



This repository Search

Explore Gist Blog Help



itpp16



nbs-system / naxsi

Watch 75

Star 507

Fork 77

# configuration

Edit

New Page

John Bohn edited this page on Oct 16, 2013 · 8 revisions

## A quick note on whitelist generation

Whitelist generation process should be iterative.

The first step is usually to generate a bit of legitimate traffic, to raise all the obvious exceptions, then generate a first set of whitelists. Once you put these whitelists in production, the concerned exceptions will not be raised/logged anymore.

Once this is done, learning can continue from your usual traffic. You should then again generate whitelist from this traffic, and repeat until you are satisfied and/or the only exceptions raised are not false positives.

It is a good idea, for the initial learning, to setup naxsi as a reverse proxy to the existing website, and generate legitimate traffic with your browser on it (<http://127.0.0.1>), and/or edit your `/etc/hosts` to make your site point to 127.0.0.1. It will allow to test naxsi on the real site, while generating a first set of whitelists.

## Generating whitelists from logs

Whitelists are created from logs. This can be done either manually, or using an automated tool like [nxutil](#). This page will only cover [nxutil](#) usage. To write rules by yourself, and for overall understanding of naxsi rules & configuration, please see the [rulesyntax whitelists](#) and [naxsilogs](#) pages.

### Pages 24

Find a Page...

[A fail2ban profile for Naxsi](#)[basicsetup](#)[compatibility](#)[configuration](#)[deniedurl](#)[dynamicmodifiers](#)[faq](#)[Home](#)[How to create an Apparmor profile for Naxsi](#)[installation](#)[Knownbugs](#)[libinjection](#)[Naxsi on Windows with nginx](#)[naxsilogs](#)


When the site is simple, manually creating the whitelist is a good solution. [nxutil](#) remains a simple tool, and you should not rely solely on it.

I am the only one generating traffic for a website, and did just a very quick navigation. We can generate whitelists from log files, with -o option :

```
$ nx_util.py -l /tmp/*.log -o
05/30/2013 09:22:16 Deleting old database :naxsi_sig
05/30/2013 09:22:16 List of files :['/tmp/access_main.log', '/tmp/error.log', '/tmp/
##### Optimized Rules Suggestion #####
# total_count:2 (16.67%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
BasicRule wl:1011 "mz:$URL:/|$HEADERS_VAR:cookie";
# total_count:2 (16.67%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
BasicRule wl:1010 "mz:$URL:/|$HEADERS_VAR:cookie";
# total_count:2 (16.67%), peer_count:1 (100.0%) | mysql keyword (|)
BasicRule wl:1005 "mz:$URL:/|$HEADERS_VAR:cookie";
# total_count:1 (8.33%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
BasicRule wl:1011 "mz:$URL:/wp-content/uploads/nbs-system-new-logo.png|$HEADERS_VAR:
# total_count:1 (8.33%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
BasicRule wl:1010 "mz:$URL:/wp-content/uploads/nbs-system-new-logo.png|$HEADERS_VAR:
# total_count:1 (8.33%), peer_count:1 (100.0%) | mysql keyword (|)
BasicRule wl:1005 "mz:$URL:/wp-content/uploads/nbs-system-new-logo.png|$HEADERS_VAR:
# total_count:1 (8.33%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
BasicRule wl:1011 "mz:$URL:/wp-content/themes/nautica/images/nbs-header-cerberhost.p
# total_count:1 (8.33%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
BasicRule wl:1010 "mz:$URL:/wp-content/themes/nautica/images/nbs-header-cerberhost.p
# total_count:1 (8.33%), peer_count:1 (100.0%) | mysql keyword (|)
BasicRule wl:1005 "mz:$URL:/wp-content/themes/nautica/images/nbs-header-cerberhost.p
```

You can see that so far, all the exceptions are located in the cookies (\$HEADERS\_VAR:cookie), on various pages. [nxutil](#) will usually try to factorize rules, but here is not enough traffic to do so.

With -p option, we can force the number of pages to be hit before a rule is factorized :

[naxsivsasscan](#)[Show 9 more pages...](#)**Clone this wiki locally**<https://github.com/nbs-system/naxsi> **Clone in Desktop**

```
nx_util.py -l /tmp/*.log -o -p 1
05/30/2013 11:03:06 Deleting old database :naxsi_sig
05/30/2013 11:03:06 List of files :['/tmp/access_main.log', '/tmp/error.log', '/tmp/
##### Optimized Rules Suggestion #####
# total_count:4 (33.33%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
BasicRule wl:1011 "mz:$HEADERS_VAR:cookie";
# total_count:4 (33.33%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
BasicRule wl:1010 "mz:$HEADERS_VAR:cookie";
# total_count:4 (33.33%), peer_count:1 (100.0%) | mysql keyword (|)
BasicRule wl:1005 "mz:$HEADERS_VAR:cookie";
```

As the match was always in the cookies, and cookies are transmitted at every request, it's safe to assume that we can whitelist those exceptions on the whole site.

Those three rules can be even be written like this :

```
BasicRule wl:1011 "mz:$HEADERS_VAR:cookie";
```

If you enabled naxsi extensive log `naxsilogs#naxsi_exlog`, you will have more detailed output, including sample data that triggered the exception :

```
# total_count:18 (33.33%), peer_count:1 (100.0%) | parenthesis, probable sql/xss
#example (from exlog) : 'csrftoken=...; __utma=96...; __utmb=969...; __utmc=96992031
__utmz=96992031.1369940344.3.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none) '
BasicRule wl:1011 "mz:$HEADERS_VAR:cookie";
```

The percentages that appear in the comments above the exception correspond to :

- The ratio of this exception among all the exceptions triggered
- The ratio of the number of peers that triggered this exception among all peers

Those two indicators are of great help to differentiate false positives from real attacks.

Usually, an exception with a very low percentage indicates a real exception.

On the other hand, if your website doesn't drive much traffic and is publicly exposed, the learning might be biased by the fact that, even if the sea is full of fishes, the ratio of malevolent bots versus legitimate users is so low that hack attempts would be classified as false positives.

One can as well filter events used to generate whitelists :

```
nx_util.py -f "country = FR" -l /var/log/nginx/nginx-blog.memze.ro_error.log -o
```

This will imports and generate whitelists only from events coming from French IPS.

(Requires python-geoiip)

### Applying whitelists

Once generated, you can directly include them in your location, and reload nginx.

If you repeat the same navigation scenario, you will not see exceptions appear anymore.

Naxsi whitelist (along with naxsi configuration), is always **location** specific. As well, naxsi will not handle internal requests.

