GIGAOM

BENCHMARK/FIELD SERVICE

High Performance Application Security Testing v1.0

Product Evaluation: NGINX App Protect vs. ModSecurity (plus AWS Web Application Firewall)

JAKE DOLEZAL AND WILLIAM MCKNIGHT | SEP 17, 2020 - 2:50 PM CDT

CREDIT: PETE LINFORTH

SPONSORED BY NGINX Part of F5

GIGAOM

High Performance Application Security Testing

Product Evaluation: NGINX App Protect vs. ModSecurity (plus AWS Web Application Firewall)

TABLE OF CONTENTS

1. Summary

Data, web, and application security has evolved dramatically over the past few years. Just as new threats abound, the architecture of applications—how we build and deploy them—has changed. We've traded monolithic applications for microservices running in containers and communicating via application programming interfaces (APIs)—and all of it deployed through automated continuous integration/continuous deployment (CI/CD) pipelines. The frameworks we have established to build and deploy applications are optimized for time to market—yet security remains of utmost importance.

The challenge of securing and innovating is profound, and requires a lightweight and integrated security solution that won't impede performance and delivery. For example, DevOps teams need security controls that work across distributed environments without invasively slowing down or burdening the release cycle. The maturation of these controls and processes ultimately transitions into the realm of DevSecOps, where security is built into the CI/CD pipeline.

The multitude of deployed apps, APIs, and microservices produces a constant flow of communication and data among applications that requires active management—both internal and external. Apps themselves can vary greatly in the protocols, allowed methods, authorization/authentication schemes, and usage patterns. Perhaps most important, IT departments need granular control over the entire application ecosystem to prevent security breaches and attacks, be they man-in-the-middle, distributed denial of service, or script/code/SQL injection attacks.

While security is of utmost importance, the pace of modern business demands high performance, and this is especially true in application- and microservice-enabled enterprises. The conventional approach—deploying a perimeter Web Application Firewall (WAF) to protect applications by filtering and monitoring traffic between the app and the Internet—is no longer enough. Even internal communication between apps and microservices on the trusted corporate network can be compromised and must be addressed. A defense-in-depth strategy is needed with multiple WAFs.

This report focuses on web application security mechanisms deployed in the cloud and closer to your apps. The cloud enables enterprises to differentiate and innovate with microservices at a rapid pace, and allows microservice endpoints to be cloned and scaled in a matter of minutes. The cloud also offers elastic scalability compared to on-premises deployments, enabling faster server deployment and application development and less costly compute. However, the cloud is just as vulnerable, if not more so, to attacks and breaches as on-premises APIs and apps are.

Our focus is specifically on approaches to securing apps, APIs, and microservices that are tuned for high performance and availability. We define "high performance" as companies that experience workloads of **more than 1,000 transactions per second (tps)** and require a **maximum latency below 30 milliseconds** across the landscape.

Make no mistake, for many organizations, performance is a big deal—they need to ensure secured transactions at rates that keep pace with the speed of their business. A WAF or application security solution *cannot* be a performance bottleneck. Many of these companies seek a solution that can load balance across redundant microservices and enable high transaction volumes.

The numbers add up. If a business experiences 1,000 transactions per second, that translates into 3 *billion* API calls in a month. And it is not uncommon for large companies with high-end traffic levels to experience 10 billion or more API calls in a 30-day period. Make no mistake, performance is a critical factor when choosing an API security solution.

Benchmark Testing

In this report, we performance test three security mechanisms on NGINX: **ModSecurity**, **NGINX App Protect**, and **AWS Web Application Firewall (WAF)**. This last product was tested as a fully managed security offering. Note, ModSecurity is commercially distributed by NGINX and will be referred to as "ModSecurity" throughout the rest of this report.

In our benchmarks, NGINX App Protect outperformed ModSecurity at all tested attack rates. NGINX App Protect produced 92% lower latency than NGINX running ModSecurity at the 99th percentile at 1,000 transactions per second (tps) on the 5% bad request test. In our tests, the latencies for App Protect and ModSecurity diverged at the higher percentiles, becoming pronounced at the 95th percentile and above.

For fully managed offerings, NGINX App Protect produced 82% lower latency than AWS WAF at 1,000 tps on the 5% bad request test. Since AWS WAF is fully managed, we do not know what underlying compute resources are working behind the scenes, which makes an apples-to-apples performance comparison difficult. Once again, latency differences were minimal until the 90th percentile, with a significant difference witnessed at the 99th percentile and above.

On a single small 2 CPU and 5.25GB of RAM EC2 instance, we captured the maximum transaction throughput achieved with 100% success (no 5xx or 429 errors) and less than 30ms maximum latency. NGINX App Protect produced about 5,000 requests per second, compared to only 2,000 requests per second with ModSecurity. App Protect provides the same level of throughput as hitting the API directly without a WAF in between.

Testing hardware and software in the cloud is very challenging. Configurations may favor one vendor over another in feature availability, virtual machine processor generations, memory amounts, storage configurations for optimal input/output, network latencies, software and operating system versions, and the workload itself. Even more challenging is testing fully managed, as-a-service offerings where the underlying configurations (processing power, memory, networking, and the like) are unknown. Our testing demonstrates a narrow slice of potential configurations and workloads.

As the sponsor of the report, NGINX opted for a default NGINX installation and API gateway configuration out of the box – the solution was not tuned or altered for performance. GigaOm selected identical hardware configurations for both App Protect and ModSecurity. The fully managed AWS WAF was used "as-is," since, by virtue of being fully managed, we have no access, visibility, or control over its infrastructure.

We leave the issue of fairness for the reader to determine. We strongly encourage you to look past marketing messages and discern for yourself what is of value. We hope this report is informative and helpful in uncovering some of the challenges and nuances of security architecture selection.

We have provided enough information in the report for anyone to reproduce this test. You are encouraged to compile your own representative workloads and test compatible configurations applicable to your requirements.

2. API Security in the Cloud

The landscape of API security in the cloud varies greatly based on an organization's need and underlying architecture. API security solutions offer either build-your-own or fully managed cloud deployment styles, and a few offer both. While there are many ways to secure APIs, we are interested specifically in enabling security while maintaining high performance. Again, for this report, we define "**high performance**" as companies that experience workloads of **more than 1,000 transactions per second** and need a **maximum latency below 30 milliseconds** across backend APIs and microservices.

Furthermore, in terms of high performance, we focus particularly on latency results at the 99th percentile and above. At first glance, this might seem like an outlier case. However, in our experience, these measures are extremely important in latency results, which tend to be multi-modal over time, with the tops of the spikes representing "hiccups" in response times.

These hiccups matter. If the median response time or latency is less than 30 milliseconds, but there are "hiccups" with latencies above 1 second, the cumulative effect will impact subsequent user experiences. For example, if you visit a fast food drive-through where the median wait time for food is 1 minute, you probably think that was a good customer experience. However, what if the customer in front of you has a problem with their order, and it takes 10 minutes to resolve? Your wait time would actually be 11 minutes. Because your request came in line after the "hiccup," the 99.99th percentile's delay becomes your delay too.

This report aims to explore vendor API security options to better support this high-end performance use case.

NGINX App Protect

NGINX Open Source was first released in 2004 as a reverse proxy load balancer, web server, mail proxy, and HTTP cache. NGINX is offered as free-to-use, open-source software. Its popularity is evident in a March 2020 report by Netcraft[1] that found that 37% of all public websites use NGINX, compared to 24% that use Apache (whose usage peaked at 70% of all websites over a decade ago). NGINX was acquired by F5 Networks in 2019.

NGINX Plus is a commercial offering built atop NGINX Open Source and was first introduced in 2013. It quickly grew into a robust load balancer, Kubernetes Ingress controller, API gateway, and sidecar proxy solution, with hundreds of millions of instances deployed worldwide.

NGINX App Protect combines the proven effectiveness of F5's advanced WAF technology with NGINX's agility and performance. App Protect runs natively on NGINX Plus to address security challenges facing modern DevOps environments.

For App Protect, we installed the latest version of Attack Signatures and Threat Campaigns to enable the highest level of protection for underlying applications and APIs.

ModSecurity

ModSecurity was developed in 2002 to protect the Apache HTTP web server. In 2010, ModSecurity was acquired by Trustwave SpiderLabs and released as an open-source, cross-platform web application firewall (WAF) module. Known as the "Swiss Army Knife" of WAFs, it enables web application defenders to gain visibility into HTTP traffic and provides a rules-language and API to implement advanced protections.

For this test, we employed the OWASP ModSecurity Core Rule Set (CRS). CRS is a set of generic attack detection rules for use with ModSecurity and compatible web application firewalls. This ruleset aims to protect APIs from a wide range of attacks, including the OWASP Top Ten, with a minimum of false alerts.

Amazon Web Services Web Application Firewall

Amazon Web Application Firewall (WAF) is a fully managed cloud service that provides a web application firewall to help protect web applications and APIs against common web exploits that may affect availability, compromise security, or consume excessive resources.

For AWS WAF, we installed the AWSManagedRulesCommonRuleSet, which is also the OWASP implementation of the Core Rule Set for AWS WAF.

---

1. https://news.netcraft.com/archives/category/web-server-survey/

3. GigaOm API Workload Test Setup

API Workload Test

The GigaOm API Workload Field Test is a simple workload designed to attack an API or an API management worker node (or a load balancer in front of a cluster of worker nodes) with a barrage of identical GET requests at a constant number of requests per second.

To perform the attacks, we used the HTTP load testing tool, Vegeta, a free-to-use workload test kit available on GitHub. The Vegeta tool returns a results bin file that contains the latencies and status code of every request. The attacker measured latency as the elapsed time between the points when an individual API request was made and when the API response back was received. Thus, if we tested 1,000 requests per second for 60 seconds, the attack tool recorded 60,000 latency values. We used that data to compile and interpret the results of the test.

The test also requires a backend API that can listen and respond to requests. In this case, our backend API listens for a GET request that includes an integer, such as:

```
http://ipaddress/size/10
```

The API would respond with a string of pseudorandom Unicode characters, such as:

```
taZ3psgHkQ
```

where “10” is the size in bytes of the desired return string. Thus, if we wanted 1 KB of data returned, we would send the request:

```
http://ipaddress/size/1024
```

For these tests, we used a request return size of 1 KB.

The backend API we used is further documented in the Appendix.

