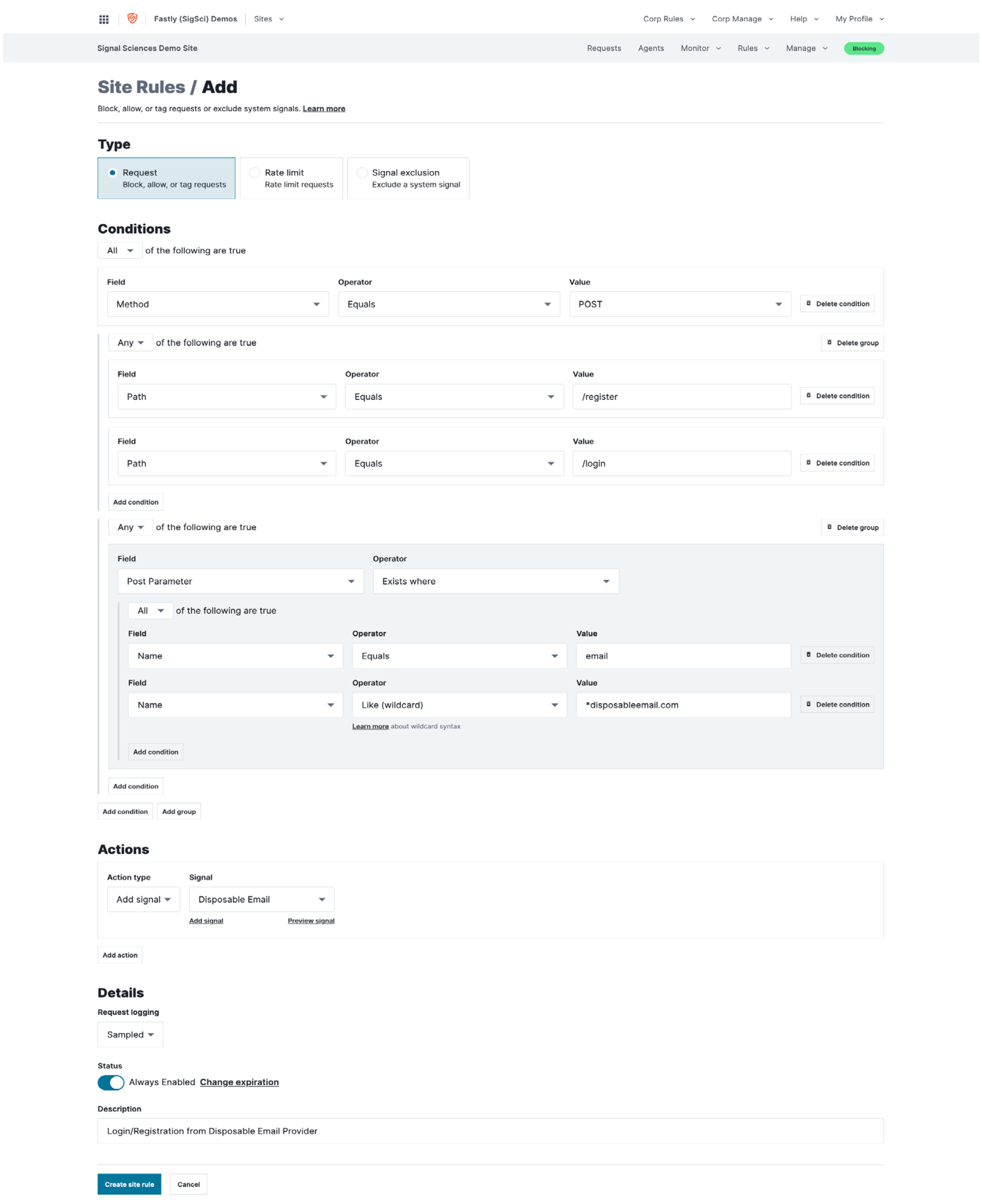
Now that the signal and rules are in place, you can go ahead and inspect the incoming traffic to see progress. On the dashboard, you can see that requests coming from a random bot crawler that doesn't know the name of the site are being blocked.



