



PENETRATION TEST REPORT · COMPOSITE SAMPLE

Composite Sample Penetration Test

Prepared for [Client] by PXLSecurity LTD.

This is a sample, not a client deliverable. It is a composite assembled from several anonymised PXLSec engagements of different types — web, API, mobile, internal network and external perimeter. Every finding has been rewritten with synthetic, illustrative data on reserved example names and addresses. It describes no single real client, and no real client, host, person or credential appears anywhere in it.

REPORT REFERENCE

PXL-2026-0142

VERSION

1.0 (Final)

REPORT DATE

[Engagement dates]

TESTING WINDOW

[Agreed dates]

CLASSIFICATION

Confidential — sample

PREPARED BY

PXLSecurity LTD., Sofia, Bulgaria

PXLSecurity LTD. · 12 Vasil Levski Blvd., 1142 Sofia, Bulgaria · VAT BG203239331 · hello@pxlsec.com · independent since 2014

FRONT MATTER

Document control

This page records what the document is, who holds it and how it changed. In this sample every client-identifying field is left as an obvious placeholder.

Document title	Composite Sample Penetration Test — report
Report reference	PXL-2026-0142 · version 1.0 (Final) · supersedes 0.9 (draft)
Client	[Client]
Engagement dates	[Agreed dates] · report dated [Engagement dates]
Author	Senior security consultant, PXLSecurity LTD.
Quality assurance	Second senior consultant — technical review and independent reproduction of every finding before release
Classification	Confidential. This sample contains no client data.

Revision history

Version	Date	Author	Change
0.1	[Agreed dates]	Lead consultant	Initial draft issued at the end of active testing
0.9	[Agreed dates]	QA reviewer	Technical review; every finding reproduced independently
1.0	[Engagement dates]	Lead consultant	Final issue to the distribution list below

Distribution

Name	Role	Organisation	Copy
[Client contact]	Engagement sponsor	[Client]	Full report
[Client contact]	Head of engineering	[Client]	Full report
Lead consultant	Author	PXLSecurity LTD.	Full report
[Auditor]	External auditor	[Audit firm]	On client instruction only

Confidentiality. A genuine report of this kind describes, in reproducible detail, how to compromise a live production environment. It should be handled as one of the most sensitive documents the organisation holds: encrypted at rest and in transit, shared only with the named distribution list, and never posted to a general-purpose ticket system or file share. **This particular document is a sample and contains no client data, so none of that applies to it — it is published deliberately.**

FRONT MATTER

Contents

The structure below is the structure of every PXLSec penetration test report. Leadership can read sections 1 to 4 and act; engineering can work from section 5 onwards without reading anything else.

	SECTION	PAGE
1	Executive summary What was tested, the headline risk picture and the three things to fix first — written for a non-technical reader.	4
2	Engagement scope Targets, assessment type, testing window, standards, exclusions and the assumptions the result depends on.	6
3	Methodology The five stages, what each one produces, how tooling is used and the safety rules we test under.	7
4	Severity ratings The scale, what each level means in practice, and how CVSS informs the rating without replacing it.	9
5	Summary of findings Every finding with its identifier, severity and current status, in remediation order.	10
6	Detailed findings One section per finding: description, evidence, reproduction, impact and a specific recommendation.	11
7	Retest and closure What the complimentary retest covers and what the closure letter states.	22
A	Appendix A — Tooling used The tools in the engagement, and the division of labour between automation and manual work.	23
B	Appendix B — Out of scope What was explicitly excluded and what that means for the assurance this report provides.	24
C	Appendix C — How to read this report Who should read which section, and how to turn it into a remediation plan.	25

A note on what this sample is. Findings F-01 to F-07 are drawn from several different anonymised engagements and are presented here as if they came from one. No real assessment of a single application would normally surface a mobile bundle, a database cluster, an Active Directory service account and an internet-facing gateway in the same scope. They are collected in one document so that a prospective client can see the full range of work in a single read. Every transcript, hostname, address, account and score is synthetic and illustrative.

SECTION 1

Executive summary

PXLSecurity LTD. was engaged by [Client] to carry out a manual, objective-driven penetration test. This section is written for a non-technical reader. It says what was tested, how serious the result is and what to do first. Nothing in it requires a security background to act on.

What was tested

Senior engineers tested the in-scope systems by hand over the agreed window, working the way a real attacker would: find a way in, see how far it goes, and prove it. The composite scope behind this sample covered a customer-facing web application and its REST API, a published mobile application, the database tier behind the application, an internal network with its Active Directory domain, and the external perimeter. Testing was authenticated where the application offered accounts, so that the result reflects what a logged-in customer or a compromised employee could reach, not only what an anonymous visitor sees.

