PatchMon PATCHMON BLOG

How to Monitor FreeBSD Package Updates Across Your Fleet

FreeBSD gets ignored by most patch management tools. Here's how to track package updates, security advisories, and base system patches across your FreeBSD servers.

AUTHOR

PatchMon Team

PUBLISHED

17 March 2026

CATEGORIES

Tutorials, Security

READ ONLINE

https://patchmon.net/blog/freebsd-package-monitoring-guide

Contents

1. FreeBSD's Two-Layer Patching Model
    - Layer 1: The Base System
    - Layer 2: Packages
    - Why This Matters for Monitoring
5. Security Advisories and VuXML
    - FreeBSD Security Advisories (SA) and Errata Notices (EN)
    - VuXML: The Vulnerability Database for Packages
    - Checking Specific Packages
9. The Challenges of Fleet-Scale FreeBSD Monitoring
    - No Built-In Automatic Updates
    - pfSense and OPNsense: FreeBSD Underneath
    - Mixed Environments
    - Output Parsing Is Fragile
14. DIY Approach: A Monitoring Script
    - Limitations of This Approach
16. Centralized Monitoring Options
    - Ansible with FreeBSD Modules
    - Custom Scripts with Centralized Logging
    - PatchMon
20. Best Practices for FreeBSD Fleet Patching
    - 1. Monitor Both Layers, Always
    - 2. Subscribe to FreeBSD-Security-Notifications
    - 3. Run pkg audit Regularly
    - 4. Handle pfSense and OPNsense Separately
    - 5. Test Before You Apply
    - 6. Track the Support Calendar
    - 7. Do Not Forget Jails and bhyve VMs
    - 8. Document Your Repositories
29. Conclusion

If you run FreeBSD in production, you already know the situation. Most patch management tools either ignore FreeBSD entirely or treat it as an afterthought: a checkbox on a features page that leads to a half-working agent that hasn't been updated in two years.

This is a practical guide to monitoring package updates and security patches across a fleet of FreeBSD servers. We will cover the tools FreeBSD gives you natively, the limitations you hit at scale, and the options available for centralized monitoring, including where PatchMon fits in.

FreeBSD's Two-Layer Patching Model

The first thing that trips up anyone coming from Linux is that FreeBSD has two completely separate update mechanisms. This is not a quirk; it is a deliberate design decision that reflects FreeBSD's clean separation between the base system and third-party software.

Layer 1: The Base System

The base system is everything that ships with FreeBSD itself: the kernel, userland utilities, OpenSSL (the base version), and system libraries. This is maintained by the FreeBSD project and updated through `freebsd-update`.

```
# Check for available base system updates
freebsd-update fetch

# Apply fetched updates
freebsd-update install
```

`freebsd-update` works with binary patches. It contacts the FreeBSD update servers, compares your installed base against what is current for your release (say, 14.1-RELEASE), downloads the deltas, and applies them. After a kernel update, you reboot. After userland updates, you may need to restart affected services.

A few things to note:

- `freebsd-update` only works on RELEASE versions. If you track -STABLE or -CURRENT, you are building from source and managing updates yourself.
- It handles both security patches and errata fixes for the base system.
- There is no built-in scheduling. You run it manually or set up a cron job.

Layer 2: Packages

Everything you install with `pkg` (nginx, PostgreSQL, Python, whatever) lives in a separate world. Packages are managed by the `pkg` tool and come from the FreeBSD package repositories (or your own poudriere builds).

```
# Update the local package repository catalogue
pkg update

# List packages with available upgrades
pkg upgrade --dry-run

# Actually upgrade
pkg upgrade
```

This is closer to what Linux admins expect, but it is still its own thing. The FreeBSD package repositories are built from the ports tree at specific branch points, and updates arrive on a different cadence than base system patches.

Why This Matters for Monitoring

Any monitoring solution that only checks one of these layers is giving you an incomplete picture. A server can have fully up-to-date packages but be running a base system with a known OpenSSL vulnerability, or vice versa. You need to track both.

