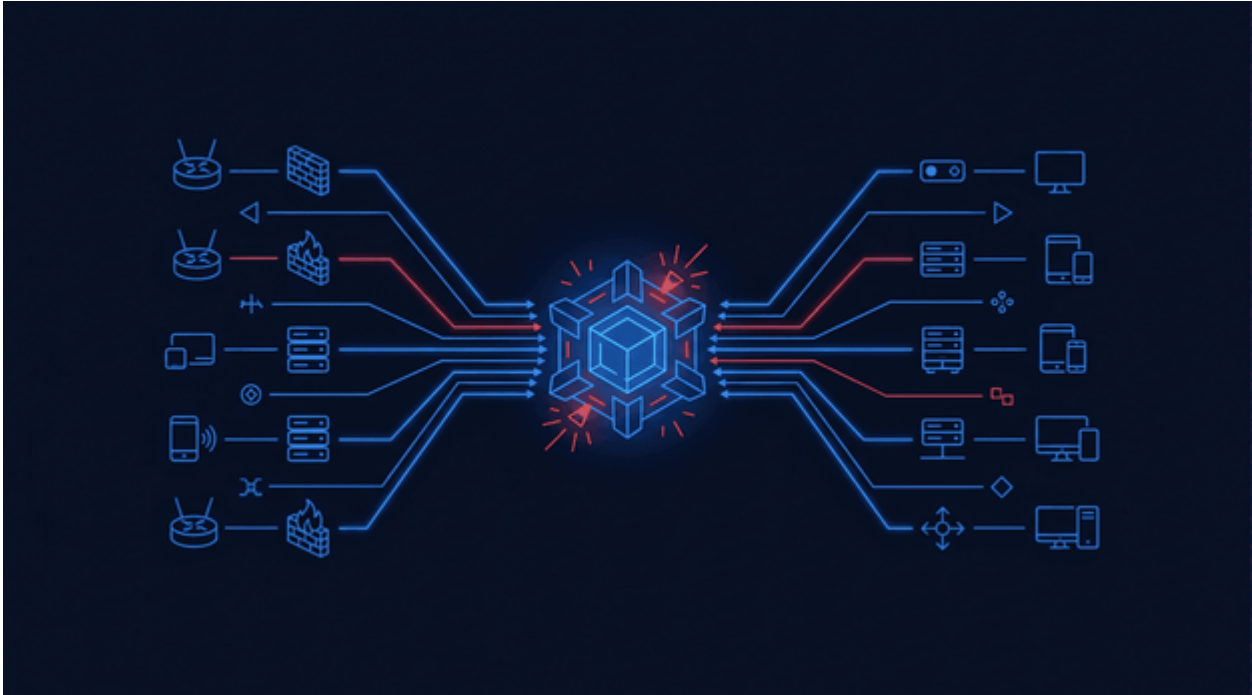


Building a Home Lab SIEM: Wazuh Deployment with Custom Detection Rules

Tue, Mar 10, 9:45 AM EDT · <https://scottslab.io/posts/home-lab-siem-wazuh-custom-detection>

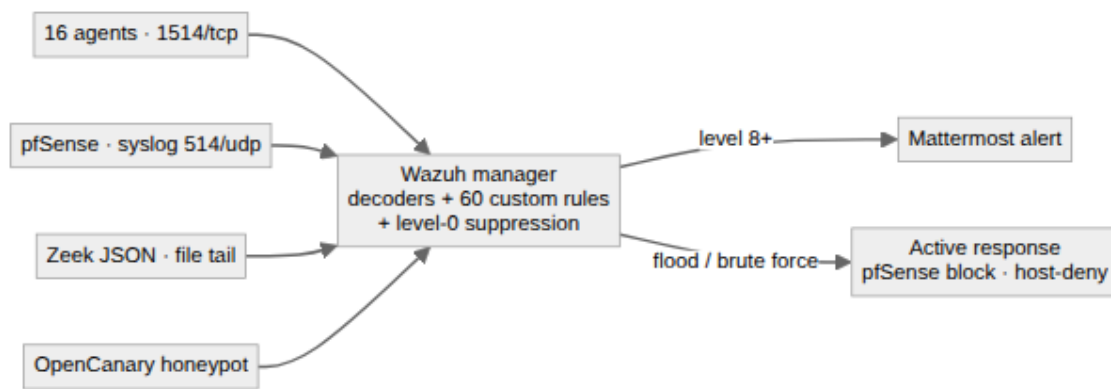


TL;DR

Sixteen hosts were each logging into the void, so everything now lands in one Wazuh 4.14.2 manager: agents, pfSense syslog, Zeek JSON, and a honeypot. About 60 custom rules do the detecting, but the level-0 suppression rules are what made the console usable: hundreds of alerts a day down to under twenty. Two rules hit back through active response, and anything level 8 or above reaches my phone through Mattermost.

Sixteen hosts, and until last winter every one of them logged into the void. The firewall kept its own block log, Zeek wrote connection records nobody read, the honeypot fired into a file, and each server rotated its auth log on a schedule I'd half forgotten. Individually, fine. Together, useless. Attackers don't announce themselves on one box. They leave a smudge on the firewall, a strange DNS query at the resolver, a failed login on the mail server, and unless something is stitching those together, the smudge is all you ever see.

So I stood up Wazuh 4.14.2 on a dedicated VM and pointed everything at it. The goal was never a pretty dashboard. It was correlation and a response loop: see the flood, block the flood, tell me about it, before I've noticed anything is wrong.