We completed three attempts per test on each platform, configuration, and request rate. We started with an attack rate of 1,000 requests per second (rps) and incremented up to attack rates of 2,000 rps, 3,000 rps, and 5,000 rps. We ran each test for 60 seconds. We captured the latencies at the 50th, 90th, 95th, 99th, 99.9th, 99.99th percentiles and the maximum latency seen during the test run. We recorded the test run that resulted in the lowest maximum latency or the highest success rate in the event of errors. Error status codes included HTTP status codes 429 Too Many Requests or any 5xx codes, most often 500 Internal Server Error. A success rate of 100% meant all requests returned a 200 OK status code.

In addition to standard attacks with "good" or expected traffic, we also tested the configuration's ability to block "bad" traffic under load. For this metric, we tested these security mechanisms with 5% and 10% bad traffic. The "bad" traffic was sent as script injections, such as:

```
    http://ipaddress/?v=<script>
```

The results are shared in the Field Test Results section.

Test Environments

Selecting and sizing the compute and storage for comparison can be challenging, particularly for fully managed as-a-service vendor offerings. The figures below give a visual layout of the configurations we tested.

Configurations Used for NGINX Comparable Tests

The first configuration we tested was single node workers of NGINX with no security. We installed Vegeta on an Attack node and performed the API requests directly to the API Worker, which routed the requests through to a Backend API. The Backend API would respond back to the API Worker Node, which routed the responses back to the Attack Node.

We installed all test components (excluding AWS WAF) onto AWS EC2 instances. For the components vital to the test, we used the "c5n" family of EC2 instances to take advantage of enhanced networking capabilities. According to Amazon:

C5n instances are ideal for high compute applications (including High-Performance Computing (HPC) workloads, data lakes, and network appliances such as firewalls and routers) that can take advantage of improved network throughput and packet rate performance. C5n instances offer up to 100 Gbps network bandwidth and increased memory over comparable C5 instances.

We also made a single operating system-level change to all NGINX instances. We increased the user limits for soft and hard open files to 65,536. The reason for this move is that during high response per second attack API testing, you can experience errors on worker nodes when user open file limits are hit.

Configurations Used for Comparable Tests of Fully Managed Platforms

For the fully managed security mechanism (AWS WAF), we were limited on the configurations we could test and still achieve a comparable result. AWS WAF itself lacks a method to route requests to a Backend API without a load balancer. Thus, we also deployed a single AWS Load Balancer and associated AWS WAF with it. The "instance type" of AWS WAF is unknown--that is, it is a fully-managed serverless platform, and we do not know its CPU or memory capabilities.

As shown in **Table 1**, we also installed the Attack Node, Backend APIs, NGINX instances, and the AWS Load Balancer test components within the same placement group to ensure the closest network proximity.

Table 1. EC2 Instances Used for Test Components

Attack Node	C5n.2xlarge
Backend API	C5n.large
NGINX No Security	C5n.large
NGINX Mod Security	C5n.large
NGINX App Protect	C5n.large

Source: GigaOm 2020

Results may vary across different configurations, and again, you are encouraged to compile your own representative workloads and test compatible configurations applicable to your requirements.

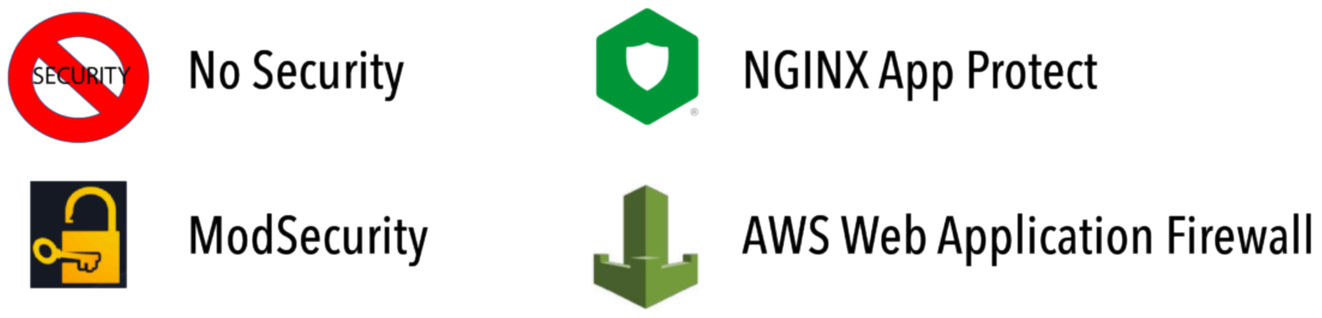
4. Test Results

This section analyzes the latencies in milliseconds from the various 60-second runs of each of the scaled GigaOm API Workload Field Tests described above. A lower latency is better—meaning API responses are coming back faster. Also, the latency reveals the response time at the 50th, 90th, 95th, 99th, 99.9th, and 99.99th percentiles and the maximum latency. These are important values for service-level agreements (SLAs) and gauging the slowest response times a user might experience.

Test Results without AWS WAF

Here we show all of the results of the App Protect, Mod Security, and No Security runs for all tests that are completed. Tested elements are identified in the charts using the images in the chart key, as follows:

Charts Key



Source: GigaOm 2020

Test Results with No Bad Traffic

For these tests, we did not send any bad traffic through, in order to get a baseline of latency. Focusing on the 99th percentile, at the 1,000 rps level, Mod Security had 3.76 times the latency of App Protect. At the 99th percentile, App Protect response times were less than 2 milliseconds slower than with no security at all. **Figure 1** shows the results.

At 2,000 rps (**Figure 2**), Mod Security response times were in the 5.5 second range, while App Protect was less than 1 millisecond. At 3,000 rps and above, Mod Security was not processing the traffic at 100% success, so it is not depicted on the charts in **Figures 3** and **4**. Meanwhile, App Protect and No Security were generally very close in latency at both the 3K rps and 5K rps levels.