The headline

The assessment identified **seven findings: two Critical, three High and two Medium**. The overall risk rating for the engagement is **High**.

7 findings raised	2 Critical	3 High	2 Medium	High overall risk rating
-----------------------------	----------------------	------------------	--------------------	------------------------------------

Findings by severity



■ Critical 2 ■ High 3 ■ Medium 2

Critical		2
High		3
Medium		2
Low		0
Informational		0

Counts are of findings raised in this composite sample. A real engagement of comparable size would normally also carry a tail of Low and Informational hygiene observations; they are omitted here so the sample stays readable. Severity is a business-risk rating, explained in section 4.

EXECUTIVE SUMMARY · CONTINUED

What it means, and what to do first

The serious findings share one theme: **credentials and privilege are not being kept where they belong**. A password was shipped inside a public mobile application. The application's database login has complete control of the database server. The key that proves who a logged-in user is can be read by anything running on the application host. An administrator-level account in the internal directory has a password that has not changed in years and can be requested by any employee.

None of these required a novel attack, specialist equipment or an undisclosed vulnerability. Each is the result of a configuration or design decision that was reasonable at the time and was never revisited. That is the good news: the same qualities that make them easy to exploit make them straightforward to correct, and most of the work is configuration rather than development.

The risk is also compounding rather than additive. The serious findings chain together: the credential from the mobile bundle reaches the API; the database account turns any application-tier foothold into control of every tenant's data; the signing key turns host access into the ability to impersonate an administrator. This is why the overall rating is High even though no single finding is a complete compromise on its own.

The three things to fix first

- 1 Rotate every exposed credential, today.** The service account and API key found in the mobile application, the session-signing key on the application host and the password on the privileged directory account should all be treated as already known to an attacker. Rotation needs no code change and no release, and it closes the window while the longer fixes are planned. (F-01, F-03, F-04)
- 2 Cut the application's database account down to least privilege.** It currently has full control of the database server, including the ability to run operating-system commands. It needs permission to read and write its own data and nothing else — a single configuration change that removes the largest amplifier in the environment. (F-02)
- 3 Get the internet-facing gateway onto a supported platform.** It no longer receives security updates, so the next vulnerability in it will have no fix available. Because it is the device that checks staff credentials, a compromise there is an immediate internal problem. Set a date and an owner. (F-05)

What happens next. Your engineers work from section 6, which gives each finding a reproduction path, the evidence behind it and a specific fix. We walk the report through with your team in a live debrief. When you are ready, the complimentary retest verifies each fix and we reissue the report with updated statuses and a closure letter you can share with customers and auditors.

Nothing here required a clever attack. Everything here required a decision that was never revisited — which is why most of the fix list is configuration, not development.

EXECUTIVE SUMMARY · THE SHORT VERSION

SECTION 2

Engagement scope

A finding can only exist for something that was in scope, reachable and tested. Anything outside these boundaries is untested rather than secure.

Client	[Client]
Targets	Composite scope for this sample: a customer-facing web application and its REST API; the published mobile application bundle; the application-to-database tier; an internal network segment with its Active Directory domain; and the external perimeter. In a real engagement each target is listed individually by hostname, URL, package identifier or address range.
Assessment type	Grey box, authenticated. Test accounts at multiple privilege levels allowed authorisation boundaries between roles to be exercised, alongside unauthenticated testing of everything reachable without an account.
Environment	Production and staging as agreed in writing. Anything carrying a risk of disruption was run against staging first and against production only with explicit sign-off.
Testing window	[Agreed dates] · business hours, with emergency contacts on call on both sides for the duration.
Standards	OWASP Web Security Testing Guide, OWASP API Security Top 10, OWASP MASVS for the mobile component, PTES for the network and Active Directory work, and CVSS v3.1 for scoring.
Exclusions	Denial-of-service and stress testing; physical intrusion; social engineering of client staff; and any system not named in the signed scope. See Appendix B.
Team	Two senior consultants, with a third performing independent quality assurance and reproducing every finding before issue.

Assumptions and constraints

Every penetration test is a snapshot taken under constraints, and those constraints shape what the result can support. They are stated plainly so that nobody reads more assurance into this document than it carries.