REQUEST	SIGNALS / PAYLOADS	SOURCE	RESPONSE
GET 192.168.1.10 / View request detail	Block unknown domains (site) Blocked Request 406	98.164.214.77 ip98-164-214-77.oc.oc.cox.net SigSci (Demo/v1.0.1) nktonovpn	Agent: 406 Server: 406 Status: Blocked Response size: 216B Response time: 7 ms

Custom Signal 2: Tagging Anonymous Email Domains

To track how many accounts are being created from anonymous email addresses, first you generate a rule that targets requests that successfully register and log into our sign-up form using a specific domain. You add a signal to those requests called Disposable Email. Just like the example above, you can create another rule that blocks any request that contains the Disposable Email signal.



Site Rules / Add

Block, allow, or tag requests or exclude system signals. Learn more

Type

- ● Request — Block, allow, or tag requests
- ○ Rate limit — Rate limit requests
- ○ Signal exclusion — Exclude a system signal

Conditions

All of the following are true

Field	Operator	Value
Method	Equals	POST

Any of the following are true

Field	Operator	Value
Path	Equals	/register
Path	Equals	/login

Any of the following are true

Field	Operator
Post Parameter	Exists where

All of the following are true

Field	Operator	Value
Name	Equals	email
Name	Like (wildcard)	*disposableemail.com

Learn more about wildcard syntax

Actions

Action type	Signal
Add signal	Disposable Email

Details

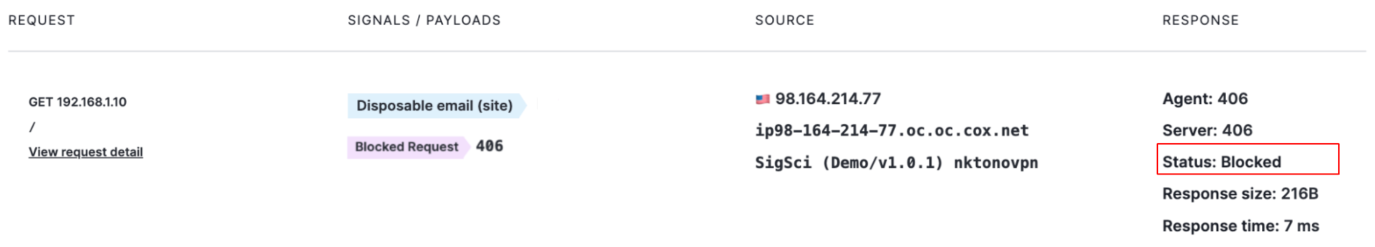
Request logging: Sampled

Status: Always Enabled Change expiration

Description: Login/Registration from Disposable Email Provider

fastly

Now when you look up the traffic containing the Disposable Email signal, you can see that it's blocked. This reduces fraudulent account registration and reduces resource consumption.



REQUEST	SIGNALS / PAYLOADS	SOURCE	RESPONSE
GET 192.168.1.10 / View request detail	Disposable email (site) Blocked Request 406	98.164.214.77 ip98-164-214-77.oc.oc.cox.net SigSci (Demo/v1.0.1) nktonovpn	Agent: 406 Server: 406 Status: Blocked Response size: 216B Response time: 7 ms

4. Signals at the Edge and more

The decisioning logic that inspects a potential attack was developed to fall in line with the way traffic requests are handled - when a client submits a request to visit a webpage, a WAF sits between the client and the origin server to make sure the request is valid (not an attack) before the origin can serve up the content.

The explosion of edge computing is bringing content closer to clients and end-users, serving content and enabling innovation faster than ever before. With our NGWAF sitting at the edge, we can enable more sophisticated blocking and rate-limiting by enforcing these policies at the edge.

In this next section, we'll now look at two additional use cases – identifying known actors and response tracking – and see how signals help in these areas to strengthen your security posture. We'll also look at how moving some of the security decisioning to Fastly's edge can further protect downstream systems through the use of custom response codes.

Identifying Known Actors

There is a constant drumbeat around preventing web threats, and for good reason: code injection attacks, customer data exfiltration, and API abuse are constant threats to service delivery on the web.

