Blog Simple.

Minimize false positives for WAF

In Security Tags apache, modsecurity, nginx, optimize waf, waf October 12, 2019 8 Views Aishee

“You can’t rely on anyone these days, you gotta do everything yourself.”

The Joker!

I haven’t written in a while. Now I have some time to share some of my old experiences.

Start

We often use WAF to protect website, but in the process of using, certainly many times you encounter the case of mistakenly blocking non-malicious requests.

That seems to make us uncomfortable or take a lot of effort to find its cause, probably because we have not optimized the rules of use properly. The false positive could be due, waf working too hard, consuming compute resources for something wrong, will make it block clean traffic.

Sometimes tired, they might think about it, uninstall WAF :v

Tuning WAF is a tedious process, here I will share some ways to minimize false positive for WAF, It will help you properly block malicious traffic and greatly reduce false positives. This article will talk about ModSecurity.

ModSecurity, sometimes called **Modsec**, is an open-source web application firewall (WAF). Originally designed as a module for the Apache HTTP Server, it has evolved to provide an array of Hypertext Transfer Protocol request and response filtering capabilities along with other security features across a number of different platforms including Apache HTTP Server,[1] [2] Microsoft IIS and NGINX. https://en.wikipedia.org/wiki/ModSecurity

We often use OWASP ModSecurity Core Rule Set (CRS) in combination with ModSecurity. This is the best set of rules to prevent attacks on OWASP Top Ten projects. CRS is a rule set scoring anomaly incoming request.

It uses blacklist techniques to detect attacks from incoming requests. It allows you to adjust the aggressiveness level of rule set (Paranoia Level in crs-setup.conf)

Something with real attacks

For multi-user websites or a application targeted by an attacker, the number of alerts generated is extremely large. That will make a lot of mistakes happen to CRS because i see many people after installing ModSecurity and CRS rule set often run in default mode.

We should pay attention to Paranoia Level in CRS to tune it, raising Paranoia Level will turn off CRS default mode when installing, the higher the level setting, the more rules are enforced. It could be called when it was a crazy beast that caused many false positives.

Thinking of out of the box if you want to reduce the number of false positives. If intermix with traces of true attacks, the value that we receive with CRS will be lost. So, we have to remove the false positive to have a cleaner installation for valid queries and prevent a real attacker.

Problems:

– Identify a false positive practical

– Deal with false positives practical

Hundreds of alerts are generated, it's really hard to determine exactly the false positive. We have several ways to do that.

– Understand the application to be protected, it will help us determine request but very suspicious, from malicious requests. But if we are lazy we have to think of another way :))). We can filter each alert and create a data set that only includes false positive warnings. That will help you determine where exactly the attack.

– You can use the IP address to identify basic information about known users, localnet, etc. You may think that the user who successfully authenticated the login is not an attacker (very naive :))) or some other identification method, it depends on the setup and testing process you have.

When identifying a false positive entity, we will have many approaches to avoid repeating similar errors in the future. With CRS, you probably won't revise the rule set because you think it works without interference or works very smoothly and coherently. You will reconfigure the ruleset usage through the ModSecurity directives, which allow you to apply the same changes to future versions of CRS without having to repeat your changes.

I offer simple ways to handle false positives:

– Disable a rule

– Delete an argument from the regex according to a rule

– Disable rule based on request URI at run time

– Remove an argument for a runtime request from a regex according to the rules

But it is clear that disabling a rule will affect the overall detection rate of the rule set. In fact, it takes a bit of experience to make the optimal choice in all cases, the slightest change to ruleset allows for suspicious but not really dangerous pass requests.

Scaling Tuning

To get the best option, try and try. Normally, many people just approach a bunch of alerts and try to work with it, sometimes it is very laborious but the effect is very low.

Example:

```
2019/07/19 22:47:21 [info] 7962#7962: *1 ModSecurity: Warning. Matched "Operator `PmFromFile' wit
```

If we receive hundreds of thousands of alerts every day, our fatigue will increase many times.
To solve this problem earlier, I also wrote some tools to support, it will be public soon for everyone.

Example:

```
{"maturity": "0", "tag": "PCI/6.5.4", "file": "/etc/modsec/rules/custom/crs/rules/REQUEST-930-APP

{"maturity": "0", "tag": "PCI/6.5.2", "file": "/etc/modsec/rules/custom/crs/rules/REQUEST-932-APP

{"maturity": "0", "tag": "attack-generic", "file": "/etc/modsec/rules/custom/crs/rules/REQUEST-94

{"maturity": "0", "tag": "event-correlation", "file": "/etc/modsec/rules/custom/crs/rules/RESPONS
```

I have normalized JSON format to be able to push into the common SIEM or ELK stack. It is easy for us to read and understand the alert information.