- 1 The result is time-bounded.** It reflects the systems as they were during the testing window. A deployment, a configuration change or a newly disclosed vulnerability after that window is outside what was tested.
- 2 Scope is a boundary, not a verdict.** Systems excluded from scope were not tested, and nothing here should be read as assurance about them.
- 3 Absence of a finding is not proof of absence.** No assessment of finite length can prove that no vulnerability remains.
- 4 Supplied information was taken at face value.** Documentation and credentials provided by [Client] were used as given, not independently audited.
- 5 Availability was protected.** Tests that risked disruption were omitted or run against non-production, so some attack paths were confirmed by observation rather than by execution. Where that applies, the finding says so.

SECTION 3

Methodology

Testing followed our standard five-stage approach. Each stage produces a defined output, and the engagement does not move on until that output exists and has been agreed.



1 · Scope and threat model

Before any traffic is sent we agree in writing what is in scope, what is explicitly excluded, when testing happens and who to call if something breaks. We then build a threat model: who would realistically attack this system, what they would want, and which paths they would take to get it. That model drives where the hours go. Without it a test becomes a checklist, and a checklist finds checklist bugs.

Produces: a signed scope and rules-of-engagement document, a named contact on both sides, and a prioritised list of attack scenarios.

2 · Reconnaissance and mapping

We enumerate the real attack surface rather than the documented one: hosts, services, routes, parameters, roles, authentication flows, client-side bundles, third-party dependencies and anything the organisation has published without meaning to. Automated tooling is used here for coverage and speed, because breadth is a machine problem.

Produces: a complete map of the tested surface, including anything in scope that the client did not know was exposed.

3 · Manual exploitation

This is where the engagement is won or lost, and it is entirely human work. Each candidate issue is tested by hand, chained with others where chaining is possible, and — where it is safe — exploited far enough to prove genuine impact rather than theoretical risk. Authorisation boundaries are exercised with multiple accounts. Business logic is attacked on its own terms. Nothing reaches the report on the strength of a tool's say-so.

Produces: verified findings with evidence, a reproduction path and a demonstrated impact.

4 · Reporting

Every finding is written up with its evidence, the steps to reproduce it, what it means for the business and a specific fix that suits the stack in front of us. A second senior consultant reviews the report and independently reproduces each finding before it is issued. We then walk the report through with the client's team in a live debrief.

Produces: this document, a machine-readable findings export for the client's tracker, and a debrief session.

5 · Retest

Once remediation is complete we re-test every finding, confirm what has actually been fixed and reissue the report with updated statuses and a signed closure letter. This is included as standard, not sold separately, because an unverified fix is a plan rather than a result.

Produces: an updated report with per-finding status and a closure letter suitable for customers and auditors.

METHODOLOGY · CONTINUED

Tooling, verification and safety

How tooling is used

We use automated tooling heavily and trust it for exactly one thing: coverage. A scanner can enumerate a large estate faster and more consistently than a person, and it will not get bored on the four hundredth host. What it cannot do is understand what the application is for. It does not know which records an account is entitled to see, which workflow steps may be skipped, or which of two correct-looking responses represents a boundary being crossed.

So the division of labour is fixed. Automation maps the surface and raises candidates. A human decides what is real. **Every finding in this report was manually verified and, where it was safe to do so, exploited to demonstrate genuine impact.** Nothing is reported on the strength of a version banner or a tool's confidence score, and candidates that proved not to be exploitable in context are discarded rather than padded into the findings list.

What automation contributes

- Host, port and service discovery across the full in-scope range.
- Crawling and parameter enumeration on the application surface.
- Version and configuration fingerprinting at scale.
- Known-vulnerability candidates for human triage.
- Regression coverage at retest.

What only a person contributes

- Authorisation testing across multiple accounts and roles.
- Business-logic abuse that produces a valid-looking response.
- Chaining low-severity issues into a serious outcome.
- Judging real impact in this environment, not in the abstract.
- Deciding what is safe to exploit on a production system.

Safety and rules of engagement

Testing a production system carries real risk, and the controls on that risk are agreed in writing before the first request is sent. The rules below applied throughout.

- 1 Authorisation first.** No testing begins until a signed authorisation naming the in-scope systems is in place, and the team carries it for the duration.
- 2 Availability is protected.** Denial-of-service and stress testing are excluded by default. Anything with a plausible route to an outage runs against staging, or against production only with named sign-off at the time.
- 3 Minimum necessary proof.** Exploitation stops at the point impact is demonstrated. We do not exfiltrate data in bulk, destroy or modify records, or persist beyond what the proof requires.
- 4 Live contact throughout.** Named contacts on both sides are reachable for the whole window. A suspected genuine incident, a pre-existing compromise or an unexpected outage is reported immediately, not at the end.
- 5 Critical findings are escalated on discovery,** the same day they are confirmed and with interim mitigation advice, rather than held back for the report.
- 6 Clean exit.** Test accounts, uploaded files and any other artefact created during testing are removed and listed, so the client can verify the environment is as it was.