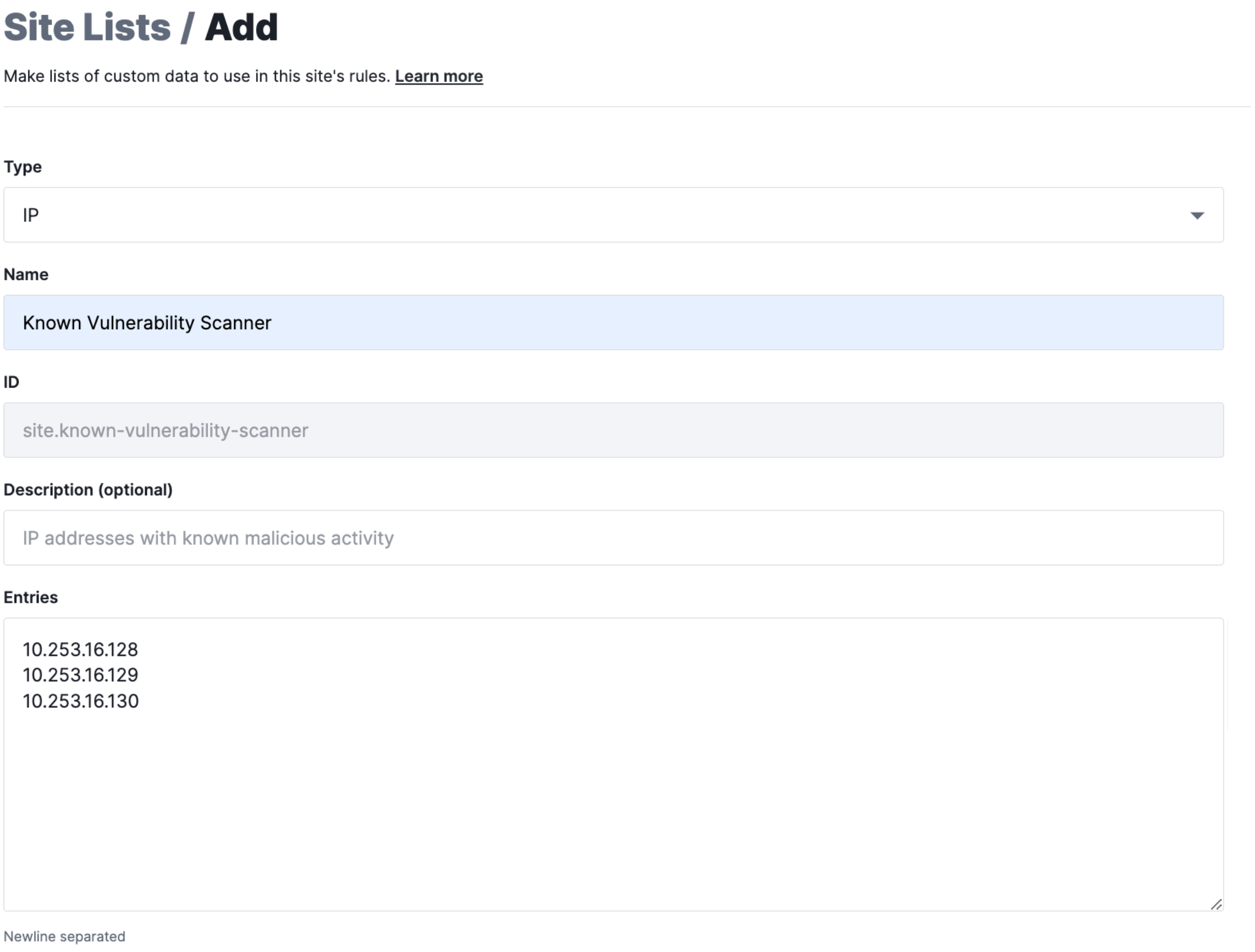
There is, however, another side to this coin that's often not addressed during threat modeling: reducing noise and alert fatigue for overburdened teams. When security teams are chasing false positives instead of actual threats, they're operating with reduced effectiveness.

According to a report by ESG, **23% of security leaders think that keeping up with the volume of security alerts is their organization's primary challenge**. One way to combat cybersecurity alert fatigue is by using your WAF to automatically identify known actors. By tagging traffic or requests with metadata that marks them as "good," more mental room is left for teams to maximize efficiency in their threat mitigation.

Using the NGWAF, you can add a specific signal to this category of traffic. This is done using an IP address or any other distinguishing identifier in the HTTP request. Here is how you might configure your site rules to add a custom signal to identify a popular scanner based on a list of IP addresses.

Once a list is created, the site rule itself is very straightforward. Let's outline the basic steps in the UI.