On Linux, `apt`, `dnf`, or `pacman` handle essentially everything on the system. On FreeBSD, you are always dealing with two independent update streams. This is something most cross-platform tools get wrong because they try to map FreeBSD onto the Linux model.

Security Advisories and VuXML

FreeBSD has a well-structured security advisory system. Understanding it is important for knowing what you are actually monitoring for.

FreeBSD Security Advisories (SA) and Errata Notices (EN)

The FreeBSD Security Team issues two types of notices:

Security Advisories (SA): Vulnerabilities in the base system. These are what `freebsd-update` patches. Each SA has an identifier like `FreeBSD-SA-26:03.openssl` and covers a specific CVE or set of CVEs.

Errata Notices (EN): Non-security bugs in the base system that are significant enough to warrant a patch outside the normal release cycle. Same update mechanism, lower urgency.

Both are published on the FreeBSD Security Advisories page (https://www.freebsd.org/security/advisories/) and sent to the `freebsd-security-notifications` mailing list.

VuXML: The Vulnerability Database for Packages

For third-party packages, FreeBSD uses VuXML, the Vulnerability and eXposure Markup Language database. This is an XML database maintained in the FreeBSD ports tree that maps known vulnerabilities to affected package versions.

The `pkg audit` command checks your installed packages against this database:

```
# Update the vulnerability database and audit installed packages
pkg audit -F
```

The `-F` flag fetches the latest VuXML data before running the audit. Without it, `pkg audit` uses whatever local copy it has, which may be stale.

Sample output:

```
curl-8.6.0 is vulnerable:
  curl: TLS certificate check bypass with mbedTLS
  CVE: CVE-2024-XXXXX
  WWW: https://vuxml.FreeBSD.org/freebsd/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.html

1 problem(s) in 1 installed package(s) found.
```

Each VuXML entry links to the relevant CVE and provides the affected version ranges. The database is updated frequently, usually within hours of a new advisory being published for a port.

Checking Specific Packages

If you want to check whether a specific package has known issues:

```
# Audit a specific package
pkg audit curl

# List all vulnerable packages with full details
pkg audit -F -r
```

The `-r` flag includes the package repository as a data source, which can catch issues that are fixed in the repo but not yet installed on your system.

The Challenges of Fleet-Scale FreeBSD Monitoring

Running `pkg audit -F` and `freebsd-update fetch` on a single server is straightforward. Doing it across 20, 50, or 200 FreeBSD servers, and making sense of the results, is where things get difficult.

No Built-In Automatic Updates

Linux distributions have `unattended-upgrades` (Debian/Ubuntu) and `dnf-automatic` (RHEL/Fedora) for automatic security patching. FreeBSD does not ship anything equivalent. You can set up cron jobs, but there is no standard, well-tested framework for unattended FreeBSD updates that handles reboots, service restarts, and failure recovery.

The closest you get out of the box is:

```
# In /etc/crontab
@daily root freebsd-update cron
```

`freebsd-update cron` is designed for cron use; it adds a random delay to avoid thundering herd problems on the update servers. But it only fetches; it does not install. The installation step is deliberately left manual because base system updates can require reboots, and unattended reboots of production servers are generally not what you want.

pfSense and OPNsense: FreeBSD Underneath

If you run pfSense or OPNsense firewalls, you are running FreeBSD. But these appliances have their own update mechanisms layered on top, and you cannot just SSH in and run `pkg upgrade` without potentially breaking the appliance.

pfSense uses its own `pfSense-upgrade` tool. OPNsense uses `opnsense-update`. Both rebuild from a curated subset of FreeBSD packages. This means your fleet monitoring needs to understand that not all FreeBSD systems update the same way.