SECTION 4

Severity ratings

Findings are rated on business risk — what the issue costs this organisation if it is used — informed by CVSS v3.1, not dictated by it.

Rating	What it means	Expected response
CRITICAL	Immediate, severe business impact and straightforward to exploit. Typically unauthenticated and reliable, leading directly to data loss, financial loss or control of the environment.	Fix now. Escalate the same day; apply interim mitigation if the full fix takes time.
HIGH	Significant impact and realistic to exploit, though it may require a position, an account or a precondition. Often the decisive link in a chain.	Fix in the current cycle. Do not let it enter a general backlog.
MEDIUM	Moderate impact, or serious impact that depends on conditions outside the attacker's full control.	Plan remediation with a committed date in the next cycle or two.
LOW	Limited direct impact. Hardening and defence in depth; useful to an attacker mainly in combination with something else.	Schedule alongside related work.
INFORMATIONAL	No direct security impact. An observation or a deviation from good practice.	Note it. Act at your discretion.

How CVSS is used — and where it stops

Every finding here carries a CVSS v3.1 base score and vector. CVSS is a common language: it lets an auditor, a vendor and an engineer compare two findings without arguing about adjectives.

What CVSS deliberately does not know is your business. The base score describes a vulnerability in the abstract: it has no view on whether the affected system holds regulated personal data or a staging copy of last year's catalogue, whether a compensating control makes the attack impractical, or whether an outage there stops trading.

So we use the base score as an input and rate on business risk. Where the two disagree, the finding states the rating, the score and the reason. A headline vulnerability that is not exploitable in your context will be rated Low here; a modest-looking issue on the path to your most sensitive data will be rated High.

Scores in this sample are illustrative. This document is a composite built from anonymised engagements with synthetic data, so the CVSS scores and vectors shown are representative examples chosen to match each finding type. They are not scores assigned to any real system and should not be cited as such.

CVSS tells you how bad the vulnerability is. Only your context tells you how bad the Tuesday is.

SECTION 4 · THE SHORT VERSION

SECTION 5

Summary of findings

All seven findings, in remediation order. Section 6 sets out the evidence, reproduction path, impact and recommendation for each one.

ID	Finding	Component	Severity	CVSS	Status
F-01	Hardcoded backend credentials in a shipped mobile app	Mobile app bundle	CRITICAL	9.1	Open
F-02	Application database account is a cluster superuser	App-to-database connection	CRITICAL	9.1	Open
F-03	Production session-signing key recoverable from the app host	Web session / token signing	HIGH	8.1	Open
F-04	Kerberoastable Domain Admin service account, password unchanged for years	Active Directory service account	HIGH	8.1	Open
F-05	Internet-facing remote-access gateway on an end-of-life OS	External authentication gateway	HIGH	8.1	Open
F-06	Long-lived API token leaked via URL fragment and JavaScript global	SSO handoff / single-page app	MEDIUM	6.4	Open
F-07	Stored spreadsheet formula injection in data export	Table / CSV export	MEDIUM	6.4	Open

Status is as at the date of this report. After the retest, each row is reissued as Closed, Partially remediated or Open, with a note explaining any status that is not Closed. CVSS v3.1 base scores are illustrative in this sample.

Reading the table

The findings are ordered by severity, which here is also the order we would remediate in. That is not always the case: where a Medium finding is a prerequisite for a Critical one, or where one configuration change closes several rows at once, the debrief will say so.

- Five of seven findings are about credentials or privilege.** F-01 to F-04 and F-06 are all, at root, a secret kept in the wrong place or an account holding more rights than its job requires — one organisational problem in five systems, best treated as a single programme of work rather than five tickets.
- Not one finding required a novel attack.** No zero-day, no exotic technique. Every finding turns on a configuration or design decision, so the fixes are mostly operational and can begin before the next release.
- The two Medium findings are not filler.** F-06 produces a multi-year credential that survives logout; F-07 puts executable content on an internal reviewer's workstation. Both are Medium because each needs a precondition, not because the outcome is small.