First, let's create a list that we'll use in the rule itself. In this example, we know the list of IP addresses to identify:



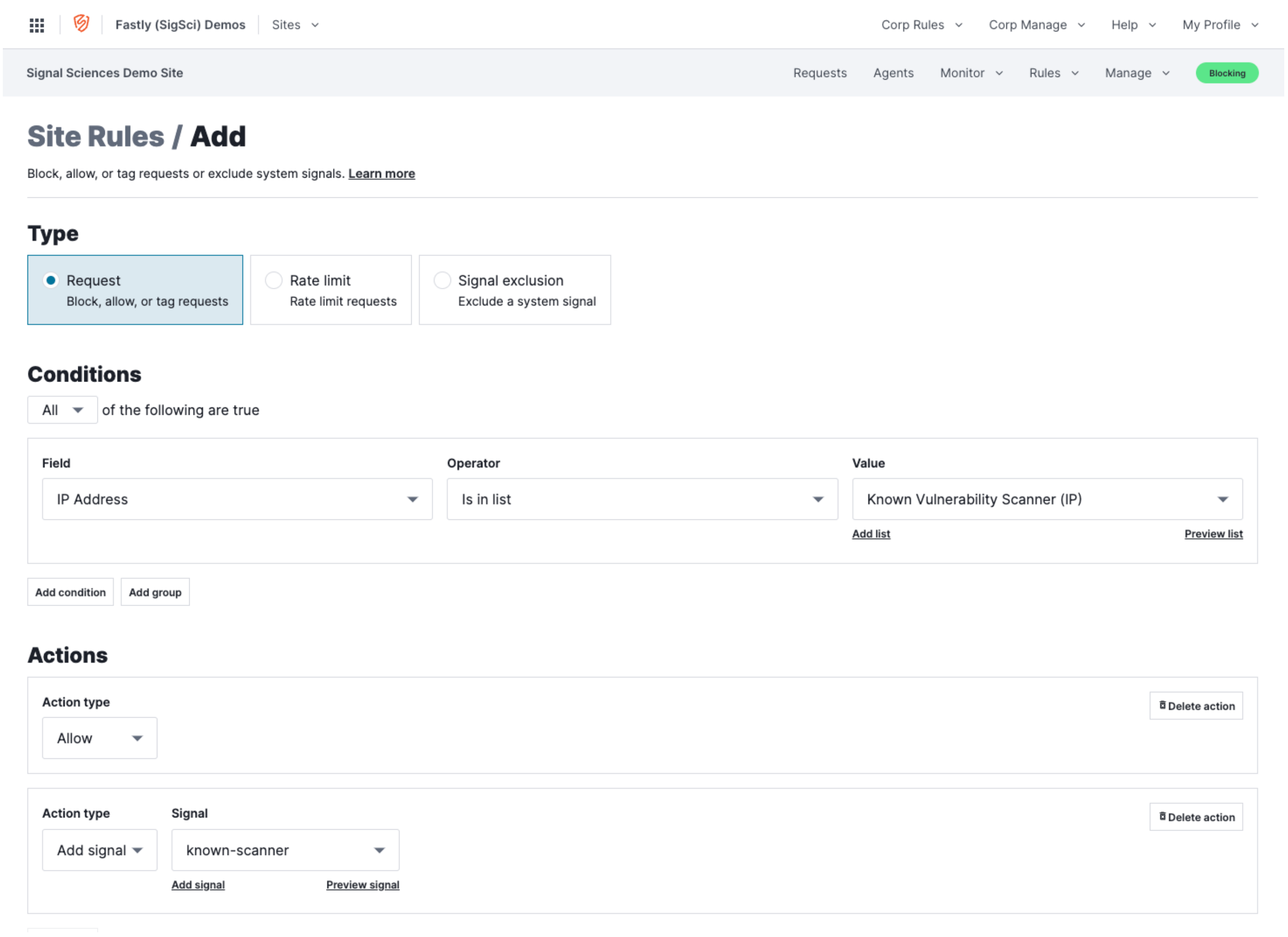
Site Lists / Add

Make lists of custom data to use in this site's rules. Learn more

Type	IP
Name	Known Vulnerability Scanner
ID	site.known-vulnerability-scanner
Description (optional)	IP addresses with known malicious activity
Entries	10.253.16.128 10.253.16.129 10.253.16.130

Newline separated

In the above screenshot, we've added three fictitious IP addresses representing known vulnerability scanners. Once the list is created, we can attach a signal to this category of traffic. This can be done by creating a new request rule as follows:



Site Rules / Add

Block, allow, or tag requests or exclude system signals. Learn more

Type

● Request — Block, allow, or tag requests
○ Rate limit — Rate limit requests
○ Signal exclusion — Exclude a system signal