Mixed Environments

In most organizations, FreeBSD is the minority OS. You might have 200 Linux servers, 50 Windows servers, and 15 FreeBSD boxes running your firewalls, storage (TrueNAS), or specific workloads. The temptation is to handle FreeBSD as a special case: a couple of manual checks, maybe a wiki page someone last updated in 2024.

This is how FreeBSD servers end up running unpatched for months. Not because anyone decided to skip patching, but because the FreeBSD hosts were never integrated into whatever patch monitoring the rest of the fleet uses.

Output Parsing Is Fragile

Both `freebsd-update` and `pkg audit` produce human-readable output, not structured data. If you want to feed their results into a monitoring system, you are parsing text output. Text output formats are not guaranteed to be stable across FreeBSD versions, and edge cases (partial updates, interrupted fetches, custom repositories) produce output that may not match your parsing assumptions.

`pkg` does support some structured output options:

```
# JSON output for installed packages
pkg query -a '%n %v'

# Check for available upgrades in a scriptable way
pkg version -vRL=
```

But `freebsd-update` has no structured output mode at all.

DIY Approach: A Monitoring Script

Before reaching for external tools, here is a practical script that covers the basics. This is the kind of thing most FreeBSD admins start with.

```
#!/bin/sh
# freebsd_patch_check.sh - Check for outstanding updates and vulnerabilities
# Run via cron, send output to your monitoring system

HOSTNAME=$(hostname)
REPORT_FILE="/var/tmp/patch_report_$(date +%Y%m%d).txt"

echo "=== Patch Report for ${HOSTNAME} ===" > "${REPORT_FILE}"
echo "Date: $(date)" >> "${REPORT_FILE}"
echo "FreeBSD: $(freebsd-version)" >> "${REPORT_FILE}"
echo "" >> "${REPORT_FILE}"

# Base system updates
echo "--- Base System Updates ---" >> "${REPORT_FILE}"
if freebsd-update fetch 2>&1 | grep -q "No updates needed"; then
  echo "Base system is up to date." >> "${REPORT_FILE}"
else
  echo "Base system updates are available." >> "${REPORT_FILE}"
  echo "Run: freebsd-update fetch install" >> "${REPORT_FILE}"
fi
echo "" >> "${REPORT_FILE}"

# Package updates
echo "--- Package Updates ---" >> "${REPORT_FILE}"
pkg update -q
UPGRADES=$(pkg upgrade --dry-run 2>&1)
UPGRADE_COUNT=$(echo "${UPGRADES}" | grep "^[[:space:]]" | wc -l | tr -d ' ')

if [ "${UPGRADE_COUNT}" -gt 0 ]; then
  echo "${UPGRADE_COUNT} package(s) have updates available:" >> "${REPORT_FILE}"
  echo "${UPGRADES}" >> "${REPORT_FILE}"
else
  echo "All packages are up to date." >> "${REPORT_FILE}"
fi
echo "" >> "${REPORT_FILE}"

# Security audit
echo "--- Security Vulnerabilities ---" >> "${REPORT_FILE}"
AUDIT=$(pkg audit -F 2>&1)
VULN_COUNT=$(echo "${AUDIT}" | grep "problem(s)" | awk '{print $1}')

if [ -n "${VULN_COUNT}" ] && [ "${VULN_COUNT}" -gt 0 ]; then
  echo "ALERT: ${VULN_COUNT} vulnerable package(s) found:" >> "${REPORT_FILE}"
  echo "${AUDIT}" >> "${REPORT_FILE}"
else
  echo "No known vulnerabilities in installed packages." >> "${REPORT_FILE}"
fi

# Output the report (for cron email or log collection)
cat "${REPORT_FILE}"
```

Set this up in cron:

```
# /etc/crontab
0 6 * * * root /usr/local/bin/freebsd_patch_check.sh
```

Limitations of This Approach

This script works. Many FreeBSD admins run something like it. But it has real limitations:

No central dashboard. You get individual reports per server, delivered via cron email or syslog. Correlating across a fleet means reading emails or grep-ing through logs.