Also delivered. Alongside this PDF you receive the same findings as a machine-readable CSV and JSON export, so identifiers, severities and statuses load straight into your issue tracker.

SECTION 6

Detailed findings

One section per finding: severity and illustrative CVSS, affected component, what was found, the evidence, a reproduction path, the impact and a fix.

About the evidence below. In a real report each capture is a redacted screenshot of the engineer's own session, with the full transcript in the appendix. Every capture here is a synthetic reconstruction — the commands are representative; the hosts, accounts, keys and output are invented.

CRITICAL**F-01 Hardcoded backend credentials in a shipped mobile app**

SEVERITY	CVSS v3.1 (ILLUSTRATIVE)	AFFECTED COMPONENT	STATUS
Critical	9.1 AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	Mobile app bundle	Open

Class: Use of hard-coded credentials (CWE-798) · **Reference:** OWASP MASVS-STORAGE-1 · OWASP API Top 10 API2:2023 · *Composite element: mobile application assessment*

Description

The published application bundle was downloaded from its distribution channel, unpacked and searched for embedded secrets. A configuration asset inside it carried a service account name, its password and a long-lived API key in plaintext.

The credentials were not inert. Replayed from a workstation with no connection to the mobile estate, they returned an authenticated response from an internal customer-data API that is not otherwise reachable without a valid session. The application was not holding a credential for its own limited use — it was publishing a working service identity to everyone who installs it, in every build that carried the same asset.

Evidence

Backend credentials hardcoded in the shipped mobile appSYNTHETIC EXAMPLE

Anyone who installs the published bundle obtains a working service credential.

```
analyst@wkstn ~$ unzip -o -q app-release.apk -d app/ && ls app/res/raw
Synthetic transcript - fabricated app bundle and invented key material
config.json  seed.bin    ca-bundle.pem

analyst@wkstn ~$ grep -nEi 'key|user|pass' app/res/raw/config.json
7: "api_base": "https://api.example.test/v3",
8: "api_key": "ak_live_7c41e0...<truncated - fabricated value>",
9: "svc_user": "svc_web",
10: "svc_pass": "<example-password>"

analyst@wkstn ~$ curl -s -o /dev/null -w '%{http_code}\n' \
-u svc_web:<example-password> \
-H 'X-API-Key: ak_live_7c41e0...<truncated>' \
https://api.example.test/v3/internal/customers
200

▶ The credential shipped in the public app returns an authenticated 200 from the internal API.
```

i Illustrative example — synthetic data, not client evidence.

Evidence for F-01 — the bundle is unpacked, the embedded service credential and API key are read from a configuration asset, and the pair returns an authenticated 200 from the internal API. Illustrative reconstruction; synthetic data, not client evidence.

Reproduction steps

- 1 Obtain the published application bundle from its distribution channel and unpack it to a working directory.
- 2 Search the bundled resources for credential-shaped strings; in the sample transcript `grep -nEi 'key|user|pass'` against the configuration asset returns the API base URL, the API key and the service account name and password.
- 3 Replay the recovered pair against the API base URL from an unrelated host, with no device, no app and no interactive login.
- 4 Observe an authenticated 200 from the internal customer endpoint, confirming the credential is live rather than a build-time placeholder.

Impact

Anyone who installs the published application obtains a working service identity against an internal customer-data API: unauthenticated access to regulated personal data at internet scale, with no account to disable and nothing anomalous for a login-monitoring control to notice, because the requests are correctly authenticated — just not by the intended party. The same key also widens the blast radius of any future application-tier bug, since an attacker starts with a valid identity rather than having to earn one.

Recommendation

- 1 Treat the exposed service account and API key as compromised and rotate both immediately, before any code change ships.
- 2 Remove every embedded credential from the bundle and from version control history; a secret that was once committed is not removed by deleting the line.
- 3 Broker authentication server-side: the application should obtain a short-lived, per-user token through SSO or an OAuth authorization-code flow with PKCE, and should never hold a shared service identity.
- 4 Add secret scanning to the build pipeline and fail the build on a positive, so the next instance is caught before release rather than at the next test.

CRITICAL

F-02 Application database account is a cluster superuser

SEVERITY	CVSS v3.1 (ILLUSTRATIVE)	AFFECTED COMPONENT	STATUS
Critical	9.1 AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H	App-to-database connection	Open

Class: Execution with unnecessary privileges (CWE-250) · **Reference:** OWASP ASVS V1.4 · CIS Database Benchmark ·
Composite element: web application and infrastructure assessment

Description

The account the application uses to reach its database holds the cluster superuser attribute. Connecting with the application's own credentials and querying the role catalogue confirmed it directly; listing databases showed the same role owning every tenant schema on the instance, not only the one the application serves. Superuser is not simply a wide grant — on this engine it also unlocks server-side command execution, so the database account is an operating-system foothold wearing a SQL interface.