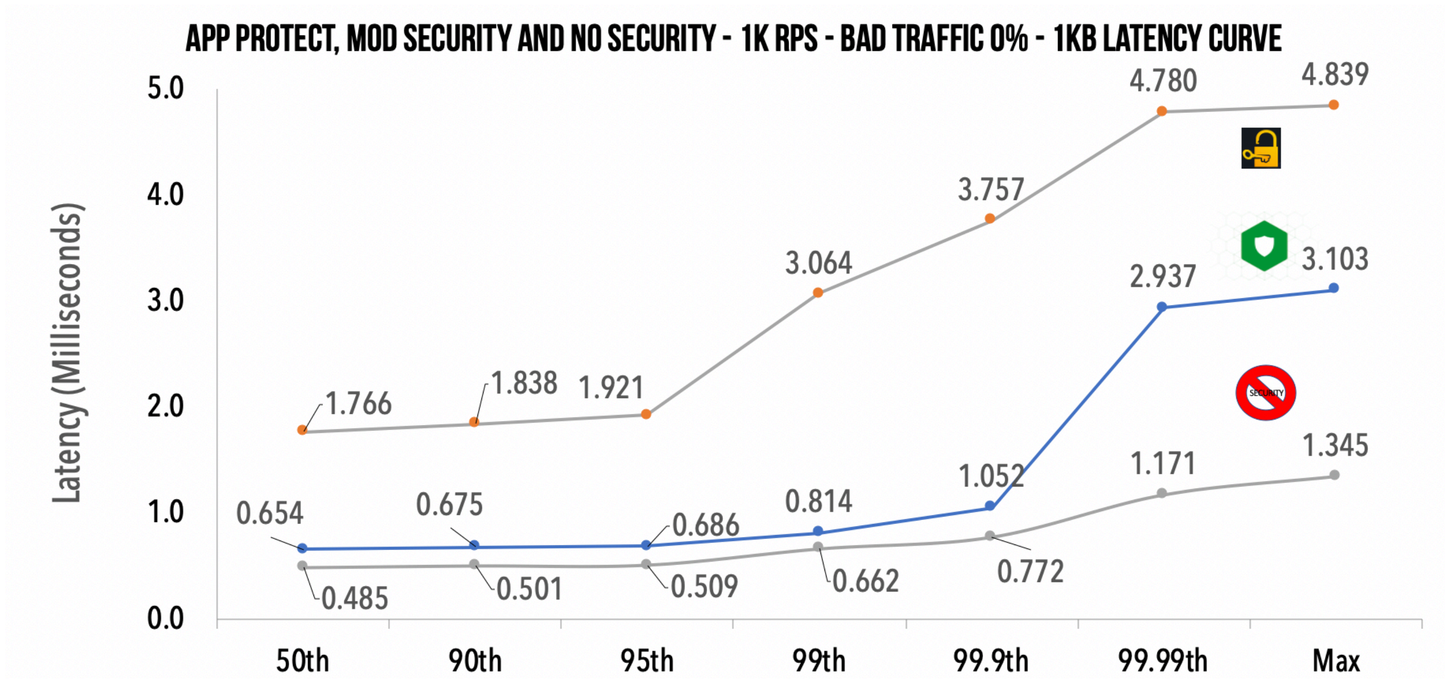


Figure 1. Latency Curve at 1K rps

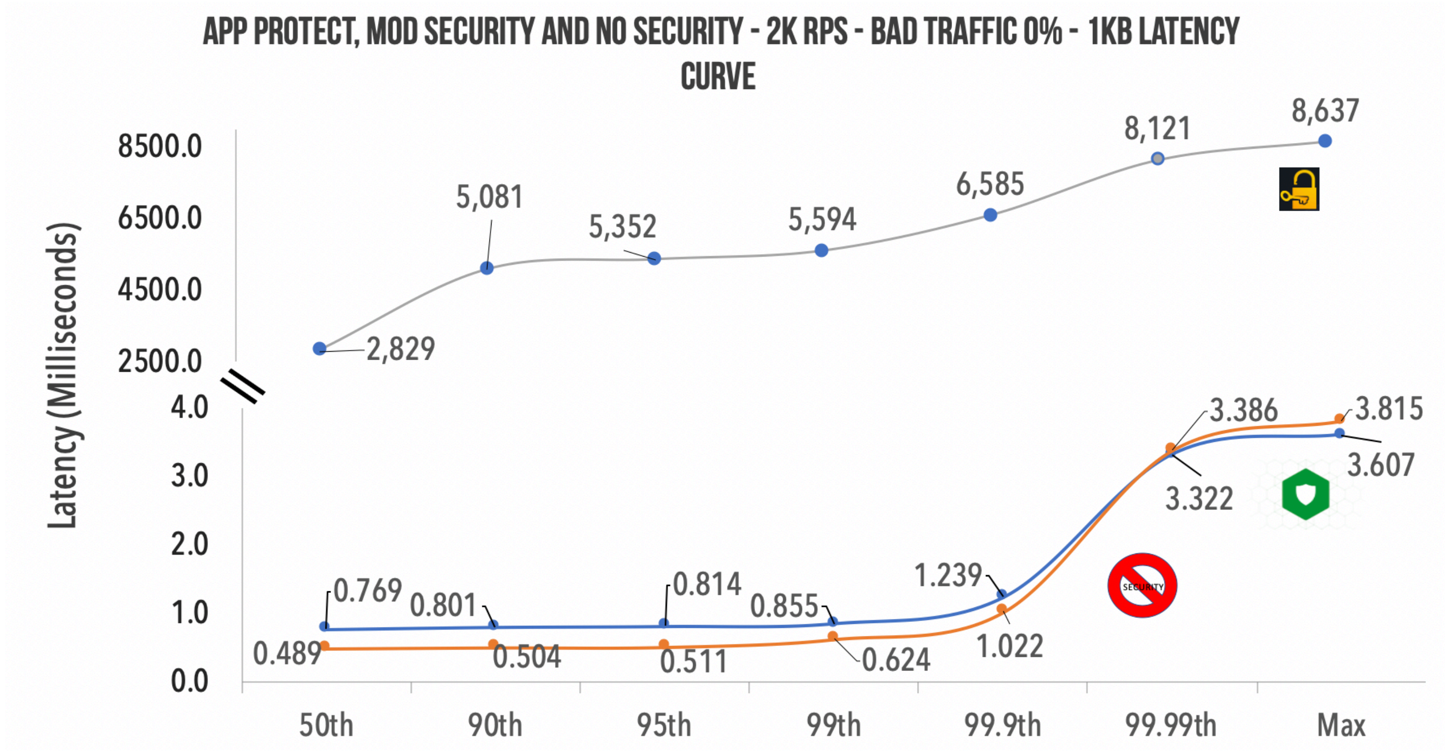


Figure 2. Latency Curve at 2K rps

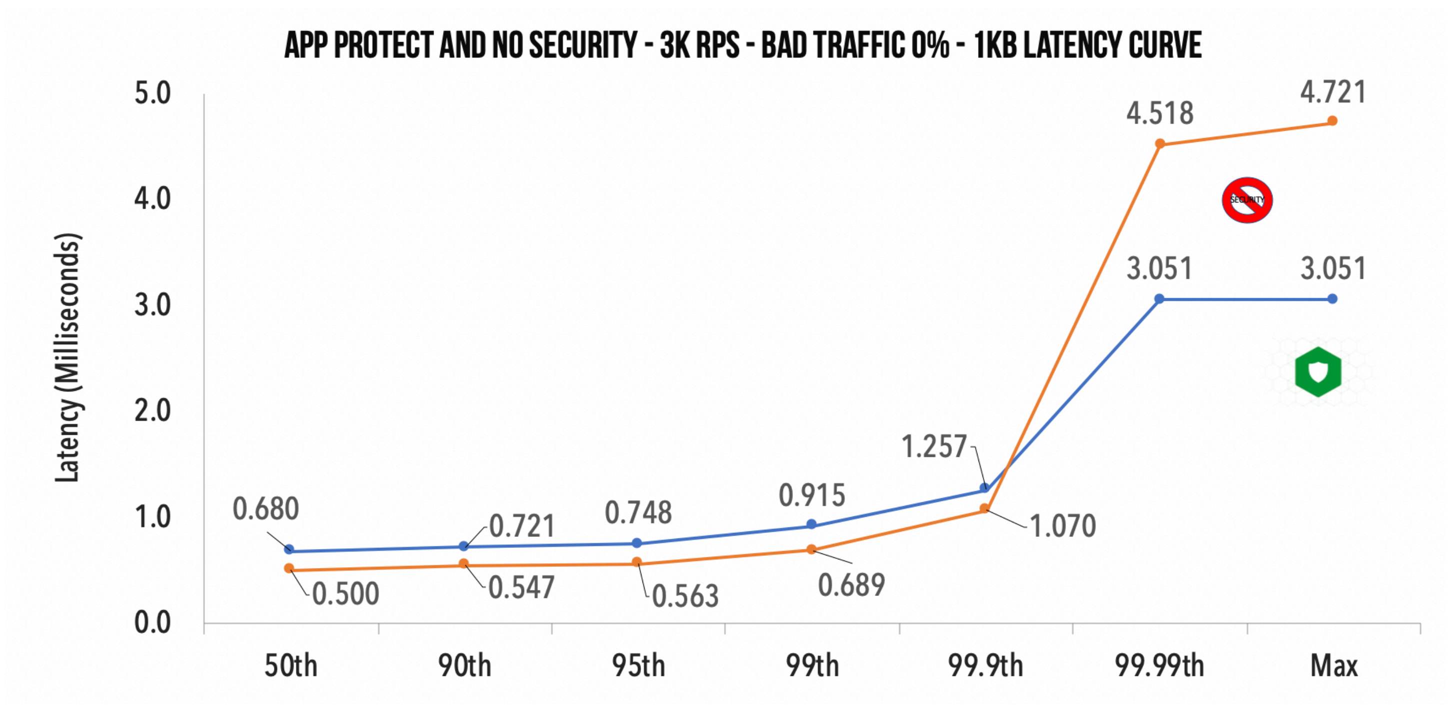


Figure 3. Latency Curve at 3K rps

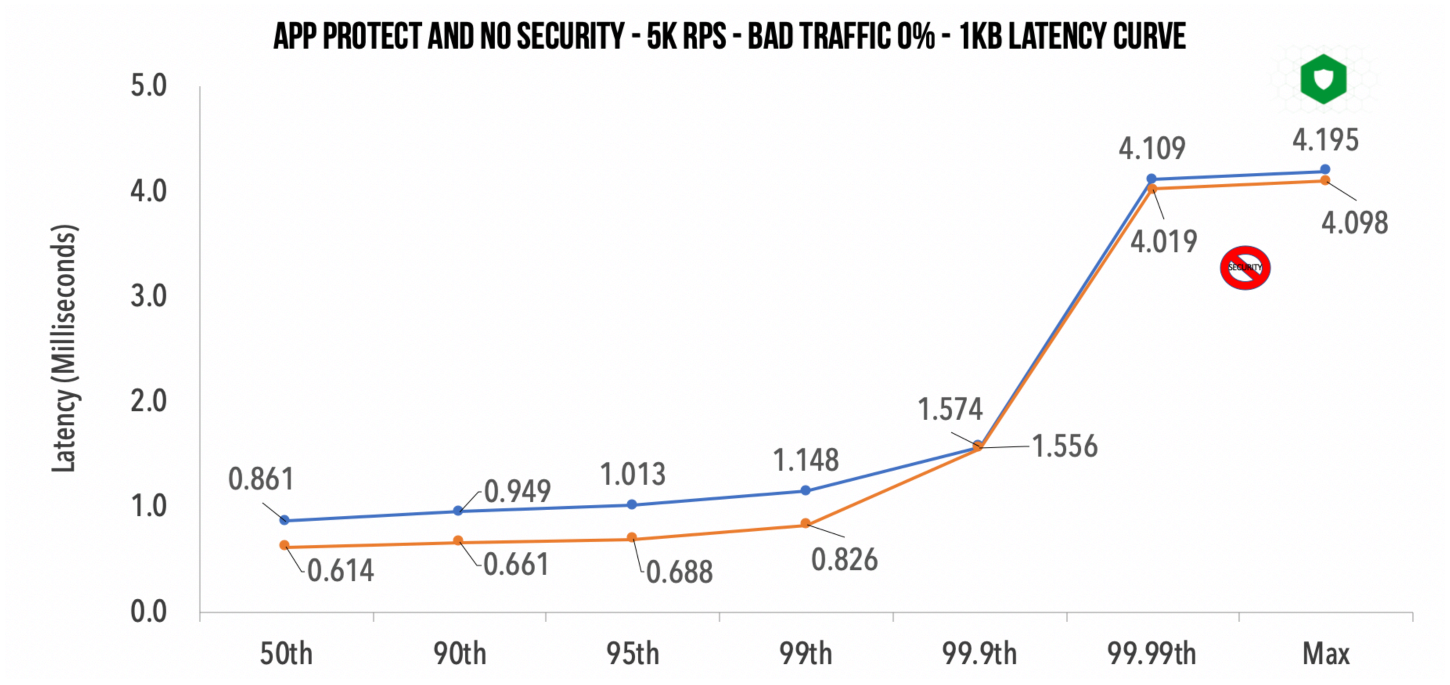


Figure 4. Latency Curve at 5K rps

Test Results with 5% Bad Traffic

With 5% bad traffic, Mod Security consistently produced higher latency times than App Protect did. At 1,000 rps of combined good and bad traffic, shown in **Figure 5**, Mod Security latency was 11.5 times higher than App Protect at the 99th percentile. We show charts for the good traffic (950 rps) and bad traffic (50 rps) below.

