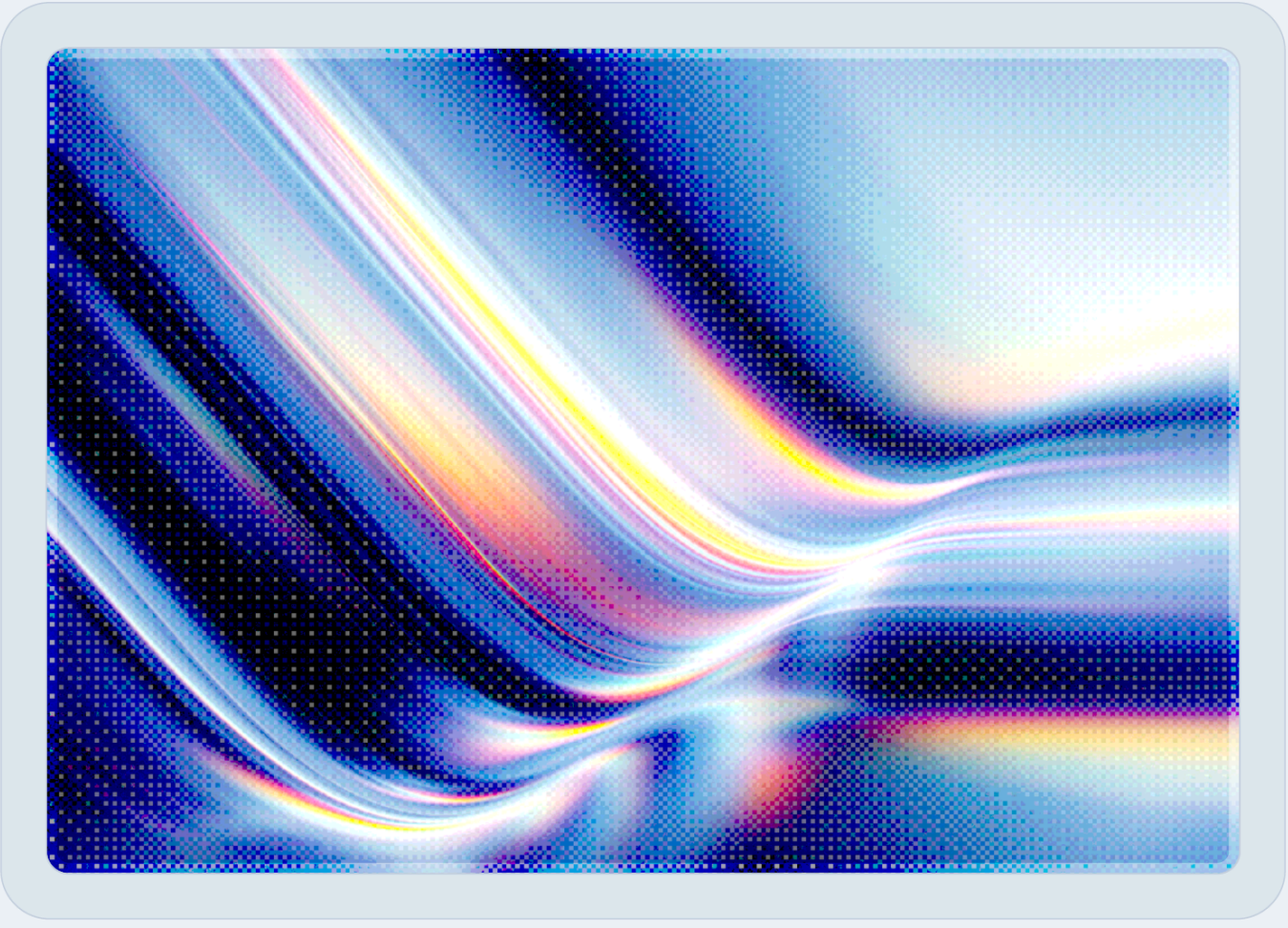
CSAI Foundation | Cloud Security Alliance

Any Long Password Works: CVE-2026-123158 and the Alpine/musl Container Estate

OpenSSH Password Authentication Bypass on musl Without PAM: Security Implications and Guidance

2026-11-07

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

CVE-2026-123158, which its discoverer calls "*sshole," is an authentication bypass in OpenSSH's sshd. It affects Linux systems where sshd runs on musl libc without PAM, which is the default build of Alpine Linux's `openssh-server` package. On an affected system, an attacker who sends a password between 257 and 1,024 bytes long can log in to any account whose shadow password field is `*`. That is the field Alpine and many administrators use to mark an account as having no password. MITRE scores the flaw CVSS 3.1 8.1 (High). OpenSSH 10.6p2 and patched Alpine packages for every branch from 3.16 through edge fixed it on November 5, 2026 [1][2][4].