Nothing about this is exploitable on its own. It needs an application-tier flaw, a leaked connection string or access to an application host first, and that is precisely why it matters: it is the amplifier that converts any one of those ordinary events into a full-cluster and full-host compromise, and it was present on every path we examined.

Evidence

The application database account is a cluster superuser

SYNTHETIC EXAMPLE

One application-tier flaw becomes full control of every tenant database on the cluster.

```
analyst@wkstn ~$ psql "host=db-01.example.test user=app_rw dbname=appdb"
Synthetic transcript — invented cluster, roles and output
appdb=# SELECT current_user, usesuper FROM pg_user WHERE username = current_user;
current_user | usesuper
-----
app_rw       | t

appdb=# \l      -- databases visible to this role
  Name      | Owner   | Access privileges
-----
appdb       | app_rw  |
tenant_a    | app_rw  |
tenant_b    | app_rw  |

appdb=# COPY (SELECT 1) TO PROGRAM 'id -un';
COPY 1      -- superuser-only: OS command runs as the DB service user
> usesuper = t: the app role reads every tenant database and runs OS commands on the host.
```

i Illustrative example — synthetic data, not client evidence.

Evidence for F-02 — the application role reports `usesuper = t`, every tenant database is visible to it, and a superuser-only statement executes an operating-system command on the database host. Illustrative reconstruction using synthetic data on reserved example.test names; not client evidence.

Reproduction steps

- 1 Connect to the database using the application's own connection parameters (host, role and database taken from the application configuration).
- 2 Query the role catalogue for the current user's superuser attribute; the sample transcript shows `usesuper` returning `t`.
- 3 List the databases visible to the role and confirm that tenant schemas belonging to other customers are owned by, and readable to, the same account.
- 4 Issue a superuser-only statement that reaches the operating system — in the sample, `COPY (SELECT 1) TO PROGRAM 'id -un'` — and observe it succeed.

Impact

Any compromise of the application process — an injection flaw, a leaked environment file, a stolen deployment artefact, a mis-scoped backup — becomes read and write access to every tenant's data and command execution on the database host. For a multi-tenant platform that is the difference between one customer's incident and every customer's incident, and it converts a contained application bug into a reportable breach. It also removes the audit value of database logging, since the account can alter the records that would describe the event.

Recommendation

- 1 Demote the application role now. It needs `SELECT / INSERT / UPDATE / DELETE` on its own schema and nothing more — no superuser, no ownership of other tenants' objects, no object creation at runtime.
- 2 Separate duties: a migration role with schema rights used only by the deployment pipeline, and a runtime role with data rights used only by the application.
- 3 Restrict the database at the network layer so only application hosts can reach the listener, and require TLS with certificate verification.

- 4 Add a recurring check that fails if any application role gains superuser, CREATEROLE or ownership outside its own schema.

HIGH

F-03 Production session-signing key recoverable from the app host

SEVERITY	CVSS v3.1 (ILLUSTRATIVE)	AFFECTED COMPONENT	STATUS
High	8.1 AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N	Web session / token signing	Open

Class: Insufficiently protected credentials (CWE-522, CWE-321) · **Reference:** OWASP ASVS V3.5 · OWASP WSTG-SESS ·
Composite element: web application and internal infrastructure assessment

Description

The secret used to sign production session tokens was held in a world-readable configuration file on the application host. Any local account — an unprivileged service account, a build agent, a monitoring collector, or anything that lands through an unrelated local flaw — can read it.

Possession of a signing key is possession of the authentication system. With the key in hand a valid session can be minted offline for any subject, any role and any tenant, with an arbitrary lifetime. The forged token never touches the login flow, so multi-factor authentication, lockout thresholds, conditional access and sign-in logging are bypassed rather than defeated: there is no authentication event to evaluate or record. We verified the chain end to end against an administrative endpoint, which returned the expected administrative response.