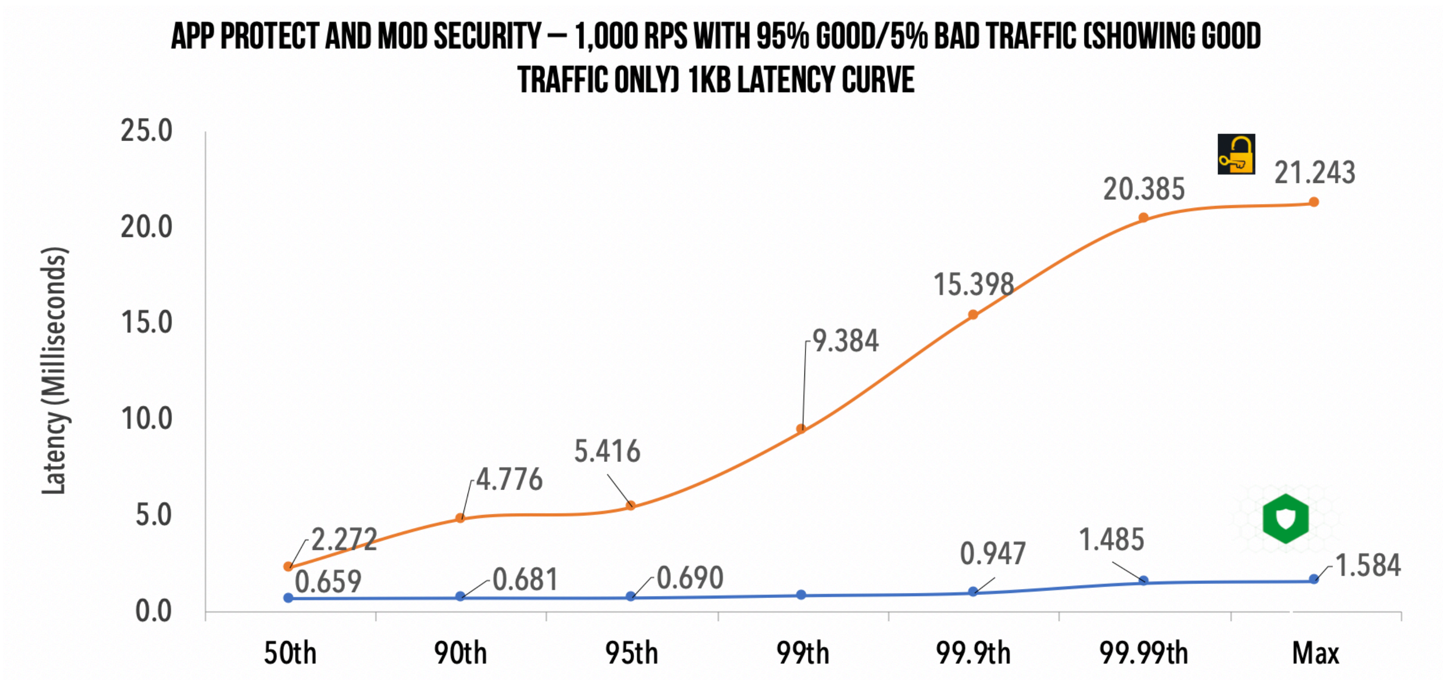


Figure 5. Latency Curve at 1K rps and 5% Bad Traffic

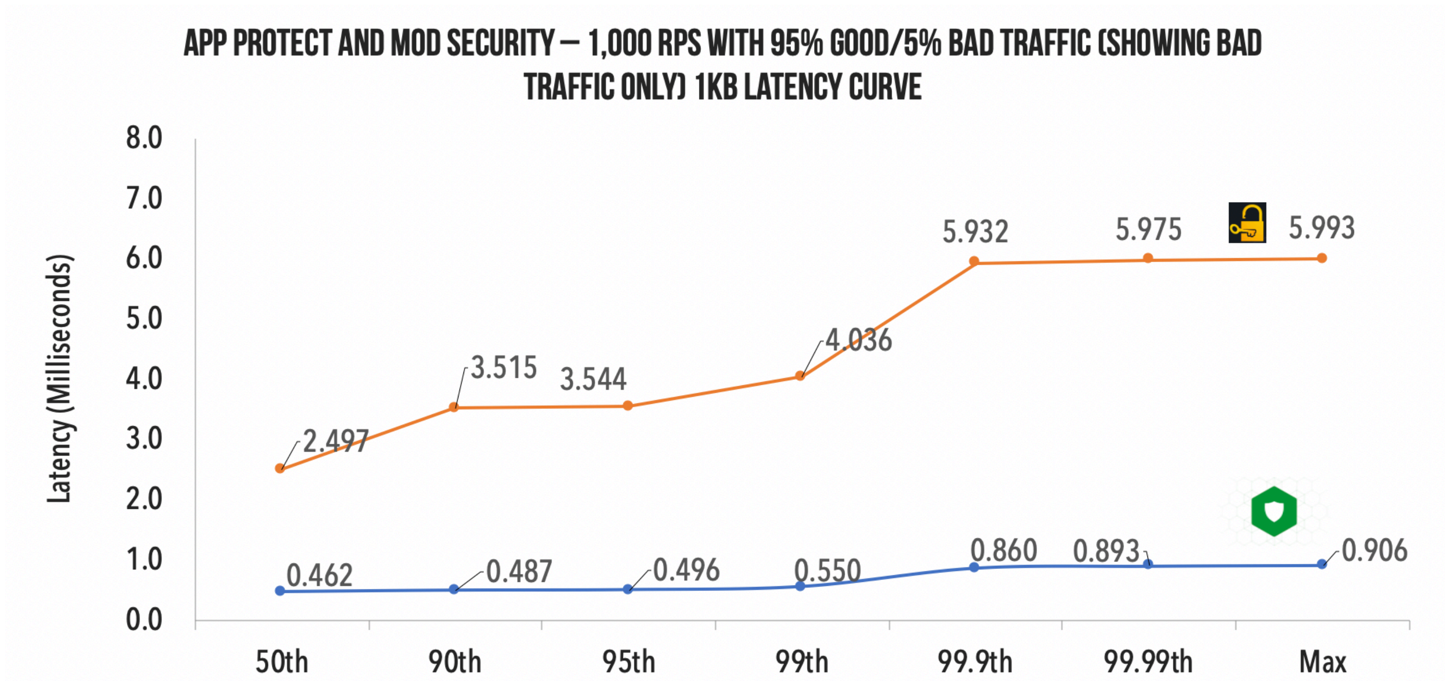


Figure 6. Latency Curve at 1K rps and 5% Bad Traffic

Test Results with 10% Bad Traffic

With 10% bad traffic, Mod Security consistently continued to have more latency than App Protect. At 1,000 rps of combined good and bad traffic, the latency gap was 6.3 times at the 99th percentile. We show charts for the good traffic (900 rps in **Figure 7**) and bad traffic (100 rps in **Figure 8**) below.