Exposure requires several conditions at once: sshd on musl without PAM, `PasswordAuthentication` enabled (the default), at least one account whose password field is `*`, and at least one other local account with a real SHA-256 or SHA-512 hash. Alpine 3.16 and later set root's field to `*`, but root is exploitable only where `PermitRootLogin yes` is configured. That setting is common in development and legacy container images, but it is not the OpenSSH default of `prohibit-password` [4]. Non-root `*` accounts are exposed regardless of root-login settings. Alpine's tiny-cloud images ship an `alpine` user with passwordless `doas`, which offers a path to root once a salt-donor hash is present on the system, although Alpine notes these images are safe on first boot [4][5]. Where the conditions hold, exploitation is deterministic and requires no special tooling. Alpine's advisory says the attack works over a single connection on every attempt, and the discoverer's write-up and public lab repository document the technique. No in-the-wild exploitation had been reported as of publication [4][5][6][8].

Remediation has two parts that are easy to get wrong. Updating the package is not enough: running sshd daemons must be restarted, and container images that bundle the old package must be rebuilt from a fixed base and redeployed [4][7]. Where an immediate update is not possible, disabling password authentication fully mitigates the flaw [4]. Finally, the discoverer reports finding the bug with "agent swarms," and the OpenSSH project says it will release more often because security bugs found by AI tools are being independently rediscovered by other researchers [2][5]. Taken together, these points suggest that defenders should plan for less time between disclosure and exploitation.

Background

The flaw comes from two components that each behave reasonably on their own. The POSIX `crypt()` function is supposed to return a null pointer when it cannot hash a password. musl, the lightweight C library used by Alpine Linux, OpenWrt, and many Buildroot and Yocto embedded builds, has instead returned the string `"*"` on failure since August 2012. musl added SHA-512 crypt weeks later, and its SHA-256 and SHA-512 implementations reject passwords longer than 256 bytes as a resource-exhaustion safeguard. Meanwhile, Unix systems have long used `*` in the password field to mark an account that cannot log in with a password, a convention that dates to early BSD [5][6]. When `crypt()` fails and returns `"*"` and sshd compares that string to a stored `"*"`, the two match.

A further step is needed to reach musl's failure path at all, and it comes from OpenSSH's 2016 fix for CVE-2016-6210. That vulnerability let attackers enumerate valid usernames by measuring how long sshd took to reject a password, because hashing a real password took noticeably longer than rejecting a missing one [9]. To equalize timing, OpenSSH's `pick_salt()` routine hashes the candidate password with a salt borrowed from another account on the system, even when the target account has no usable hash. If another local account has a `$5$` (SHA-256) or `$6$` (SHA-512) hash, a password over 256 bytes sent to a `*` account goes through musl's SHA-crypt code, fails, and returns `"*"`, which matches the stored field [5][7]. The upper bound of 1,024 bytes comes from sshd's own maximum accepted password length, defined as `MAX_PASSWORD_LEN` in the OpenSSH source [3]. The CVE record describes the vulnerable window as "between 256 and 1024 characters," so the exact bounds should be treated as approximate where multibyte input is involved [1].

The history of Alpine's root account explains why this matters especially for containers. In 2019, CVE-2019-5021 showed that official Alpine Docker images had shipped with an empty root password [10]. Alpine then locked root with `!`, which, according to the discoverer's timeline, broke key-based SSH for root on non-PAM builds. Alpine 3.16 (2022) switched root to `*`, restoring key-based logins but recreating the exposure described here [5]. The bug was reported by Özgün Kültekin of Wakefield, who describes building OpenSSH in glibc, musl, PAM, and non-PAM configurations and running about 17,600 unusual login attempts against them, largely using AI agents. Only the musl, non-PAM build accepted logins it should have refused [5][8]. The table below summarizes how these decisions accumulated over fourteen years.