Conditions

All of the following are true

Field	Operator	Value
IP Address	Is in list	Known Vulnerability Scanner (IP)

Add list · Preview list · Add condition · Add group

Actions

Action type: Allow — Delete action

Action type: Add signal — Signal: known-scanner — Add signal · Preview signal — Delete action

With this rule, we are allowing traffic from the IP addresses in this specific list and then tagging that request with the signal known-scanner for future reference. Without this rule, the NGWAF would perform attack detection on this traffic and block the scanner from doing its job, creating unnecessary alerts for your security team. With the signal known-scanner attached to the request, you can also confirm and visualize the activity you expect from the scanner.

Next, we'll discuss a more complex way of tracking attacks, by looking not only at the incoming payload but also at the effect the request had (or didn't have) on a downstream application.

Response Tracking

Basic approaches to web security threat modeling often appear to follow a simplistic storyline about a would-be attacker. First, they're wearing a dark hoodie (of course), and second, they possess a single, well-crafted payload that, if sent at the right part of the application (URL), will wreak havoc on your system -- likely in the form of data exfiltration, denial of service or other forms of abuse.

The second part of this story goes like this: If only we had a perfect way to catch this attack (expressed as a malicious HTTP request) and stop it before it did damage, we'd achieve some sort of web security bliss.

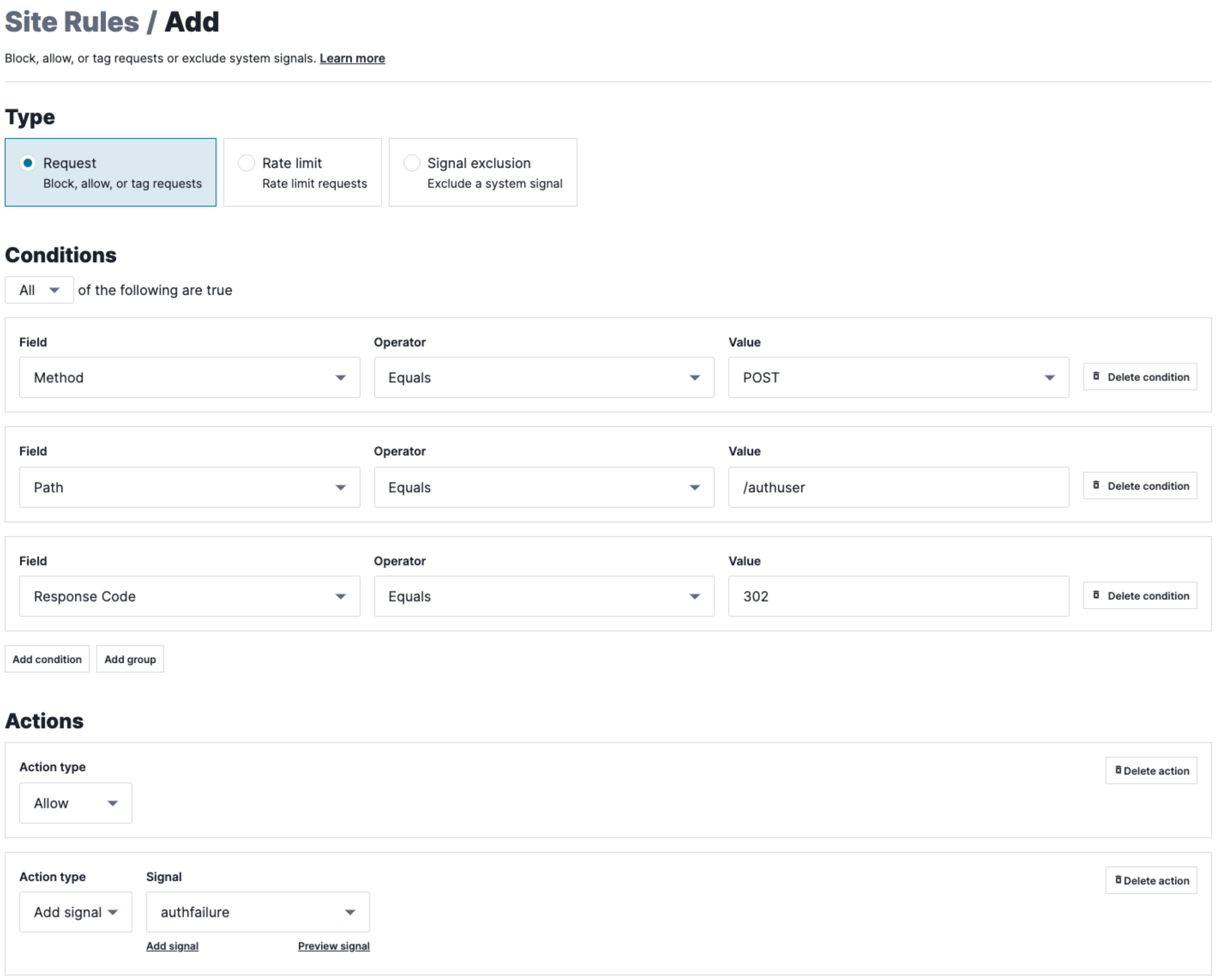
This thought process is the common first layer of defense for web application security. The consequence of this line of thinking is a model based on known threats or signatures. It's no wonder companies end up with the hundreds (or potentially thousands) of rules commonly found in legacy WAFs.