For example, the above warning shows that rule 930120 is triggered by having an existing command “ping 10.0.0.1 cat / etc / passwd” in request.
An attacker is trying to execute operating system commands, is the ping command and reads /etc/passwd.

When we transmit an encrypted string or exist some type of data such as hex code, CRS is often confused and gives false positive warnings. CRS will think of this as a clean request as a malicious request that the attacker encrypts information to conceal, such as session ID, etc.

Think

For optimal handling, one should think of a different approach, try different perspectives, not look at different rules, but look in the direction of requests that trigger those rules up.

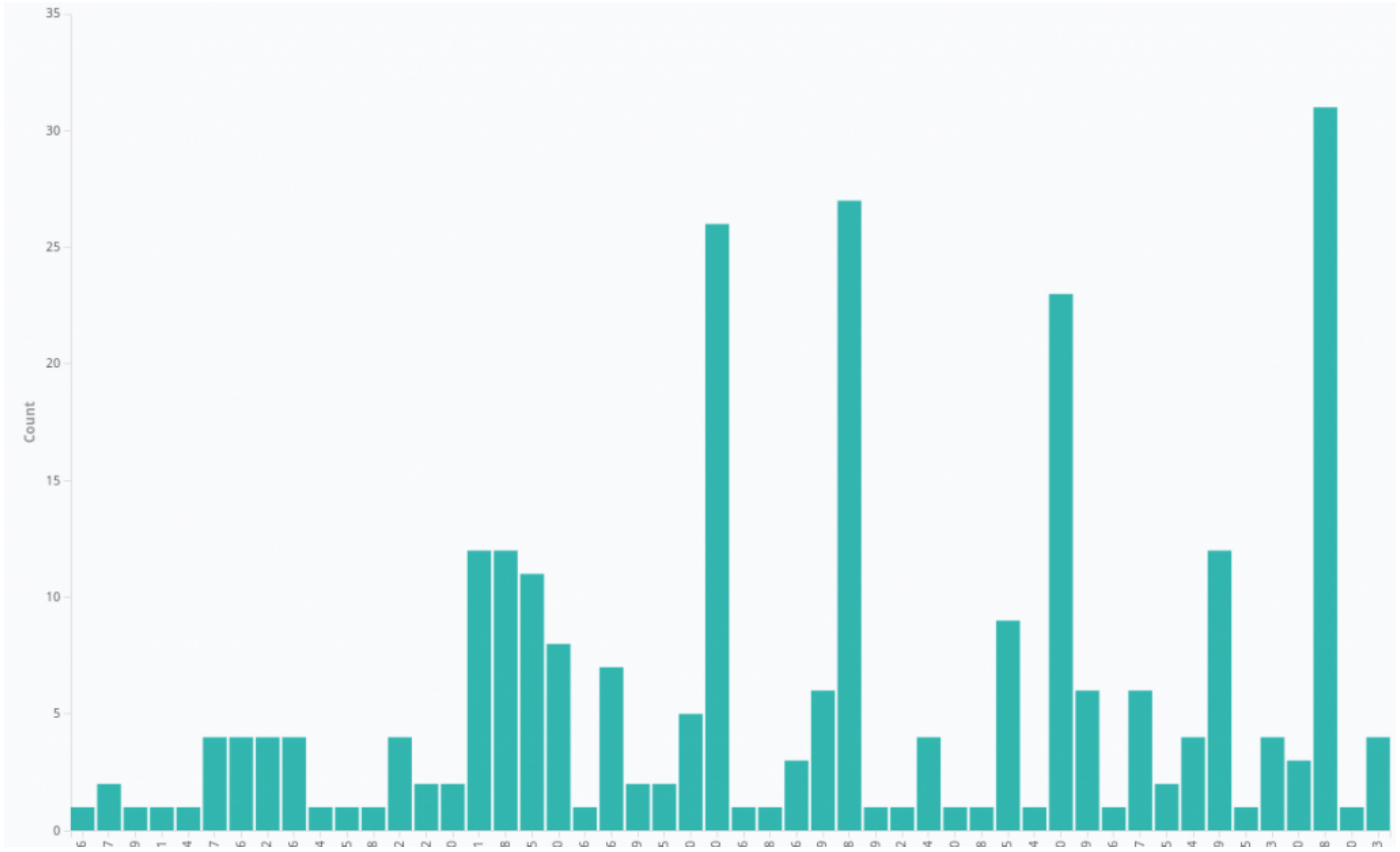
Finding out the anomaly of all requests to the server is important. Not only properties related to scored evaluation, but also those that do not generate any warnings.
We can clearly see its score is zero, but we have to calculate the number of requests in this list to better understand the false positive ratio.
For apache and nginx applications, using apache is easy for ModSecurity to create unusual points of every request right in the access logs. For nginx to use a different technique, add a rule that creates anomalies after the request is made.

```
SecAction \
    "id:980xxx,\
    phase:5,\
    pass,\
    t:none,\
    log,\
    noauditlog,\
    msg:\'Incoming Anomaly Score: %{TX.ANOMALY_SCORE}\'"
```

Result:

```
cat /var/log/nginx/error.log | grep 980xxx | egrep -o "Incoming Anomaly Score: [0-9]+" | cut -b2
0
10
4
5
8
0
12
0
```

```
3
1
0
0
...
```



On the right are requests that are blocked and a warning appears. The highest number of requests with an unusual score is 1, where all requests bypassed through the rules without any alerts are generated.