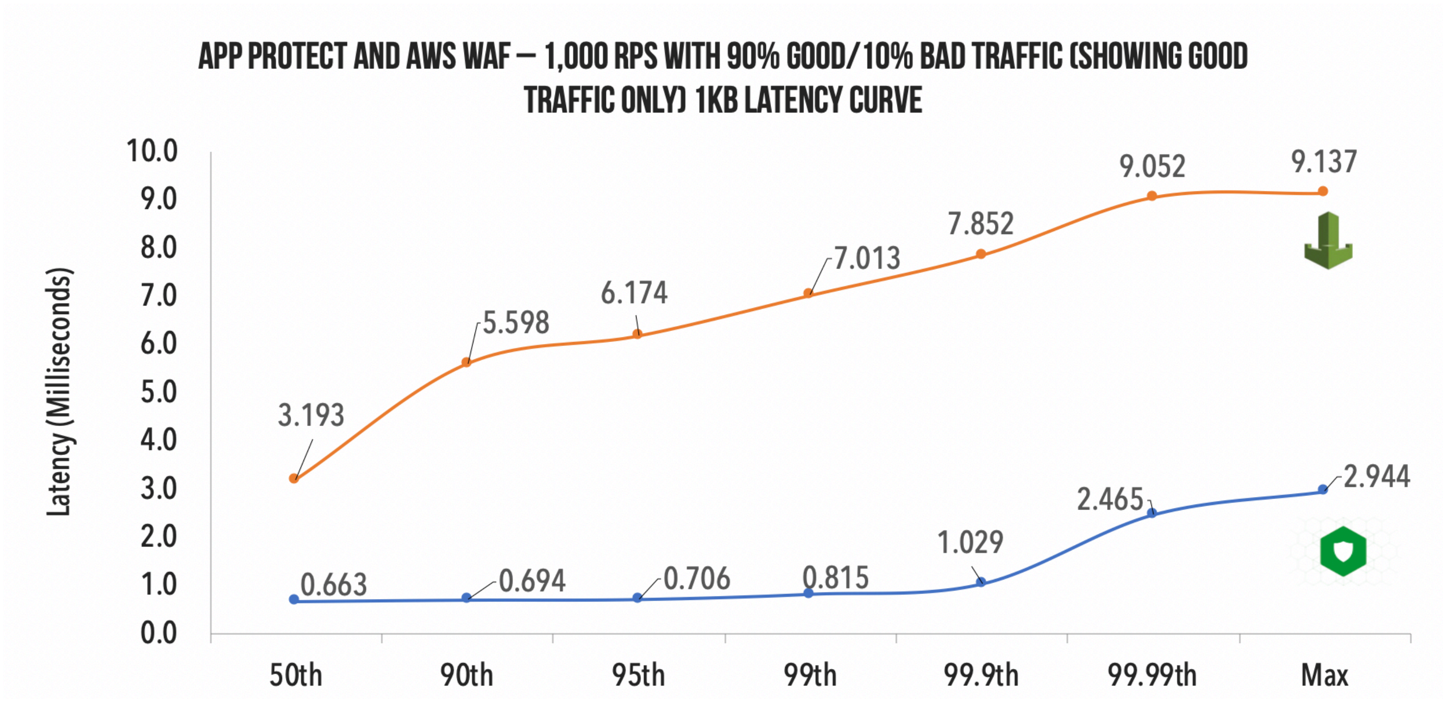


Figure 7. Latency Curve at 1K rps and 10% Bad Traffic

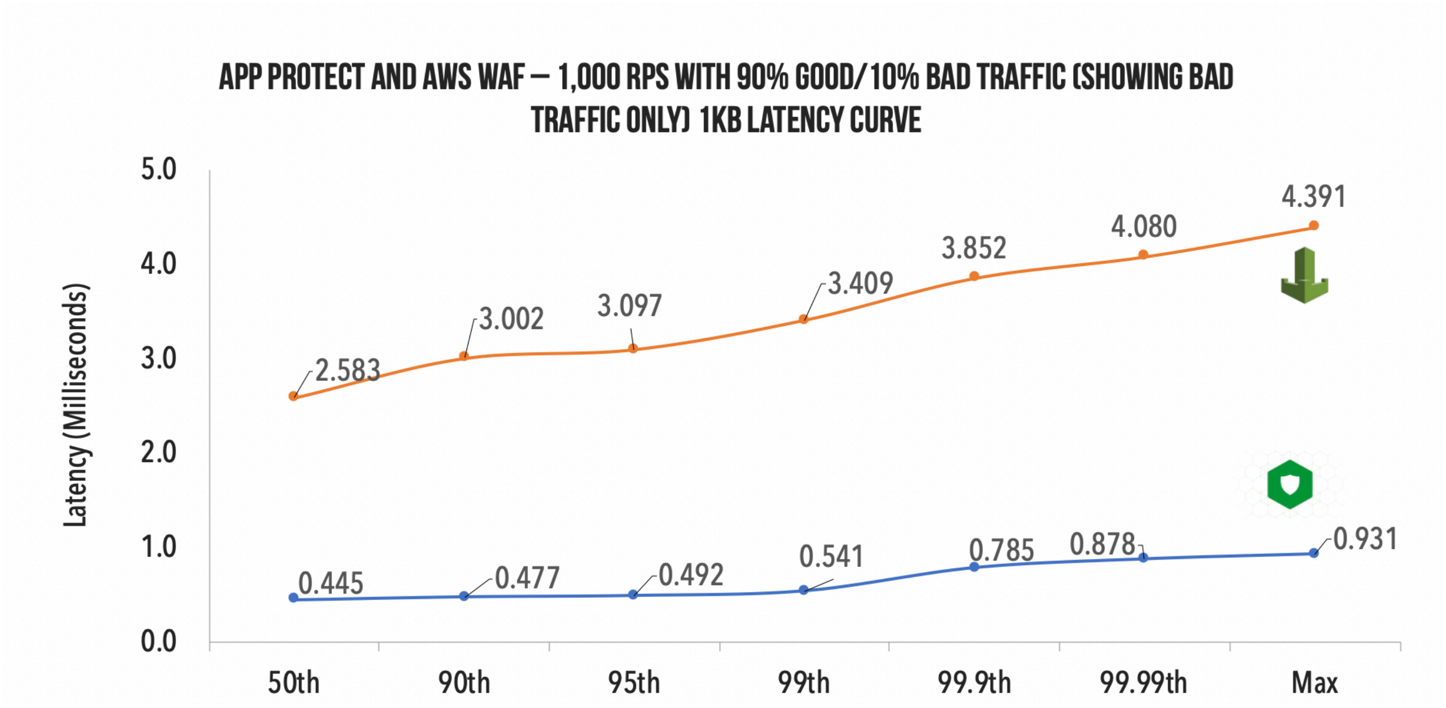


Figure 8. Latency Curve at 1K rps and 10% Bad Traffic

Test Results with AWS WAF

Test Results with No Bad Traffic

With no bad traffic, AWS WAF produced much higher latency than App Protect, with the gap expanding at the 99.9th percentile for 1K rps (**Figure 9**), 2K rps (**Figure 10**), and 3K rps (**Figure 11**). At 5K rps, the difference was dramatic from the start, as shown in **Figure 12**.