How do we improve on this? With custom signals, we can extrapolate this logic and go even further. In the NGWAF, both the request and response are used to determine whether an attack or anomaly signal is added to a request/response pair. For example, a good handful of our system signals (such as forceful browsing) rely on knowledge of the request and response before tagging a request with the signal.

This extends the protection from "fire and forget" types of attacks towards gaining visibility and insights into more sophisticated attacks. What if we could determine how attackers are probing for weaknesses in an authentication endpoint?

Authentication APIs are a common target for attacks because, by default, they need to be widely exposed to work properly. A custom signal to track authentication failures (in this case HTTP 302) can inform you of excessive login failures which could indicate an attempted credential-stuffing attack.

What follows is a basic template for how you would create a site rule for this:



Site Rules / Add

Block, allow, or tag requests or exclude system signals. Learn more

Type

● Request — Block, allow, or tag requests
○ Rate limit — Rate limit requests
○ Signal exclusion — Exclude a system signal

Conditions

All of the following are true

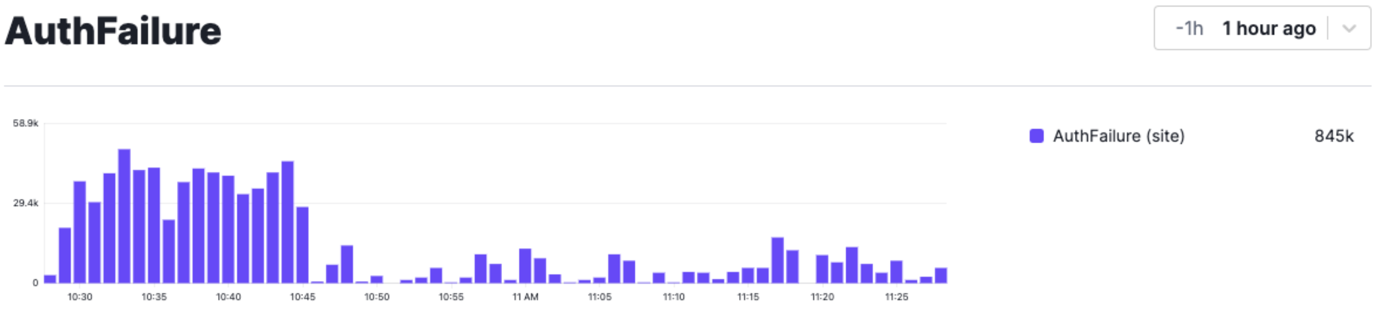
Field	Operator	Value
Method	Equals	POST
Path	Equals	/authuser
Response Code	Equals	302

Actions

Action type	Signal
Allow	
Add signal	authfailure

Here, we're tagging authentication failures with a signal AuthFailure. It's not clear that something bad has happened yet, but too many failures when you are not expecting them could indicate an attack in progress.

This type of leading indicator could alert you to add additional security controls to other parts of the application that handle authentication; it could indicate a potential coordinated attack. The signal can be graphed as follows with a custom dashboard:



In this example, we've mocked up a dashboard to track incidences of this signal over the last hour. You can see a higher concentration of signal activity between 10:30 and 10:45, which may indicate an attempted attack, especially if your known baseline is much lower.