Evidence**Production session-signing key readable from the app host**

SYNTHETIC EXAMPLE

A world-readable key file lets any local account mint a valid session for any user.

```
analyst@app-01 ~$ ls -l /etc/appsvc/secrets.yml
-rw-r--r-- 1 appsvc appsvc 412 Mar 11 09:22 /etc/appsvc/secrets.yml

analyst@app-01 ~$ grep session_secret /etc/appsvc/secrets.yml
session_secret: "s3cr3t_9f14ab07...<truncated - fabricated value>"

analyst@app-01 ~$ python3 forge_session.py --sub bob --role admin --ttl 30d
signed offline with the recovered key - no interaction with the login flow
eyJhbGciOiI...<truncated>.eyJzdWIiOiJib2IiLCJyb2x1IjoieWVhbnRtaW4i...<truncated>.Zm9yZ2Vh...<truncated>

analyst@app-01 ~$ curl -s -H "Cookie: session=$FORGED" \
https://app.example.test/admin/users | head -1
HTTP/1.1 200 OK
{"user": "bob", "role": "admin"}
```

► The server accepts an offline-forged token and grants bob the admin role.

Illustrative example — synthetic data, not client evidence.

Evidence for F-03 — a world-readable key file yields the production session secret, a token is signed offline for an arbitrary subject and role, and the server accepts it. Illustrative reconstruction using synthetic data on reserved example.test names and the placeholder users alice / bob; not client evidence.

Reproduction steps

- 1 From any local account on the application host, list the permissions on the application secrets file; the sample transcript shows mode `-rw-r--r--`, readable by every local user.
- 2 Read the session-signing secret out of that file.

- 3 Sign a token offline for an arbitrary subject and role — in the sample, subject bob with the administrative role and a 30-day lifetime — with no interaction with the login flow.
- 4 Present the forged token as the session cookie against a privileged endpoint and observe 200 OK with the elevated role reflected in the response body.

Impact

Authentication can be bypassed entirely for any user, including administrators, without a password, without a second factor and without generating a sign-in record. Account takeover becomes silent and unbounded, and any action attributed to a user in the application logs stops being evidence of what that user did — which undermines both incident response and any audit or non-repudiation claim built on those logs. Rotating the key invalidates every live session, so the cost of the fix grows the longer it is deferred.

Recommendation

- 1 Rotate the signing key immediately and invalidate all existing sessions. Plan the forced re-authentication; do not let the disruption delay the rotation.
- 2 Move the key out of the filesystem into a managed secrets store or an HSM, fetched at process start and never written to disk or to a log. If a file must remain, restrict it to the service account with mode 0600, enforced by configuration management.
- 3 Adopt asymmetric signing so verifying services hold only the public key and a compromised verifier cannot mint tokens.
- 4 Introduce key identifiers, a scheduled rotation and server-side revocation of individual sessions, so rotating becomes routine rather than an incident.

HIGH

F-04 Kerberoastable Domain Admin service account, password unchanged for years

SEVERITY	CVSS v3.1 (ILLUSTRATIVE)	AFFECTED COMPONENT	STATUS
High	8.1 AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N	Active Directory service account	Open

Class: Insufficiently protected credentials (CWE-522) · ATT&CK: T1558.003 · **Reference:** CIS Microsoft Windows Benchmarks · Microsoft AD tiering guidance · *Composite element: internal network and Active Directory assessment*

Description

A service account with a registered service principal name is a direct member of the Domain Admins group, and its password had not been changed in several years. Any account that can authenticate to the domain — the lowest-privileged ordinary user, a shared kiosk login, a contractor, a compromised workstation — can request a service ticket for that principal. The ticket is encrypted with a key derived from the account's password and cracks offline, at any speed, with no further contact with the environment.

This is not a flaw in Kerberos; the protocol behaves exactly as designed. What makes it decisive is the combination of three choices: a human-chosen password, no rotation, and membership of the most privileged group in the directory. In the sample transcript the ticket recovers against an ordinary wordlist and the credential then enumerates Domain Admins — which is the whole directory.

Evidence

Kerberoastable service account is a Domain Admin

SYNTHETIC EXAMPLE

A requestable service ticket cracks offline to an account with full domain privileges.

```
operator@jump-01 ~$ bash
operator@jump-01:~/ops$ GetUserSPNs.py EXAMPLE.TEST/alice -dc-ip 192.0.2.10 -request
Synthetic transcript - invented SPN, fabricated ticket and crack result
ServicePrincipalName      Name      MemberOf
-----
MSSQLSvc/db-01.example.test  svc_sql  CN=Domain Admins,CN=Users

$krb5tgs$23$svc_sql$EXAMPLE.TEST$MSSQLSvc/db-01*$
a17f3c0b9e2d4a8f...<truncated - fabricated value>

operator@jump-01:~/ops$ hashcat -m 13100 tgs.txt wordlist.txt
Recovered.....: 1/1   svc_sql:<example-password>

operator@jump-01:~/ops$ net rpc group members "Domain Admins" -U svc_sql
EXAMPLE\administrator
EXAMPLE\svc_sql

> svc_sql cracks offline and is a member of Domain Admins - full domain compromise.
```

i Illustrative example — synthetic data, not client evidence.