The higher the column, the higher the score, which can be considered as the most suspicious requests. The lower the column, the lower the score, only activating one or two rules
As is known, the CRS is an unusual set of scoring rules, the first request will pass through the entire rule and the score will start to be calculated.
I will then compare the generated anomalies with the anomaly threshold (default threshold = 5) such as single abnormal critical pricing warning will result in the request being blocked. This is how I want it to be set up securely!
We can change this threshold if we want and in fact it means a lot. For example, once, I put the threshold up to 12,000 (in the actual process I have seen the number of score up to 12,000). I can assure you that no clean requests are blocked by CRS. In such a way I could make adjustments to minimize false positives, balancing them with real systems.
My advice in minimizing is that we should not immediately reduce a shot to 5 in a single shot. Go 10,000->100->50->20->10->5! Final target.

Of course, the way will get in the way, and the solution is like. Take a look at the chart again. If I set the anomaly threshold to 50, I would block requests with a score of 60, if we wanted to lower the limit, I would have to handle these requests better and all other requests got a higher score. with target limit

No need to look at the remaining false false alert results right away because we might be scared of pee =))). We will focus on the minimum number of requests with the highest anomaly score at the start. If I can determine these, I can filter alerts with this high score via unique identification of the alert ModSecurity.

Then use unique IDs to find alerts that score above

```
grep 980xxx /var/log/nginx/error.log  | grep "Incoming Anomaly Score: 60\"" | melunique_id > msg_
```

```
grep -F -f ids /var/log/nginx/error.log | melidmsg | sort | uniq -c | sort -n
  28 941140 XSS Filter - Category 4: Javascript URI Vector
 43 932130 Remote Command Execution: Unix Shell Expression Found
 47 941210 IE XSS Filters - Attack Detected.
 65 941170 NoScript XSS InjectionChecker: Attribute Injection
```

I will adjust these false positive points and position them to lower the abnormal threshold with great certainty that no clean requests will be blocked.

It will take a little while for beginners to do this.
Try and try, that's the way I recommend. Adjusting false positives makes it impossible to lower the threshold further. Look at the requests and analyze them. The number of requests that we have to repeat the test also significantly reduced compared to before

Most of the rules with the highest scores are the same rules with average scores => if I process the first rule set that leads to the highest score, then the second adjustment only has to handle the false positives are not included in requests with the highest score.

The whole method allows the structure to conform to the CRS to minimize the elimination of false positives.
My advice is to be able to give them one by one, usually dealing with five to ten false positives in a given iteration, when performing more processing, take a smaller step.
All work as a game for funs =))) and we have to play as an artist, be like The Joker
Be like an artist and you can adjust your abnormal thresholds in a good and quick way to improve the security of your website system, recommends it significantly only starting with the

same threshold of 20 or lower

In the NUTSHELL

1. Work in blocking mode

2. Originally, set Anomaly threshold to a very high number

– Try try and try

3. Review the request with the highest abnormal score and handle the false positive results it causes

4. Lower the abnormal point threshold to rinse and continue the loop until the points of abnormalities fall to five
Above are some of my small experiences, when I have time I will share some more deep and optimal approaches.