No historical tracking. You know the current state, but not when a vulnerability was first detected, how long it was open, or what the patching trend looks like over time.

Alert fatigue. If you have 30 FreeBSD servers, that is 30 daily emails. Most days, nothing has changed. You start ignoring them.

Parsing is brittle. The grep and awk commands in this script work for the common cases. A FreeBSD version change, a new output format from `pkg`, or an unusual error message can break the parsing silently.

No pfSense/OPNsense support. This script assumes standard FreeBSD with `pkg` and `freebsd-update`. Appliance-based FreeBSD systems need different handling.

For a handful of servers that you personally manage, this approach is fine. For anything larger, or in an environment with compliance requirements, you need something more structured.

Centralized Monitoring Options

Ansible with FreeBSD Modules

If you already use Ansible, it has a `community.general.pkgng` module for FreeBSD package management and you can run shell commands for `freebsd-update` and `pkg audit`.

```
# playbook: freebsd_patch_audit.yml
- hosts: freebsd_servers
  become: true
  tasks:
 - name: Update pkg catalogue
  community.general.pkgng:
    name: "*"
    state: latest
    cached: false
  check_mode: true
  register: pkg_updates

 - name: Run pkg audit
  command: pkg audit -F
  register: pkg_audit_result
  failed_when: false
  changed_when: false

 - name: Check freebsd-update
  command: freebsd-update fetch
  register: freebsd_update_result
  failed_when: false
  changed_when: false
  environment:
    PAGER: cat

 - name: Report results
  debug:
    msg: |
    Host: {{ inventory_hostname }}
    Package updates available: {{ pkg_updates.changed }}
    Vulnerabilities: {{ pkg_audit_result.stdout_lines | select('match',
'.*problem.*') | list }}
    Base updates: {{ 'Updates available' if 'No updates needed' not in
freebsd_update_result.stdout else 'Up to date' }}
```

This is a solid approach if Ansible is already in your stack. The main limitation is that Ansible gives you point-in-time checks, not continuous monitoring. You run the playbook, you see results, but there is no persistent state, no dashboard, and no automatic alerting when a new vulnerability appears.

Custom Scripts with Centralized Logging

A more involved DIY approach is to push the output of your monitoring script into a centralized logging system (ELK, Graylog, Loki, or even a simple syslog server). Structure the output as JSON and you can build dashboards.

```
# Structured output variant
pkg audit -F 2>/dev/null | \
  logger -t "pkg-audit" -p local0.warning
```

This works, but you are building and maintaining a bespoke monitoring system. Every new edge case or output format change becomes your problem.

PatchMon

Yes, this is the PatchMon blog. Yes, we are about to tell you PatchMon is good at this. In our defence, we built FreeBSD support because we actually run FreeBSD, not because a product manager saw a checkbox on a competitor comparison table.

PatchMon is one of the few patch monitoring tools with a native FreeBSD agent. Not a Linux agent that might compile on FreeBSD, but an agent that is built for FreeBSD with precompiled binaries for amd64, i386, arm64, and arm, and that understands both `pkg` and `freebsd-update` as first-class update sources. You can run it on PatchMon Cloud (https://patchmon.net/pricing) as a managed SaaS or self-host the AGPLv3 Community Edition, with the same FreeBSD agent on either side.

The PatchMon agent on FreeBSD monitors both layers:

- **Package updates:** Tracks installed package versions against the repository, reports available upgrades, and checks `pkg audit` data for known vulnerabilities.
- **Base system patches:** Monitors for available `freebsd-update` patches, including security advisories and errata notices.

Both streams feed into the same dashboard as your Linux and Windows servers, so you get a single view of your entire fleet's patch status regardless of OS. If you run a mixed environment where FreeBSD boxes sit alongside Linux servers, this eliminates the "FreeBSD is a special case we handle separately" problem.