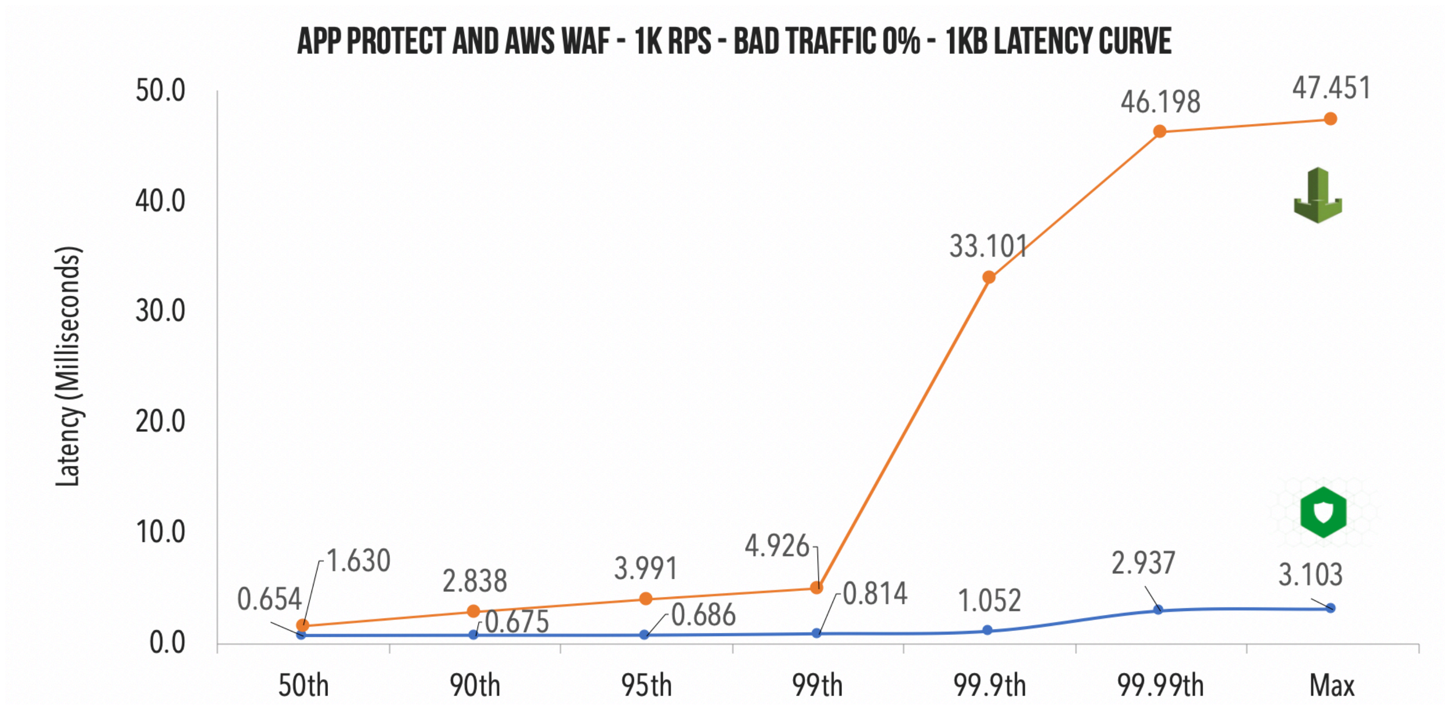


Figure 9. Latency Curve versus AWS WAF at 1K rps

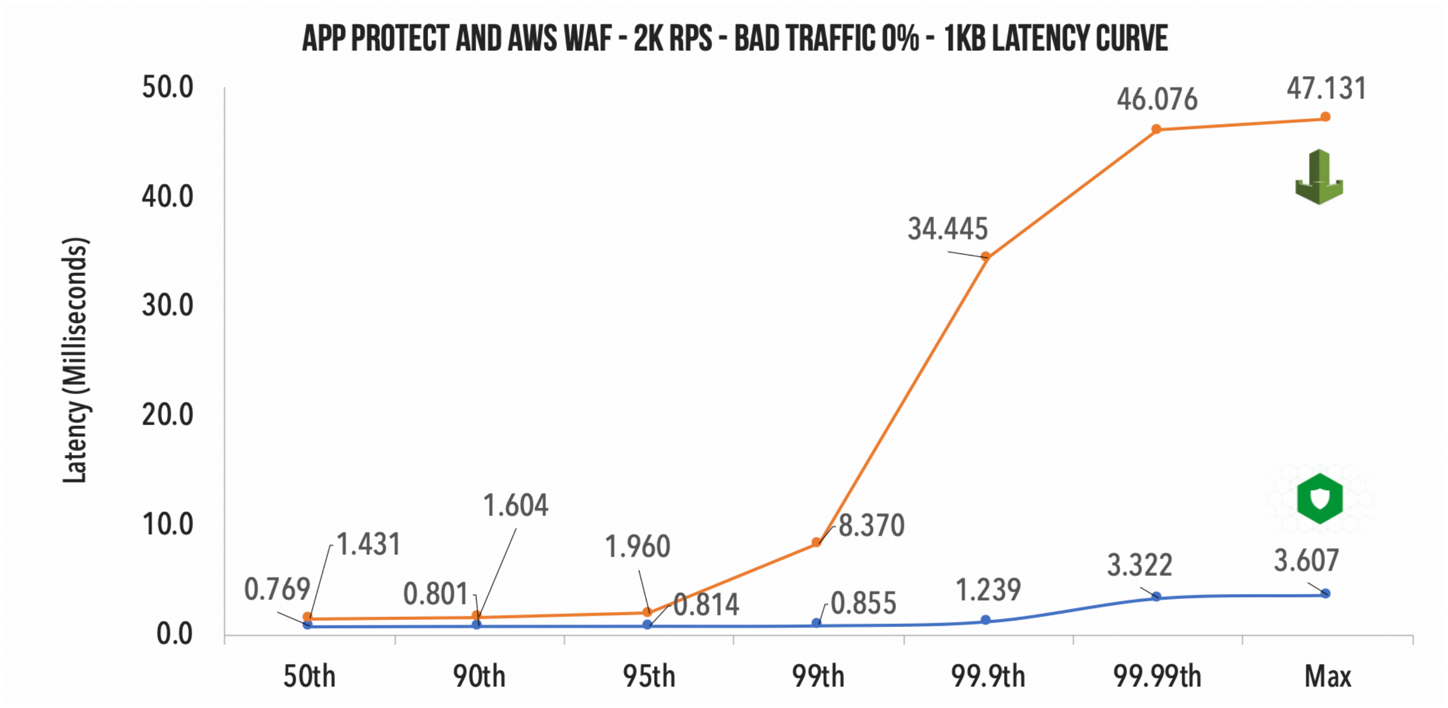


Figure 10 - Latency Curve versus AWS WAF at 2K rps

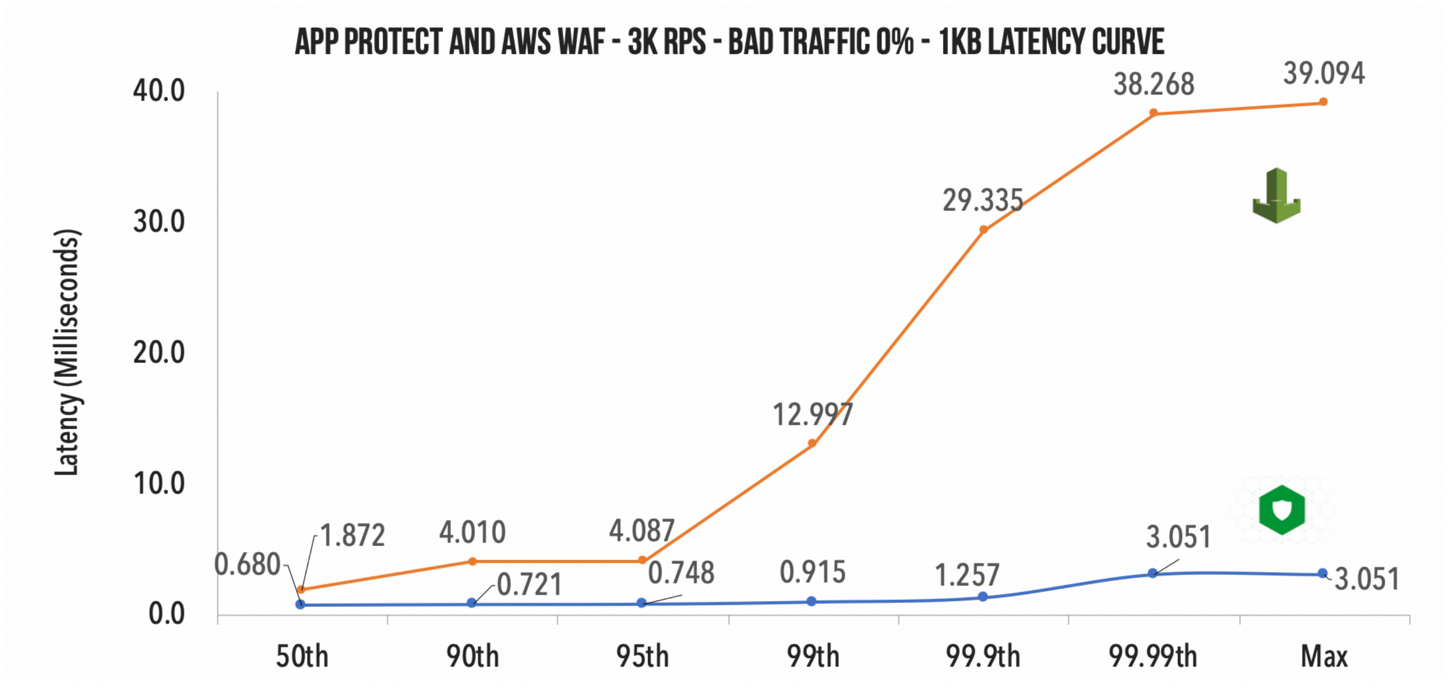


Figure 11. Latency Curve versus AWS WAF at 3K rps

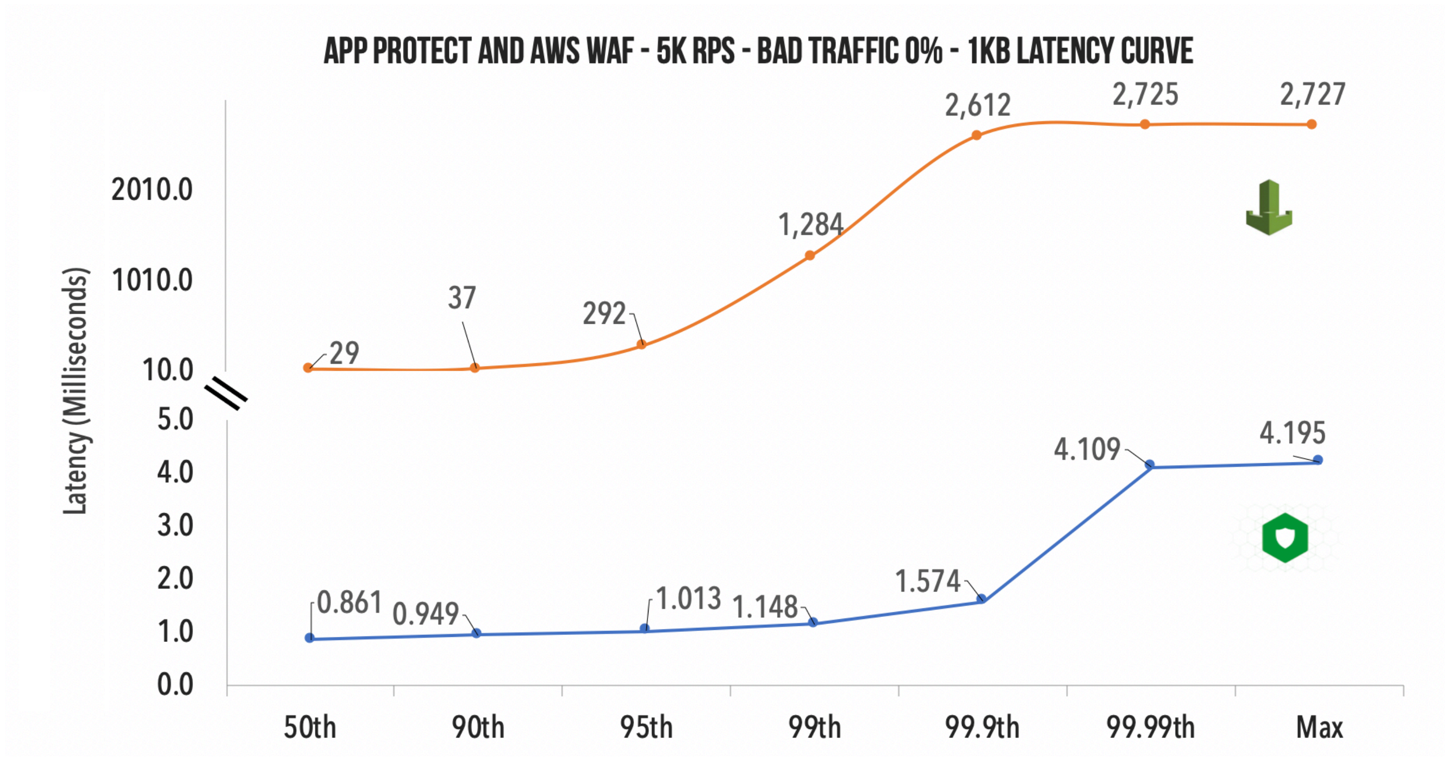


Figure 12. Latency Curve versus AWS WAF at 5K rps

Test Results with 5% Bad Traffic

When bad traffic was introduced, AWS WAF experienced more latency at the higher percentiles, with an expanding gap at the 99.99th percentile. **Figures 13** through **16** show the results.

