

Adding Inspection

Attaching application-layer inspection to the reverse proxy. Standing up the edge gave you a proxy that routes and terminates TLS. This page adds the part that examines requests and refuses those identified as malicious, and sets up the principles that makes blocking safe later.

Installing open-appsec

Open-appsec attaches to nginx as a module plus a set of agent services. With nginx already installed from its official repository and pinned, install open-appsec:

```
cd /tmp
wget https://downloads.openappsec.io/open-appsec-install
chmod +x open-appsec-install
sudo ./open-appsec-install --auto --no-email
sudo ./open-appsec-install --download
```

The `--download` step prints the exact commands to install the nginx attachment module for your specific nginx version. Run the printed commands rather than assuming paths, because they are version-specific. They copy the attachment libraries into place and the nginx module itself:

```
# example – use the paths from your own --download output
sudo cp /tmp/open-appsec/nginx_module_<ver>/libosrc_shmem_ipc.so /usr/lib/
sudo cp /tmp/open-appsec/nginx_module_<ver>/libosrc_compression_utils.so /usr/lib/
sudo cp /tmp/open-appsec/nginx_module_<ver>/libosrc_nginx_attachment_util.so /usr/lib/
sudo cp /tmp/open-appsec/nginx_module_<ver>/ngx_cp_attachment_module.so
/usr/lib/nginx/modules/
```

Load the module by adding this at the very top of `nginx.conf`:

```
load_module /usr/lib/nginx/modules/ngx_cp_attachment_module.so;
```

Then install the agent services that do the actual inspection work behind the nginx module:

```
sudo /tmp/open-appsec/openappsec/install-cp-nano-agent.sh --install --hybrid_mode --server
'NGINX Server'
sudo /tmp/open-appsec/openappsec/install-cp-nano-service-http-transaction-handler.sh --install
```

```
sudo /tmp/open-appsec/openappsec/install-cp-nano-attachment-registration-manager.sh --install
sudo systemctl restart nginx
```

`--hybrid_mode` runs open-appsec in standalone mode, managed by the local policy file rather than a cloud console. That keeps management local and self-hosted, consistent with the rest of the environment.

The learn-then-prevent lifecycle

This is the single most important operational fact about running open-appsec: **the machine-learning model must observe real traffic before it can be trusted to block**. Open-appsec scores requests by how anomalous they are relative to what it has learned normal traffic looks like. Before it has learned, it does not have a reliable baseline, and turning on blocking prematurely means legitimate but unusual requests potentially get refused.

So the lifecycle is:

1. **Install and run in a learning posture.** Let it watch real traffic for a period, weeks, not hours, so it sees the normal range of what your applications actually do: large uploads, automation calls, the odd-looking-but-legitimate request patterns each app has.
2. **Review what it would have blocked.** During learning, the logs show what the model flags. This is where you find the false positives before they become outages.
3. **Switch to prevention, one protection at a time.** Once the baseline is solid, enable blocking. From then on, flagged requests are actually refused.

Skipping the learn phase, or switching to prevention too soon, is the path breaking legitimate traffic and ripping out open-appsec in frustration. The learning period is the cost of the model-based approach, and it is what makes the approach work. Budget time to do it right.

Open-appsec expresses this as **modes** on each protection, and the naming is worth understanding because it appears throughout the policy file:

- **detect-learn**: observe and learn, log what would be blocked, block nothing. This is where everything starts.
- **prevent-learn**: block, and keep learning to refine the model. You move a protection here only once you trust its baseline, which is covered on the *Operating It* page.

You move a protection from `detect-learn` to `prevent-learn` when you trust its baseline.

The policy file

The policy lives in `/etc/cp/conf/local_policy.yaml`. Below is the working policy as an **initial deployment**, with every protection in `detect-learn`. It has three parts: the default policy applied to traffic, the practice that defines what protections run and in what mode, and the log trigger.

```
policies:
  default:
    mode: detect-learn          # overall posture: observing and learning, blocking
    nothing
    practices: [appsec-best-practice]  # which practice (protection set) applies
    triggers: [appsec-log-trigger]     # how events are logged
    custom-response: 403-forbidden    # what a blocked client receives, once blocking is
    enabled
    source-identifiers: ""
    trusted-sources: ""
    exceptions:
      - name: allow-backup-sync
        condition:
          source-ip: "203.0.113.10"    # placeholder for a specific trusted source IP
          action: allow                # exempt this source from inspection
    specific-rules: []

practices:
  - name: appsec-best-practice
    openapi-schema-validation:
      configmap: []
      override-mode: detect-learn      # schema validation: learning only
    snort-signatures:
      configmap: []
      override-mode: detect-learn      # signature matching: learning only
    web-attacks:
      max-body-size-kb: 1000000
      max-header-size-bytes: 102400
      max-object-depth: 40
      max-url-size-bytes: 32768
      minimum-confidence: critical     # when promoted, only block at highest confidence
    (see note)
      override-mode: detect-learn      # web attacks: learning only for now
    protections:
      csrf-protection: detect-learn
      error-disclosure: detect-learn
```

```
non-valid-http-methods: true
open-redirect: detect-learn
anti-bot:
  injected-URIs: []
  validated-URIs: []
  override-mode: detect-learn      # anti-bot: learning only
```

A few settings are worth understanding now, even though nothing is enforcing yet, because they govern what happens once you promote protections to blocking:

- `minimum-confidence: critical` on web-attacks is a deliberately conservative choice for when that protection is eventually promoted. Open-appsec will only block a web-attack when its confidence is at the highest level, minimising false positives at the cost of letting lower-confidence suspicious requests through to be logged rather than blocked. For a small environment where a false block locks out a real person and there is no one else on call to fix it, erring toward fewer false positives is the right call. A higher-security posture with a team to tune continuously would lower this threshold and accept more false positives.
- `non-valid-http-methods: true` is a plain boolean rather than a mode: malformed or non-standard HTTP methods are rejected outright, since there is no legitimate reason for them.
- `custom-response: 403-forbidden` defines what a blocked client eventually receives. It has no effect while everything is in `detect-learn`, because nothing is being blocked yet, but it is in place for when protections are promoted.
- **The `exceptions block`** exempts a specific trusted source from inspection entirely. This is sometimes a real necessity, a backup or sync endpoint whose legitimate traffic trips the model, but every exception is a hole in the inspection. Keep each one as narrow as possible (a single source IP, not a range), document why it exists, and revisit it periodically. An exception whose reason no one remembers is a liability.

Applying policy changes

After editing the policy, apply it:

```
open-appsec-ctl --apply-policy
```

Revision #4

Created 2026-07-27 18:06:55 UTC by Chris Landis

Updated 2026-07-27 20:04:06 UTC by Chris Landis