Best Practices for FreeBSD Fleet Patching

Regardless of what tools you use, these practices apply to any FreeBSD fleet.

1. Monitor Both Layers, Always

Do not just track packages and ignore the base system, or vice versa. A base system OpenSSL vulnerability is just as exploitable as a vulnerable nginx package. Your monitoring must cover `freebsd-update` and `pkg` on every host.

2. Subscribe to FreeBSD-Security-Notifications

The freebsd-security-notifications (https://lists.freebsd.org/subscription/freebsd-security-notifications) mailing list is low-volume and high-signal. Every Security Advisory and Errata Notice goes here. This is your early warning system; you should know about a new SA before your monitoring tools report it.

3. Run pkg audit Regularly

At minimum, run `pkg audit -F` daily via cron. The VuXML database updates frequently, and a package that was clean yesterday might have a new advisory today.

4. Handle pfSense and OPNsense Separately

If your fleet includes pfSense or OPNsense appliances, do not try to manage them with the same tools and processes as your standard FreeBSD servers. Use their respective update mechanisms and track their versions against the vendor's security advisories. These appliances follow their own release and patching schedules.

5. Test Before You Apply

FreeBSD's conservative approach to updates means that patches are generally safe, but "generally" is not "always." For production systems, maintain a staging environment (even if it is a single jail or VM running the same version) where you apply updates first.

6. Track the Support Calendar

FreeBSD releases have defined end-of-life dates. A 13.2-RELEASE server that is not receiving security patches is not "stable"; it is abandoned. Know when your running versions go EOL and plan upgrades accordingly. The FreeBSD Security page (https://www.freebsd.org/security/) publishes the supported versions and their expected end-of-life dates.

7. Do Not Forget Jails and bhyve VMs

If you use FreeBSD jails, each jail has its own package set that needs independent monitoring. A host system can be fully patched while its jails run vulnerable packages. The same applies to bhyve VMs. Your monitoring needs to cover every FreeBSD instance, not just the bare-metal hosts.

For jails specifically:

```
# Audit packages inside a specific jail
pkg -j mywebapp audit -F
```

8. Document Your Repositories

If you run custom package repositories built with poudriere, document which ports tree branch they track and how often they rebuild. A custom repository that has not been rebuilt in three months is effectively three months behind on security patches, even if the servers running against it show "all packages up to date."

Conclusion

FreeBSD's patching model is different from Linux, but it is not harder - it is just less well-served by third-party tooling. The two-layer system of base updates and package management is a clean design that gives you fine-grained control. The VuXML database and `pkg audit` provide solid vulnerability checking. The FreeBSD Security Team's advisory process is thorough and well-documented.

The real challenge is maintaining visibility across a fleet, especially a mixed-OS fleet where FreeBSD is the minority platform. And this is where most tools fail you - they either do not support FreeBSD at all, or they support it the way airlines support vegetarians: technically, with minimal enthusiasm.

PatchMon is one of the few tools that treats FreeBSD as a first-class platform. The agent monitors both `pkg` and `freebsd-update`, reports into the same dashboard as your Linux and Windows hosts, and gives you a single view of your entire fleet's patch status. No separate tooling for your FreeBSD boxes. No blind spots. Try it on PatchMon Cloud (https://patchmon.net/pricing), our managed SaaS (14-day trial, billed per host that actually checks in), or self-host the AGPLv3 Community Edition (https://patchmon.net/open-source) on your existing FreeBSD or Linux infrastructure.

Your FreeBSD servers deserve the same visibility as everything else in your fleet. Make sure they get it.

PatchMon

ABOUT PATCHMON

The open source Linux patch management platform

PatchMon gives sysadmins one dashboard for patching across Linux and FreeBSD fleets, with Windows supported alongside. Run it as a managed SaaS on PatchMon Cloud (per-host billing, 14-day trial, no fleet minimum) or self-host the AGPLv3 Community Edition on your own infrastructure.

Start a trial: patchmon.net/pricing