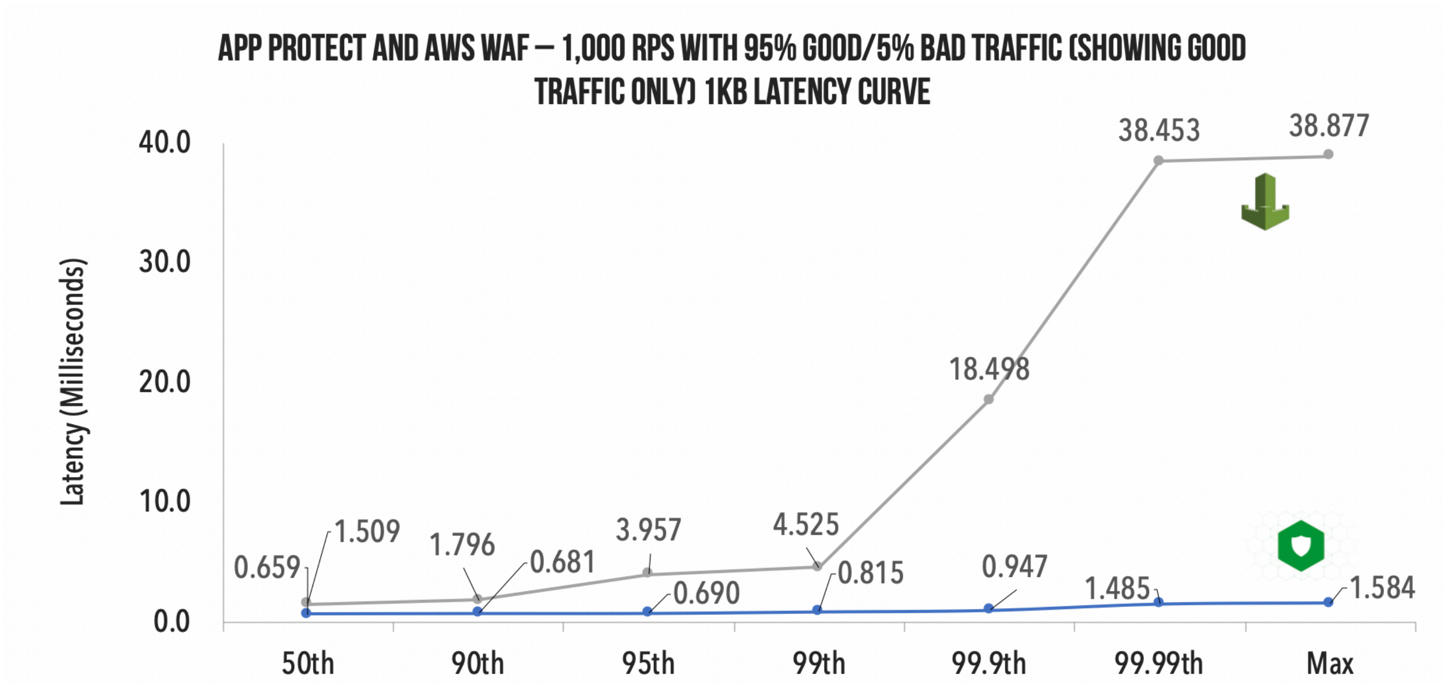


Figure 13. Latency Curve versus AWS WAF at 1K rps and 5% Bad Traffic

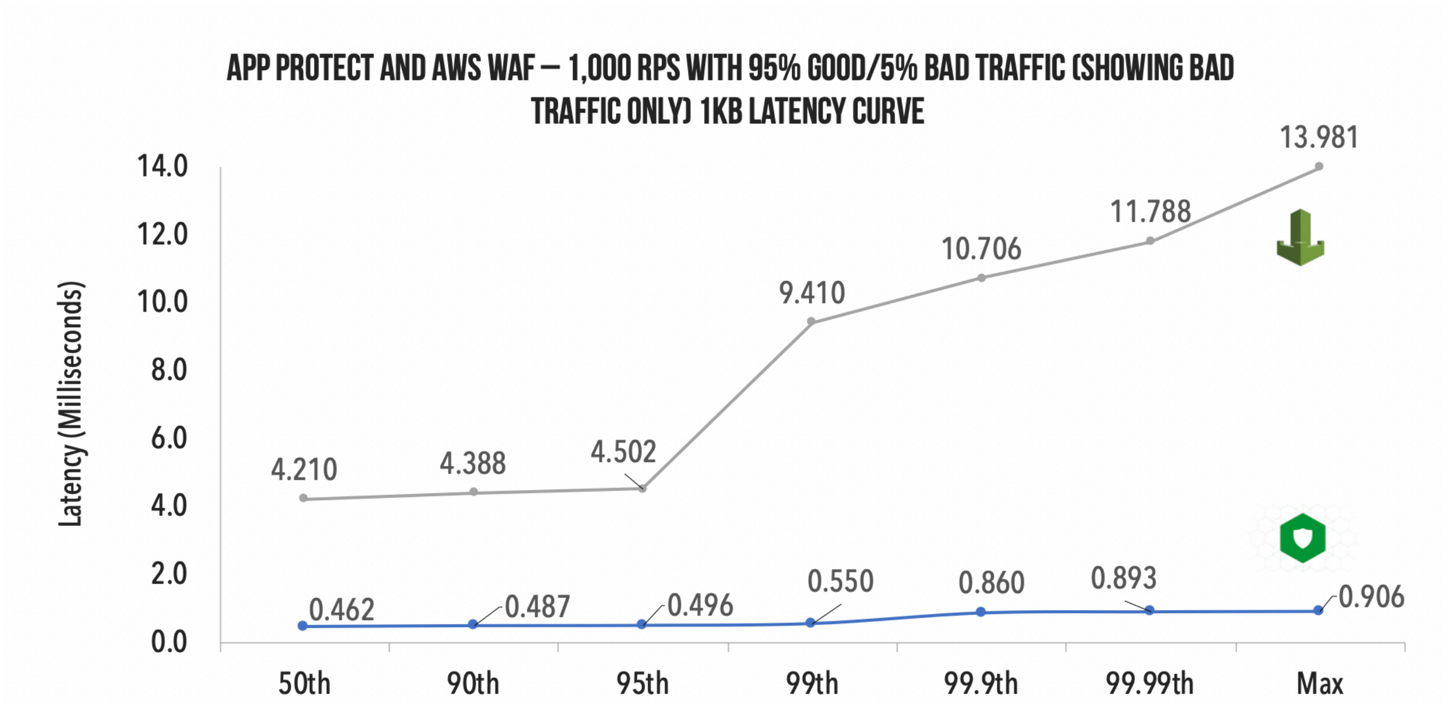


Figure 14. Latency Curve versus AWS WAF at 1k rps and 5% Bad Traffic

Test Results with 10% Bad Traffic

The 5% bad traffic pattern continued at 10% bad traffic.

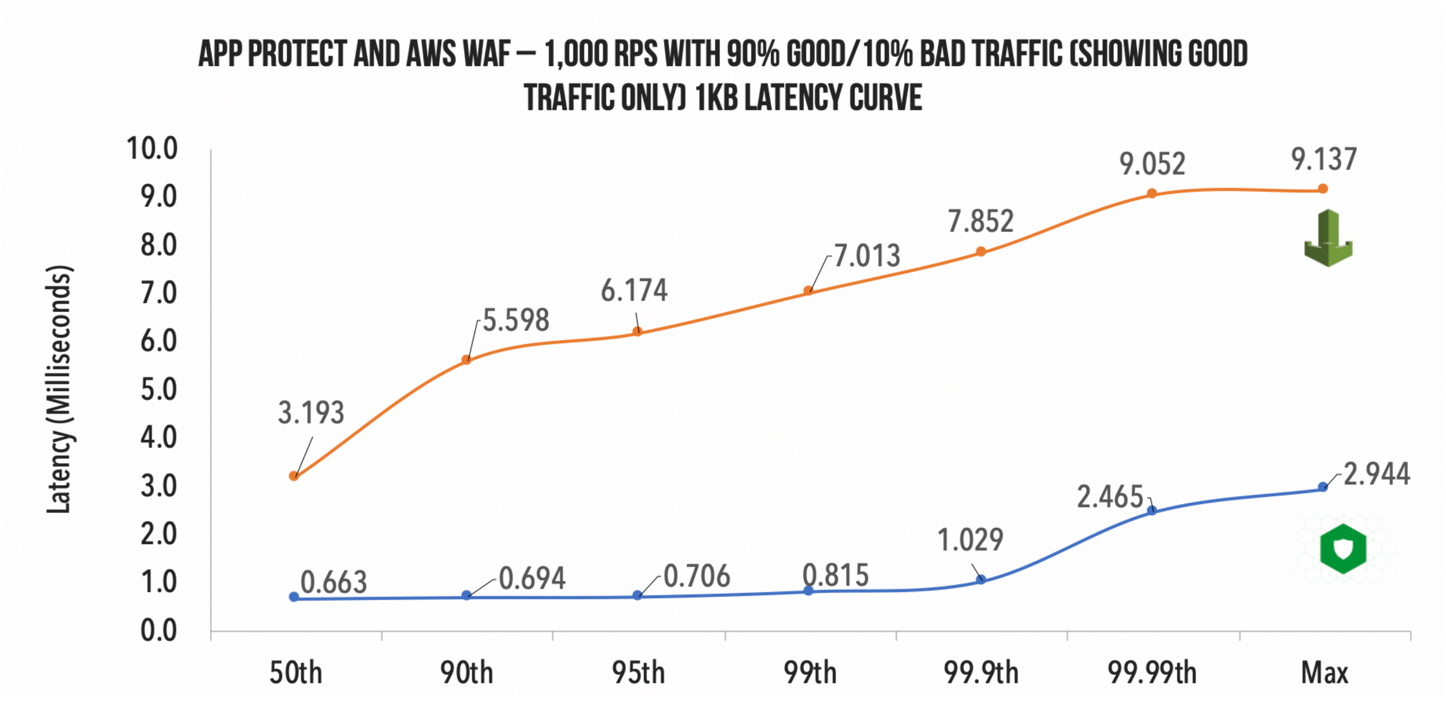


Figure 15. Latency Curve versus AWS WAF at 1K rps and 10% Bad Traffic

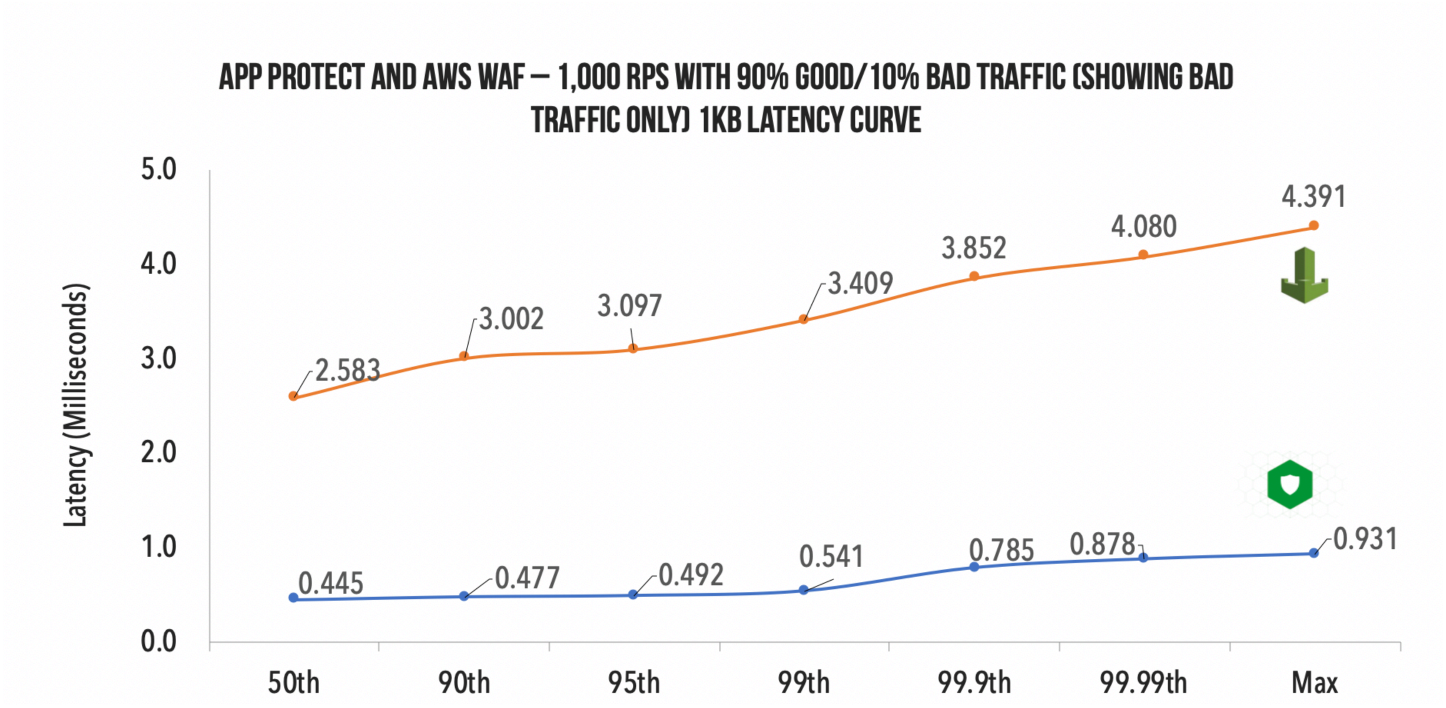


Figure 16. Latency Curve versus AWS WAF at 1K rps and 10% Bad Traffic

Finally, we highlight the maximum throughput achieved with 100% success and no 5xx or 429 errors and with less than 30ms maximum latency. See **Figure 17**.