Date	Event
Aug 2012	musl `crypt()` begins returning `"*"` on failure; SHA-512 crypt added with a 256-byte password limit [5]

Date	Event
2016	OpenSSH adds fake-salt hashing to fix CVE-2016-6210 user enumeration [5][9]
2019	CVE-2019-5021: Alpine Docker images ship with an empty root password; Alpine locks root with `!` [5][10]
2022	Alpine 3.16 changes root's password field to `*` [5]
Nov 5, 2026	OpenSSH 10.6p2 released; Alpine publishes fixed packages and advisory; CVE-2026-123158 published [1][2][4]

Security Analysis

Conditions for Exploitation

Exploitation requires a narrow set of conditions, but in common Alpine deployments several of them are met by default. Alpine's advisory lists them precisely, and the table below summarizes which configurations are exposed and which are not [4]. All conditions in the "exposed" column must hold at once. Any single item in the "not exposed" column is enough to defeat the attack, with the exception of the root-login row, which protects only the root account.

Factor	Exposed	Not exposed
sshd build	`openssh-server` (no PAM), or `openssh-server-pam` with `UsePAM no`	`openssh-server-pam` with `UsePAM yes`; any glibc-based distribution
Password authentication	`PasswordAuthentication yes` (default)	`PasswordAuthentication no`
Target account	Password field is `*` (root on Alpine 3.16+, key-only users, directory-service users, tiny-cloud `alpine` user)	Locked with `!`, or holds a real hash

Factor	Exposed	Not exposed
Root targeting	`PermitRootLogin yes`	`PermitRootLogin prohibit-password` (OpenSSH default) or `no`; does not protect non-root accounts
Salt donor	At least one local account with a `$5$` or `$6$` hash	No `$`-style hashes on the system
Attacker input	Password of 257–1,024 bytes	Any shorter password

The bypass does not let an attacker log in to accounts that have real password hashes [8]. The vulnerable accounts are the ones administrators believed were closed to password login, so the flaw is an authentication logic failure, not a cryptographic or memory-safety defect [7]. MITRE classifies it as CWE-394, Unexpected Status Code or Return Value. The vector `AV:N/AC:H/PR:N/UI:N` reflects network reachability without credentials, with the high attack complexity presumably capturing the account-configuration preconditions rather than any unreliability in the exploit itself [1]. Once those preconditions hold, Alpine describes exploitation as working over one connection, every time, with no retries [4]. The discoverer's proof of concept uses two password prompts within that connection, a wrong attempt first and the long password second [5].

Alpine, Orca, and the discoverer each identify several account types as targets [4][5][6]. Root on Alpine 3.16 and later is exposed where `PermitRootLogin yes` is set, while the OpenSSH default, `prohibit-password`, blocks password logins to root [4]. Orca's write-up describes root access on a default Alpine install, but this conflicts with Alpine's own advisory, which this note treats as authoritative [4][6]. Key-only service and deployment accounts and directory-backed users from nslcd, winbind, or sssd are exposed regardless of root-login settings. In Alpine's official tiny-cloud images, the `alpine` user has passwordless `doas`, which the researcher says gives effective root from a single connection [5][8]. Alpine notes, however, that these images are safe on first boot and become exploitable only once a `$6$` hash is added, and that images configured through cloud-init write `PasswordAuthentication no` [4].

Why the Container Estate Deserves Particular Attention

Most production containers do not run sshd, and those images are not affected [7]. The risk is concentrated in a specific minority of images: development and CI containers, "pet" containers treated as lightweight virtual machines, bastion and jump-host images, SFTP and file-transfer containers, and appliance-style images built on Alpine for its small footprint. Our inference, not drawn from any published survey, is that these images may often meet the salt-donor condition by accident. A common Dockerfile pattern sets one user's password with `chpasswd` (creating a `$6$` hash) and inherits root's `*` from the Alpine base. Those two choices, each reasonable on its own, complete the exploit preconditions, and the same images are also where `PermitRootLogin yes` is most likely to have been set for convenience.

Three properties of container estates make remediation harder than on traditional hosts. First, `apk upgrade` inside a running container fixes only that ephemeral instance. The vulnerable package stays in the image and returns on the next restart or scale-out, so the fix must be a rebuild from a patched base and a redeploy [7]. Second, Alpine's advisory groups 3.21 and older as releases that are no longer supported (3.21 reached end of support on November 1, 2026), yet fixed packages were still issued for 3.16 through 3.21 [4]. Alpine's decision to patch these branches suggests older images remain in circulation, and teams that pin old base tags may not pull fixes automatically. Third, image scanners that match on package version should flag the vulnerability once their feeds include Alpine's advisory data, but they cannot tell whether a given image actually runs sshd, enables password authentication, or holds a `*` account. Prioritization therefore needs runtime context.

