NAXSI

A web application firewall for nginx

Thibault Koechlin (nbs-system)

OWASP AppSecEU 15

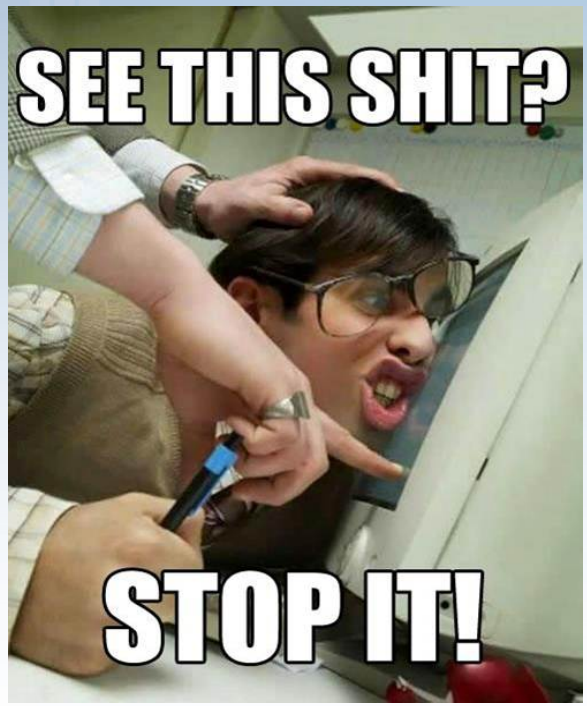
Amsterdam, The Netherlands

Why Naxsi

Web application

Classical IT

Why naxsi



Best mitigation : patch

Not always possible :

- Lack of skill
- Application is too « critical »

Why naxsi

Fallback : WAFs ? (back in 2011 \o/)

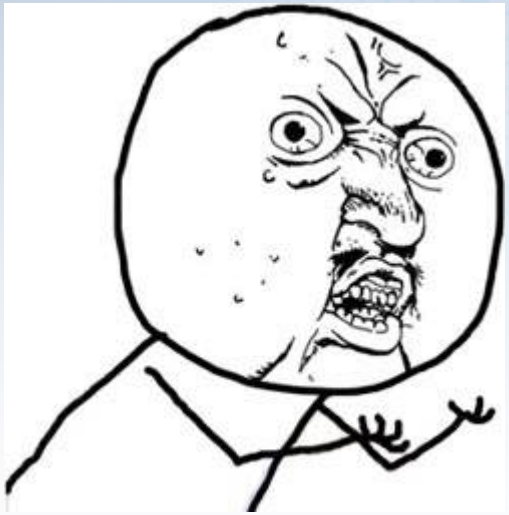
Commercial Wafs :

- Very uneaven
- Not really affordable for small companies with big infrastructures

Opensource WAFs :

- No waf at that time for nginx
- Not found of complex signatures:)

Why naxsi



Why u no protect ?!

As a pentester :

- Web application still the most vulnerable and exposed part of the perimeter

As a hoster :

- Website owners, even when web is at the core of business, lacks awareness … and get owned

As a security « consultant » :

- CISO/Admin still frightened of side effects
- Open-source WAFs only seen in very « tech savy » companies

Introducing naxsi

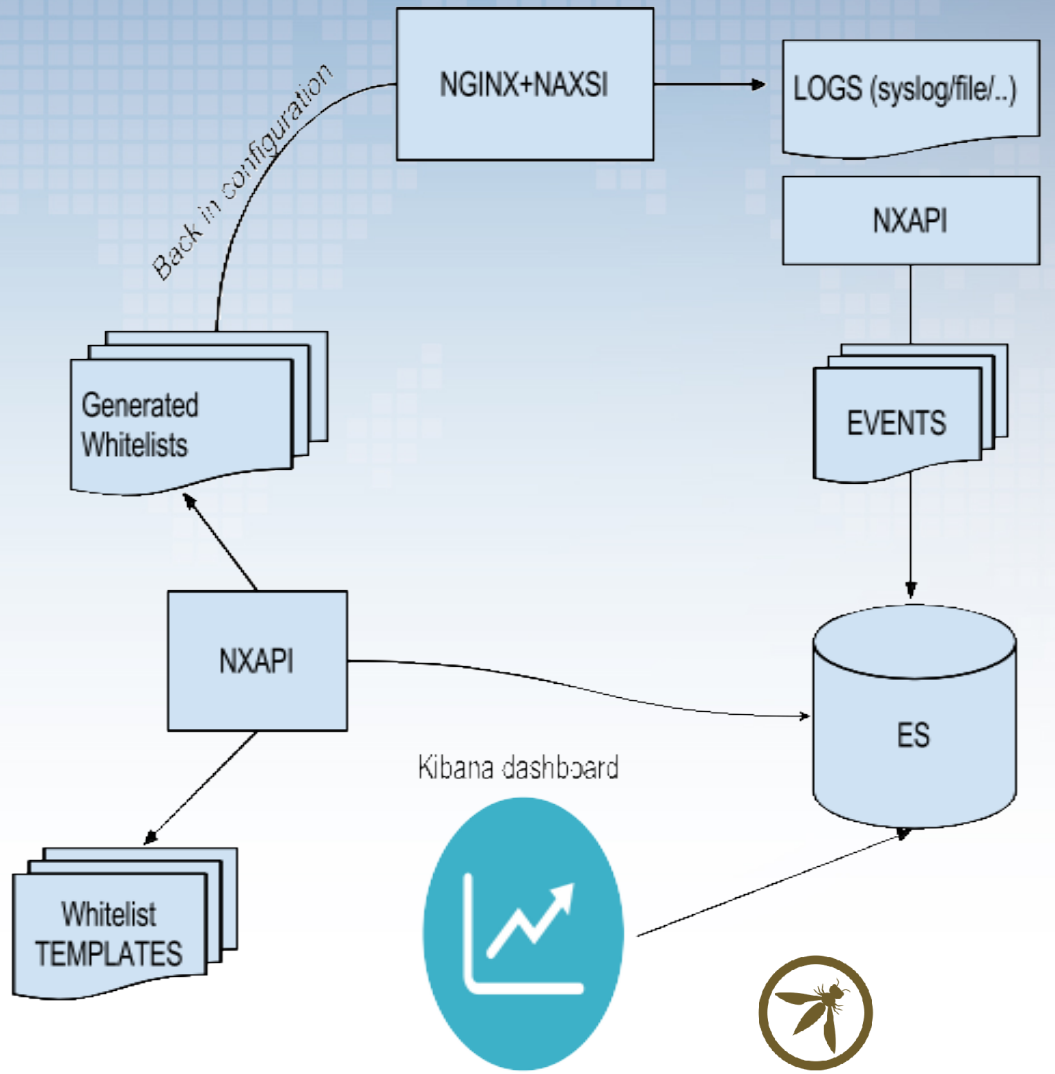
Naxsi – yet another WAF ?