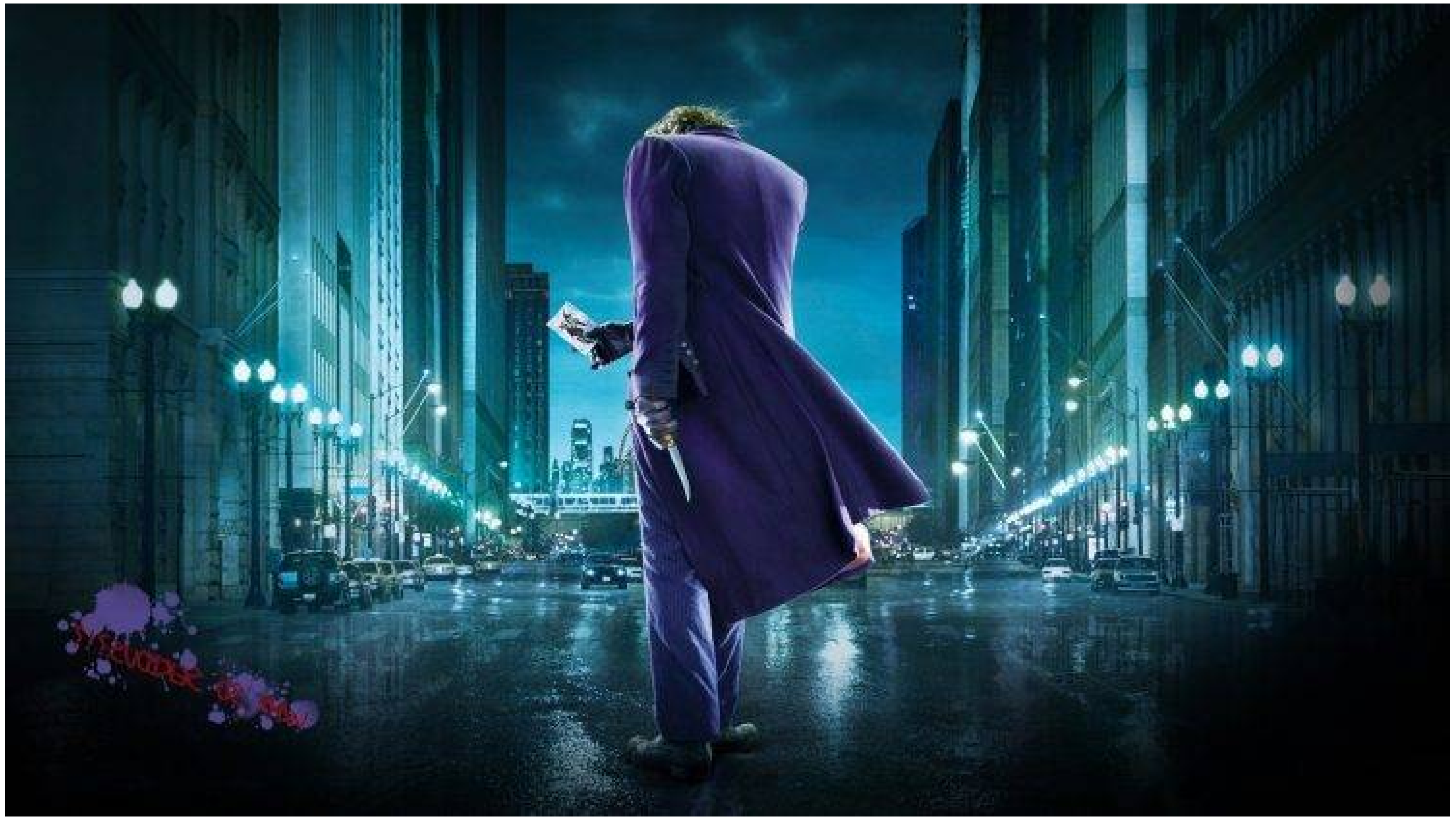
The entire method works for Apache and Nginx!

Refer:
– Thanks for my friends
– https://httpd.apache.org/
– https://www.nginx.com/
– https://coreruleset.org
– Google keyword: "ModSecurity Tutorials"

“The only sensible way to live in this world is without rules.”

I like The Joker.

I like that he’s as mysterious as he is cruel!



Aishee

Previous Post:

[FUN] Bypass XSS Detection WAF

Related Posts:

Pentestit v11 — CUPS Token (5/12)

Pentestit v11 — AD Token (4/12)

Leave a reply:

Your email address will not be published.

Name

Email

Website

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.

About Me

My name is Nguyen Anh Tai. I am an independent security researcher, bug hunter and leader a security team. Security Researcher at CMC INFOSEC. I developed the every system for fun :D. My aim is to become an expert in security and xxx!

Tags

Android aslr attack botnet bounty bug burp suite cache ddos dns export injection Fortify gzip hack hacking hackings HP Fortify SCA html5 html injection iftop iptraf js injection kernel log linux Machine

Learning malware memcached Mirai monitor password pentest pentestit pentest tool Privilege ps ps command security subdomain token unix unprivileged URL malicious waf webstorage webview

Tweets

Aishee RT @cyb3rops: Oh my gosh, this is the noisiest RAT that I've ever seen. Drops .js, .ps1, .bat, .exe, .exe as .jpg, ssh service, TeamViewer,...

about 18 days ago

Aishee RT @axi0mX: EPIC JAILBREAK: Introducing checkm8 (read "checkmate"), a permanent unpatchable bootrom exploit for hundreds of millions of iOS...

about 19 days ago

Aishee 7% through "The Simulation Hypothesis: An MIT..." by Rizwan Virk. Try the book for free: https://t.co/iaDzmZri9Y https://t.co/oq5dvJoUyr

about 4 months ago

Make by Aishee - A blog simple for social