The same flaw extends beyond Alpine. The discoverer [5], Orca Security [6], and OX Security [7] identify OpenWrt, Buildroot, and Yocto builds as potentially affected wherever sshd is built against musl without PAM. These are researcher assessments, not vendor-confirmed findings from those projects. Edge routers and embedded appliances that run OpenSSH (as opposed to Dropbear) with password logins enabled should nonetheless be inventoried, because these devices are often patched far more slowly than cloud workloads.

AI-Accelerated Discovery and the Shrinking Patch Window

This vulnerability is relevant to the AI Safety Initiative beyond its container impact. The discoverer reports that the systematic exploration of build and configuration combinations that surfaced the bug was carried out largely by "agent swarms" [5]. The OpenSSH 10.6p2 release notes state that the project has seen a number of cases where a security bug identified by AI tools was subsequently discovered independently by a different researcher, and that the team will therefore make more frequent releases instead of batching fixes [2]. The discoverer's timeline also notes that a security review changed

`pick_salt()` on October 30, 2026, four days before the discovery, while the bug sat four lines below the modified code [5]. That detail shows the code path was under active scrutiny without the flaw being caught, which underscores how cross-component logic bugs can escape conventional review.

Taken together, these points suggest that cross-component logic bugs, which no single maintainer owns, are becoming cheaper to find. Organizations should plan for exploitation to begin sooner after disclosure than historical norms would suggest. With a public lab repository [5] and a deterministic, single-connection trigger [4], the barrier to weaponizing this flaw is low.

Recommendations

Immediate Actions

Within the first 24 to 72 hours, security teams should identify every musl-based host, virtual machine, and container image that runs sshd. They should prioritize systems reachable from the internet or from broadly shared networks, and patch them. On Alpine hosts, the vendor-recommended path is `apk upgrade -U -a` followed by `rc-service sshd restart`. The restart matters because the running daemon keeps the vulnerable code until it is replaced [4]. Fixed package versions by branch are listed below.

Alpine branch	Fixed `openssh` version	Alpine branch	Fixed `openssh` version
edge	10.6_p2-r0	3.20	9.7_p1-r6
3.24	10.3_p1-r2	3.19	9.6_p1-r3
3.23	10.2_p1-r1	3.18	9.3_p2-r4
3.22	10.0_p1-r11	3.17	9.1_p1-r7
3.21	9.9_p2-r1	3.16	9.0_p1-r6

Where patching cannot happen immediately, disabling password authentication fully mitigates the flaw. Alpine recommends first confirming that key-based login works while keeping an existing session open. Administrators should then add a drop-in file under `/etc/ssh/sshd_config.d/` with `PasswordAuthentication no` (read by default from Alpine 3.19), set every `PasswordAuthentication` line in the main configuration to `no`, validate with `sshd -t`,

restart, and confirm with `sshd -T | grep -i passwordauthentication` [4]. Because `sshd -T` without `-C` reports only global settings, teams should also check for `Match` blocks that re-enable password authentication; a command such as `sshd -T -C user=root,host=x,addr=0.0.0.0` shows the effective setting for a given connection. Tiny-cloud deployments built from images older than 3.21.10, 3.22.8, 3.23.8, or 3.24.4 need this per-instance action or a package update, because the original image remains vulnerable [4].

Teams should also look for signs of prior compromise. Alpine advises searching `/var/log/messages` for `Accepted password for` entries naming accounts that have no password, such as `root`, `alpine`, or key-only service users. Any such entry is a strong indicator of exploitation. On exposed systems, teams should review `doas` usage, newly created users, recently modified files, and `authorized_keys` changes, and treat a host as compromised if exploitation cannot be ruled out [4]. In containers, however, `/var/log/messages` often does not exist because no syslog daemon runs, or it is lost when the container is recycled. Teams should therefore check centralized logging, container stdout and stderr captured by the runtime, and network flow records. Where no logs survive, exposed containers should be treated as potentially compromised, and any credentials or secrets they held should be rotated.

Short-Term Mitigations

Over the following weeks, remediation should move from running instances to the build pipeline. Teams should rebuild every image that includes `openssh-server` from a patched Alpine base, redeploy it, and add a CI gate that fails builds containing an `openssh` package below the fixed version for the relevant branch. Base-image tags pinned to unsupported releases (3.16–3.21) should be treated as technical debt. Fixed packages exist for those releases today, but future fixes are not guaranteed.