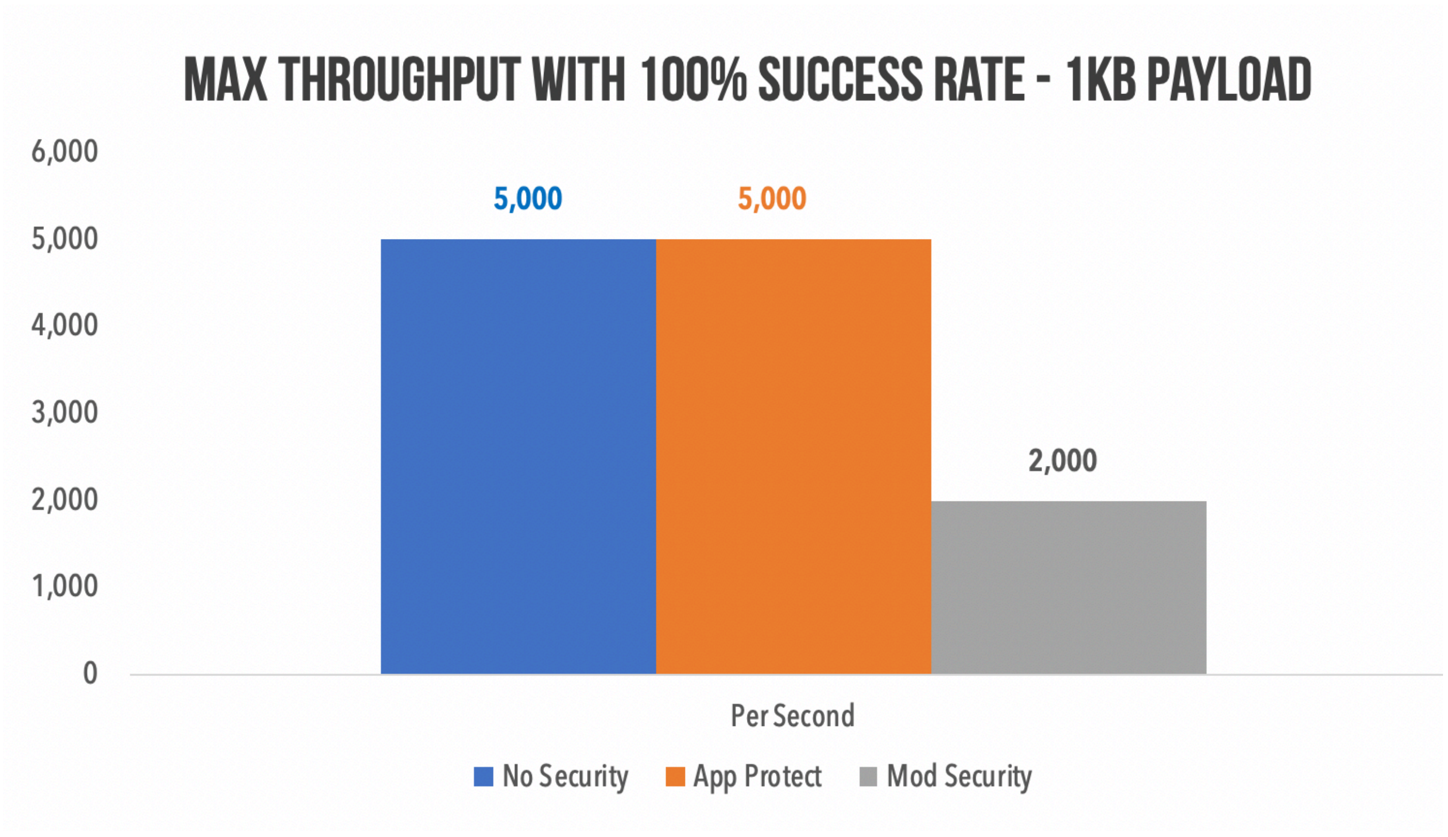


Figure 17. Max Throughput

The maximum transaction throughput achieved with 100% success (no 5xx or 429 errors) and with less than 30ms maximum latency with our tiny c5n.large (2 CPU and 5.25 GB RAM) instance was approximately 5,000 requests per second for NGINX App Protect. By comparison, ModSecurity began to produce errors at the 2,000 requests per second threshold. That is, above 2,000 rps, we started to receive 500 and 429 errors back from NGINX with ModSecurity, and above 5,000 rps, we started to receive 500 and 429 errors back from NGINX with App Protect.

5. Conclusion

This report outlines the results from a GigaOm API Workload Field Test.

NGINX App Protect outperformed ModSecurity at all tested attack rates. NGINX App Protect had 92% lower latency of NGINX running ModSecurity at the 99th percentile at 1,000 tps on the 5% bad request test. The latencies for App Protect and ModSecurity diverged at higher percentiles. Although the differences are minimal until you get to the 90th percentile, the difference in latency is pronounced at the 95th percentile and above.

For fully managed offerings, NGINX App Protect had 82% lower latency than AWS WAF at 1,000 tps, on the 5% bad request test. Since AWS WAF is fully managed, we do not know what underlying compute resources are working behind the scenes, making it difficult to perform a true apples-to-apples comparison. Although the differences are minimal until you get to the 90th percentile, the difference in latency is pronounced at the 99th percentile and above.

On a single small 2 CPU and 5.25GB of RAM EC2 instance, the maximum transaction throughput achieved with 100% success (no 5xx or 429 errors) and with less than 30ms maximum latency was approximately 5,000 rps for NGINX App Protect, compared to only 2,000 rps for ModSecurity. App Protect in the test was found to provide the same level of throughput as No Security.

For *this test* using *this particular workload with these particular configurations*, API requests came back with the lowest latencies and highest throughput on NGINX App Protect compared to any of the other tested security solutions.

Keep in mind, optimizations on all platforms would be possible as the offerings evolve or internal tests point to different configurations.

6. Appendix: Recreating the Test

The back-end API used in this test was a custom application developed by GigaOm. It is a Python application that leverages the free-to-use Gunicorn WSGI HTTP web server and Falcon API frameworks.

The application works by binding the API application to port 8000 with Gunicorn and listening for GET requests, such as:

```
    GET http://fqdn-or-ip-address:8000/size/10
```

The API would respond with a string of pseudorandom Unicode characters from /dev/urandom, such as:

```
    taZ3psgHkQ
```

where “10” is the size in bytes you desire the return string to have.

The following is the code for the back-end API. You are free to use and modify it at your own discretion. GigaOm makes no warranty or claim for its use beyond the scope of this test or report.

```
import falcon
import sys
from base64 import b64encode
from os import urandom
def generate_string(string_length):
    random_bytes = urandom(string_length)
    token = b64encode(random_bytes).decode('utf-8')
    return token
class SizeResource(object):
    def on_get(self, req, resp, size):
        resp.status = falcon.HTTP_200
        resp.body = (generate_string(int(size)))
app = falcon.API()
size = SizeResource()
app.add_route('/size/{size}', size)
```

The Python application was compiled using the PyPy just-in-time compiler to improve performance. Our goal was to create a back-end API that was as performant and lightweight as possible so the latency generated by the application itself was minimized.

7. Disclaimer

Performance is important but it is only one criterion for a Web Application Firewall selection. This test is a point-in-time check into specific performance. There are numerous other factors to consider in selection across Administration, Features and Functionality, Workload Management, User Interface, Scalability, Vendor, Reliability, and numerous other criteria. It is also our experience that performance changes over time and is competitively different for different workloads. Moreover, a performance leader can hit up against the point of diminishing returns and viable contenders can close the gap quickly.

GigaOm runs all of its performance tests to strict ethical standards. The results of the report are the objective results of the application of load tests to the simulations described in the report. The report clearly defines the selected criteria and process used to establish the field test. The report also clearly states the tools and workloads used. The reader is left to determine for themselves how to qualify the information for their individual needs. The report does not make any claim regarding the third-party certification and presents the objective results received from the application of the process to the criteria as described in the report. The report strictly measures performance and does not purport to evaluate other factors that potential customers may find relevant when making a purchase decision.

This is a sponsored report. NGINX chose the competitors and the test, and the NGINX Plus configuration was the default provisioned by NGINX Controller. GigaOm chose the most compatible configurations as-is out-of-the-box, and ran the testing workloads. Choosing compatible configurations is subject to judgment. We have attempted to describe our decisions fully in this report.

8. About NGINX

NGINX, acquired by F5 in 2019, is behind the popular open source project trusted by more than 450 million sites. NGINX offers a suite of technologies for developing and delivering modern applications. NGINX software solutions enable enterprises undergoing digital transformation to modernize legacy and monolithic applications as well as deliver new, microservices-based applications. Companies like Netflix, Starbucks, and McDonalds rely on NGINX to reduce costs, improve resiliency, and speed innovation. NGINX simplifies the journey to microservices. As enterprises move to a DevOps approach to application development and delivery, the tools, stack and interoperability of it all can get very complex. NGINX software reduces this complexity by consolidating common functions like load balancing, API management, security (App Protect), and service mesh down to far fewer components, to help make application infrastructure scalable and more manageable.

9. About Jake Dolezal

As a contributing Analyst at GigaOm, Jake Dolezal has two decades of experience in the Information Management field with expertise in analytics, data warehousing, master data management, data governance, business intelligence, statistics, data modeling and integration, and visualization. Jake has experience across a broad array of industries, including: healthcare, education, government, manufacturing, engineering, hospitality, and restaurants. He has a doctorate in information management from Syracuse University.

GIGAOM

10. About William McKnight

An Ernst & Young Entrepreneur of the Year Finalist and frequent best practices judge, William is a former Fortune 50 technology executive and database engineer. He provides Enterprise clients with action plans, architectures, strategies, and technology tool selection to manage information.

William McKnight is an Analyst for GigaOm Research who takes corporate information and turns it into a bottom-line producing asset. He's worked with companies like Dong Energy, France Telecom, Pfizer, Samba Bank, ScotiaBank, Teva Pharmaceuticals and Verizon — Many of the Global 2000 — and many others. William focuses on delivering business value and solving business problems utilizing proven, streamlined approaches in information management.

He is a frequent international keynote speaker and trainer. William has taught at Santa Clara University, UC-Berkeley and UC-Santa Cruz.

11. About GigaOm

GigaOm provides technical, operational, and business advice for IT's strategic digital enterprise and business initiatives. Enterprise business leaders, CIOs, and technology organizations partner with GigaOm for practical, actionable, strategic, and visionary advice for modernizing and transforming their business. GigaOm's advice empowers enterprises to successfully compete in an increasingly complicated business atmosphere that requires a solid understanding of constantly changing customer demands.

GigaOm works directly with enterprises both inside and outside of the IT organization to apply proven research and methodologies designed to avoid pitfalls and roadblocks while balancing risk and innovation. Research methodologies include but are not limited to adoption and benchmarking surveys, use cases, interviews, ROI/TCO, market landscapes, strategic trends, and technical benchmarks. Our analysts possess 20+ years of experience advising a spectrum of clients from early adopters to mainstream enterprises.

GigaOm's perspective is that of the unbiased enterprise practitioner. Through this perspective, GigaOm connects with engaged and loyal subscribers on a deep and meaningful level.

12. Copyright