How the logs get in

Three ingestion paths, because the sources don't agree on anything. The sixteen agents talk to the manager on 1514/tcp and hand over the richest data: file integrity, command history, auth events. pfSense can't run an agent, so it ships its filter log over syslog on 514/udp. And Zeek writes JSON to disk, which the manager tails as a local file. One SIEM, three front doors.

The firewall log is the awkward one. It arrives as a comma-separated line, not syslog, not JSON, so nothing pulls fields out of it by default. That needs a custom decoder:

```

<decoder name="pfsense-filterlog">
  <prematch>filterlog:</prematch>
</decoder>

<decoder name="pfsense-filterlog-fields">
  <parent>pfsense-filterlog</parent>
  <regex type="pcre2">filterlog:\s*(\d+),.*?,.*?,.*?,(\w+),(\w+),
(\w+),.*?,.*?,.*?,.*?,.*?,(\d+\.\d+\.\d+\.\d+),(\d+\.\d+\.\d+\.\d+),(\d+),
(\d+)</regex>

  <order>rule_number,interface,action,direction,srcip,dstip,srcport,dstport</o
rder>
</decoder>

```

Zeek is the opposite problem. Its logs are already JSON, but `conn.log`, `dns.log`, and `ssl.log` all look identical to Wazuh's built-in JSON decoder, and you can't override that decoder: it grabs anything starting with `{`. So instead of decoding by source, I tell the log types apart in the rules, by

which field is present: `conn_state` means a connection record, `qtype_name` means DNS, `validation_status` means a certificate.

The rules that earn their keep

Sixty-some custom rules ended up spread across three files (Zeek, pfSense, and the honeypot), but they all do one of two things: spot a bad shape once, or spot a bad rhythm over time.

The rhythm rules are the ones I lean on. Port scans, brute force, and floods all have a frequency signature, and Wazuh's frequency/timeframe counters catch them cleanly. The firewall flood rule is the clearest example: count blocks from a single source, and when fifty land in sixty seconds, escalate and hand it to active response:

```
<rule id="100206" level="12" frequency="50" timeframe="60">
  <if_matched_sid>100200</if_matched_sid>
  <same_source_ip />
  <description>pfSense: Flood pattern (50 blocks in 60s) - triggering active
response</description>
  <group>firewall_flood,active_response</group>
</rule>
```

The honeypot rules are the simplest and the highest severity, because the logic is trivial: nothing has a legitimate reason to touch the OpenCanary box. Any connection is level 12, SSH to it is level 13, and three services probed inside five minutes is a level 14. That's not a stray packet, that's someone walking the room.

Cutting the false positives

The first day live, the console was unusable. Hundreds of alerts, nearly all of them my own infrastructure being itself. Cloudflare tunnels look exactly like a long-lived C2 channel. Tailscale's DERP relays trip the same wire. CDN subdomains read like DGA output. None of it is an attack, and all of it drowns the things that are.

The fix is unglamorous: level-0 suppression rules for the known-good. A DNS-tunneling rule fires on any query over fifty characters; a paired level-0 rule right behind it drops the ones that end in a real CDN:

```

<rule id="100115" level="8">
  <if_sid>100110</if_sid>
  <field name="query" type="pcre2">^[a-zA-Z0-9-]{50,}\.</field>
  <description>Zeek: Possible DNS tunneling - query over 50
chars</description>
</rule>

<rule id="100116" level="0">
  <if_sid>100115</if_sid>
  <field name="query" type="pcre2">\.
(cloudfront|akamaized|fastly|azureedge)\.</field>
  <description>Suppress: known CDN long subdomain</description>
</rule>

```

Repeated for each noisy source, that took the console from hundreds of alerts a day to under twenty. The suppression rules are as much of the detection engineering as the detections themselves. A console full of false alarms is a console you stop reading, and then the real alert scrolls past unseen.

Letting it hit back

Detection without response is just note-taking, so two rules don't stop at alerting. The flood rule above calls a firewall-block script that drops the source into pfSense's block table for a day. And Wazuh's built-in SSH brute-force rule (5763) triggers host-deny on the internet-facing agent, banning the attacker for forty-five days:

```

<active-response>
  <command>firewall-block</command>
  <location>local</location>
  <rules_id>100206</rules_id>
  <timeout>86400</timeout>
</active-response>

```

The one thing I'd stress: the active-response script receives the alert as JSON on stdin, and you have to validate the source IP before you act on it. A malformed log that slips a bad value into that field is a self-inflicted block of the wrong address. Parse defensively. Everything at level 8 and up also goes to a Mattermost webhook, so the critical events reach my phone in a few seconds whether or not I'm looking at the dashboard.

What bit me, and what's next

Most of my lost hours came from Wazuh behaving reasonably in ways I didn't expect. `<field>` defaults to `OS_Match`, not `regex`: your pattern silently never matches until you add `type="pcre2"`. `<srcip>` takes exactly one value, so a suppression list of five subnets is five rules, not one. And duplicate rule IDs don't error; the second definition just quietly wins, which means you can carry a detection gap and never be warned about it. `wazuh-logtest` and `wazuh-analysisd -t` are the antidote: test every decoder and rule before you reload, and the guessing goes away.

What's left is mostly coverage: file-integrity monitoring on the config files that matter, MITRE tags on the custom rules so the dashboard maps to something real, and a Windows rule set (PowerShell abuse, WMI persistence) that I keep putting off. The bones are good. The rest is maintenance, which is the part nobody writes blog posts about but everybody actually lives in.

Building your own SIEM? The field-matching and srcip gotchas will bite you eventually. Hope this saves you the debugging time.

Written by Scott S. — Contact: scott@scottschlangen.com — scottslab.io — <https://scottslab.io/posts/home-lab-siem-wazuh-custom-detection>