- Rather than detecting « complex » signatures, will focus on « tokens » :
  _ <>()[];='#...
- Tokens presence leads to score increase → action
- Small code base :
  _ core (~4k) C
  _ learning tools (~1.3k) python

Integrates libinjection as well !

Workflow

- Generated logs are pushed into ES
- Data is relevant for monitoring and trafic inspection
- Nxapi helps the admin generate whitelists



Strengths & Weaknesses

Strength	Weaknesses
Good resilience against unknown/obfuscated attacks	Initial learning is needed
Good performances (low mem footprint, minimal runtime processing)	Fast evolving apps requires coordination with releases
No need for updates of « attack » signatures	No « intelligence », unsuitable for some specific cases
Learning process strongly assisted	

Json + x-www-form-urlencoded + multipart/form-data

What does it actually looks like

- Detection rule :

```
MainRule "str:("  "mz:ARGS|URL|BODY|$HEADERS_VAR:cookie" "s:$SQL:4,$XSS:8" id:1010;
```

- CheckRule :

```
CheckRule "$SQL >= 8" BLOCK; #(DROP|LOG)
```

- Whitelist :

```
BasicRule wl:1010 "mz:$HEADERS_VAR:cookie|$URL:/x" ;
```

ARGS

URL

HEADERS

BODY / FILE_EXT

|NAME

Learning & tools

Learning process

- Relying on (ES) injected logs
- Suggests whitelists (nxapi) :
  - Based on templates (application specific)
  - Statistics (number of occurrences, number of peers …)
- Associated events are then « tagged » into database

Learning process

- Naxsi relies on two main modes of operation « learning » / « blocking »
  - During learning phases, exceptions are loggued but not blocked
  - Once learning is over, naxsi can be set to blocking mode (bad trafic is dropped)

More into learning

Learning is the biggest downside, however :

- When it comes to « market » apps, whitelists are very predictable (templates!)
- For home-made apps, several options :
  - Relying on statistics
  - Relying on « trusted » trafic

More into learning

Nxtool templates (dnyamic)

```
{       "_msg" : "Magento checkout page (BODY|NAME)",
        "?uri" : "/checkout/onepage/.*",
        "zone" : "BODY|NAME",
        "id" : "1310 OR 1311"}
```

Naxsi templates (static)

```
BasicRule  wl:1310,1311 "mz:$URL_X:^/checkout/onepage/savebilling/$|BODY|
NAME";
```

More into reporting/visualisation

- <insert cool kibana dashboard cap here>

Tips & Tricks

Tweaks around learning

Combined with nginx scripting :

```
if ($remote_addr = "1.2.3.4") {
 set $naxsi_flag_learning 0;
 set $naxsi_libinjection_sql 1 ;
 }
```

Naxsi on/off

Learning on/off

post_action on/off

extensive_log on/off

Libinjection (sql|xss) on/off

- Learning only for some specifics URI(s)
- Learning only from some IP(s)
- Learning if the visitor fits some critetria

Tweaks around learning

Naxsi can be instructed to drop requests despite learning mode :

- Libinjection

```
CheckRule "$LIBINJECTION_XSS >= 8" DROP;
CheckRule "$LIBINJECTION_SQL >= 8" DROP;
```

- Doxi-rules http://spike.nginx-goodies.com/rules/

```
CheckRule "$UWA >= 8" DROP;
```

Feedback from real life :

- Rules syntax stay very simple :
  - Lowers the risk of breach while playing around rules
  - Lowers the needed skill to manage the rules
- Naxsi itself is very simple :
  - Low ressources
  - Specific cases might become problematic : content legitimately passed in base64

Achievements

- As been tested in several occasions :
  - Real life (a lot)
  - Audited by 3rd parties
  - Challenges
- Used to protect some website under « persistent » attacks
- Used to protect at least one very large website (1Tb+)

What's next,

Q&A

Thanks for your attention !