Organizations should also harden configuration independently of the patch. Container images that need SSH should set `PasswordAuthentication no` in the image itself rather than rely on runtime configuration, and should avoid `PermitRootLogin yes`. Teams should review whether accounts need `*` or `!` and whether any password hashes are needed at all in images. Where password authentication is a genuine business requirement on Alpine, the `openssh-server-pam` package with `UsePAM yes` is not exposed to this flaw [4]. Runtime detection is harder than it first appears. sshd does not log password contents, and SSH encrypts authentication traffic, so network sensors cannot inspect password length directly. Detection therefore relies on heuristics such as unusually large client packets during the authentication phase, and teams should validate any such rule against their own telemetry before relying on it.

Strategic Considerations

The deeper lesson is that "no password" was encoded as a magic string, `*`, and that string coincided with a library's error sentinel. Security-relevant comparisons that rely on in-band sentinels are fragile across component boundaries. The OpenSSH fix acknowledges this by refusing to match any stored hash shorter than four characters, while stating that the maintainers do not consider the root cause to lie within OpenSSH [2][3]. Organizations that maintain their own authentication code, or that vendor C libraries for minimal images, should audit for similar assumptions about `crypt()` return values.

More broadly, for ephemeral, immutable workloads the operational case for SSH inside containers is weak. Such workloads are better served by orchestration-native access (for example, `kubectl exec` with audited RBAC, or cloud session managers) than by long-lived sshd processes with local account databases. NIST SP 800-190 recommends minimizing the software and services in container images, a principle this incident illustrates [11]. Organizations should also revisit vulnerability-response service levels in light of AI-accelerated discovery. If logic bugs at component boundaries are now being found and rediscovered by automated agents, the time between public disclosure and exploitation is likely to shrink, and the "patch next maintenance window" posture for high-severity network-reachable flaws becomes harder to defend.

CSA Resource Alignment

CSA's *Best Practices for Implementing a Secure Application Container Architecture* is the most directly relevant prior guidance [12]. It stresses minimizing what runs inside container images and treating images as immutable artifacts managed through the build pipeline, not patched in place. Both principles map onto this incident. Images that did not include sshd are unaffected, and images that did must be rebuilt, not upgraded at runtime. This note's recommendation to gate builds on fixed package versions applies that paper's guidance on securing the image lifecycle.

CSA's *Building Incident Resilient Containers* presentation addresses detection and response for containerized workloads, including the shift from asking whether an incident will happen to planning for when it does [13]. The log-hunting and compromise-assessment steps above, and the observation that ephemeral containers may have discarded the logs needed to rule out exploitation, illustrate why response processes for containerized workloads must account for container churn.

The CSA AI Controls Matrix (AICM) v1.1 provides a control framework for operationalizing these responses [14]. Its Threat and Vulnerability Management (TVM) domain covers the inventory, prioritization, and remediation-timeliness concerns raised here. Organizations applying those controls

should consider whether their remediation timelines reflect the compressed disclosure-to-exploitation window this note anticipates. Its Identity and Access Management (IAM) domain covers strong authentication and the governance of dormant or credential-less accounts, which became the attack surface in this case.

References

[1] CVE Program. "CVE-2026-123158." CVE List V5 (MITRE), November 5, 2026.

[2] OpenSSH Project. "OpenSSH 10.6p2 Release Notes." OpenSSH, November 5, 2026.

[3] Damien Miller. "avoid password authn bypass (commit 4353fdc)." openssh-portable, GitHub, November 5, 2026.

[4] Alpine Linux. "Security advisory: openssh-server password authentication bypass." Alpine Linux, November 5, 2026.

[5] Özgün Kültekin. "*sshole." sshhole.com, November 2026.

[6] Orca Research Pod. "*sshole: OpenSSH Authentication Bypass Gives Attackers Root on Alpine Linux." Orca Security, November 5, 2026.

[7] Moshe Siman Tov Bustan. "OpenSSH Password Authentication Bypass on Alpine Linux (*sshole)." OX Security, November 5, 2026.

[8] Lawrence Abrams. "OpenSSH flaw lets attackers log into password-disabled Linux accounts." BleepingComputer, November 5, 2026.

[9] NIST. "CVE-2016-6210 Detail." National Vulnerability Database, 2016.

[10] NIST. "CVE-2019-5021 Detail." National Vulnerability Database, May 2019.

[11] NIST. "SP 800-190: Application Container Security Guide." NIST, September 2017.

[12] Cloud Security Alliance. "Best Practices for Implementing a Secure Application Container Architecture." CSA Application Containers and Microservices Working Group, July 2019.

[13] Cloud Security Alliance. "Building Incident Resilient Containers." Cloud Security Alliance event presentation, May 24, 2023.

[14] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, June 22, 2026.

© 2026 Cloud Security Alliance. AI-assisted Rapid Research.