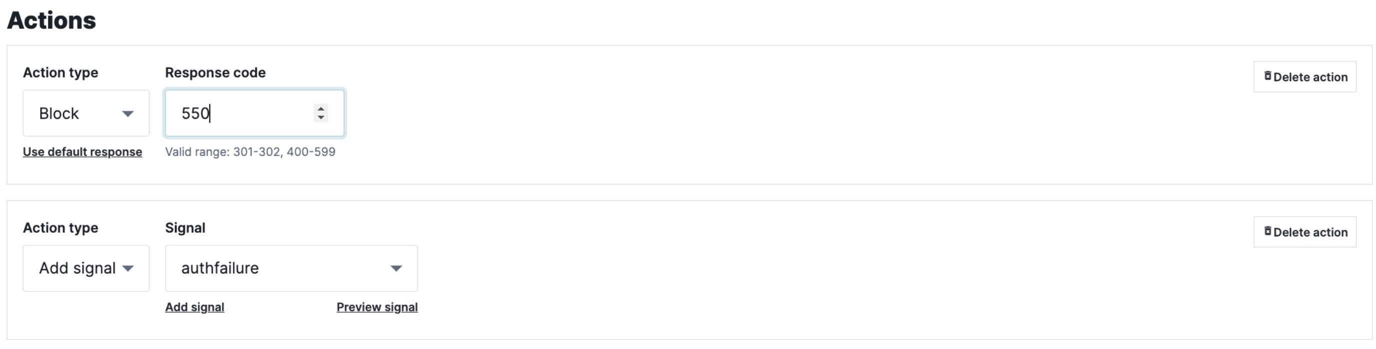
Extending Protection to the Edge

Tracking responses is great, but what about enforcement actions such as blocking or rate limiting? What if you want to perform these actions upstream? At this point, it seems like the opportunity has passed because the agent has already completed processing and the response is heading back to the client.

This is where extending your security perimeter to the edge can help further. Signals on the response can be passed back to the edge using custom response codes, which allows the intelligence generated by the NGWAF to turn into additional enforcement actions in Fastly's Delivery (Varnish) or Compute@Edge environments.

For Varnish customers, this is as simple as adding logic around the beresp.status variable. Compute@Edge customers could do something similar based on relevant variables for the response (which would be language dependent).

The beresp.status contains the HTTP status code of the response received from your origin. For example, if your site rule returned a 550 instead of a 401, your edge security decisions at this stage could include blocking, edge rate limiting, or tar pitting the client – all of which would have the effect of slowing the attacker down and keeping traffic and load away from your origin during an attack. You can enable the additional protection by updating the rule to change the rule to block and altering the response code as follows:



Actions

Action type	Block
Response code	550

Use default response — Valid range: 301-302, 400-599 — Delete action

Action type	Add signal
Signal	authfailure

Add signal — Preview signal — Delete action

Once the rule is changed in the NGWAF, additional security logic can be added to Varnish to take action when the value of beresp.status has a value of 493. The benefit of security enforcement at the edge increases as the sophistication and scale of the threat to your application increases.

Fastly's NGWAF can distribute security decisions in multiple places throughout your environment -- providing protection and decisioning in more places whether at the Edge, inside your applications themselves (Core deployment), or as a standalone reverse proxy (Cloud WAF).

Conclusion

Fastly's NGWAF is a pioneering solution in modernizing the technology used to detect, block, and analyze traffic against application layer attacks. Unlike legacy WAFs, which operate as black boxes, our NGWAF uses signals to provide visibility into why traffic was blocked, making the decision path searchable and organized.

Fastly's two types of signals, system signals and custom signals provide both out-of-the-box visibility into common web attacks and customized visibility into your specific application traffic. Additionally, signals can be used to enforce actions at the edge, keeping unwanted traffic and its load even further away from your applications. Using signals to surface what types of attack trends are hitting your apps or APIs helps you double down on your defensive protections in those areas. With signals, you can maintain strong security visibility across your entire portfolio of applications, leading to increased efficiency and productivity for your team.

Ultimately, Fastly's NGWAF is a powerful yet easy-to-use tool for any organization looking to protect its web apps and APIs against sophisticated attackers. To learn more and see for yourself how effective signals are, visit fastly.com or contact us.

Additional Resources

Fastly Next-Gen WAF Architecture and Deployment Overview
2022 Gartner® Magic Quadrant™ for Web Application and API Protection (WAAP)
Fastly Next-Gen WAF GraphQL Inspection Datasheet