Evidence for F-04 — an ordinary domain account requests a service ticket for a principal that is a member of Domain Admins, the ticket cracks offline, and the recovered credential authenticates with full domain privilege. Illustrative reconstruction using synthetic data, reserved EXAMPLE.TEST names, the TEST-NET address 192.0.2.10 and the placeholder accounts alice / svc_sql; not client evidence.

Reproduction steps

- 1 From any authenticated domain account, enumerate accounts that have a service principal name registered and note their group memberships.
- 2 Identify a principal that is a member of a privileged group — in the sample, svc_sql in Domain Admins.
- 3 Request a service ticket for that principal and export it in a crackable format. This is an ordinary Kerberos request and is logged as one.
- 4 Crack the ticket offline against a wordlist, then authenticate with the recovered credential and enumerate a privileged group to confirm the access obtained.

Impact

Any authenticated user in the domain can obtain Domain Admin. From there every domain-joined system, every file share, every backup target and every credential held in the directory is reachable, and the attacker can create persistence that survives password resets and rebuilds. This is the point at which an incident stops being about one host and starts being about the entire estate, and the offline crack leaves nothing anomalous in the authentication logs.

Recommendation

- 1 Remove the account from Domain Admins. A service that genuinely needs that level of privilege is almost always a service that needs redesigning.
- 2 Convert it to a group-managed service account so the password is machine-generated and rotated automatically; where that is not possible, set a random passphrase of at least 25 characters and rotate it on a schedule.

- 3 Audit every account in the directory carrying a service principal name and treat privileged ones as a standing exception requiring sign-off. Apply tiering so privileged credentials never authenticate to workstations or ordinary members.
- 4 Alert on service-ticket requests for privileged principals using weak encryption types, and on any bulk enumeration of service principal names.

HIGH

F-05 Internet-facing remote-access gateway on an end-of-life OS

SEVERITY	CVSS v3.1 (ILLUSTRATIVE)	AFFECTED COMPONENT	STATUS
High	8.1 AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H	External authentication gateway	Open

Class: Use of unmaintained components (CWE-1104) · information exposure (CWE-200) · **Reference:** OWASP WSTG-INFO · PCI DSS 6.3.3 · CIS Controls 7 · *Composite element: external perimeter assessment*

Description

The remote-access gateway that fronts staff authentication is published to the internet and runs an operating system build that left vendor support some time ago. Unsupported means no security patches are issued at all — the exposure is not a list of known missing fixes but an open-ended commitment to accept every future vulnerability in the platform permanently. The same host also discloses internal naming: the response headers carry the backend host name and an unauthenticated diagnostic endpoint returns the internal domain and node identifier.

Individually those are low-severity information leaks. Here they sit on the device that terminates authentication for the internal directory, so they hand an attacker the naming convention, the domain and a confirmed target before any credential is attempted. We did not attempt to exploit a platform vulnerability against this host: exploitation of an unsupported production gateway carries a real risk of outage, and the rules of engagement excluded denial of service. The finding is evidenced by version and build confirmation rather than by a shell.

Evidence**Internet-facing remote-access gateway runs an end-of-life OS**

SYNTHETIC EXAMPLE

The gateway exposes an out-of-support build and leaks internal naming in its response.

```
analyst@wkstn ~$ nmap -sV -p 443 --script http-title vpn.example.test
Synthetic transcript — invented gateway, banner and build string
Nmap scan report for vpn.example.test (192.0.2.80)
PORT      STATE  SERVICE  VERSION
443/tcp   open   https    gateway httpd (build 8.1-eol)

analyst@wkstn ~$ curl -sI https://vpn.example.test/ | grep -iE 'server|x-'
Server: Gateway-OS 8.1 (unsupported since 2023)
X-Backend-Host: gw-01.corp.example.test

analyst@wkstn ~$ curl -sk https://vpn.example.test/diag | head -2
{"domain":"corp.example.test", "node":"gw-01"}

► An internet-facing gateway on an end-of-life OS, also leaking the internal domain and host name.
```

Illustrative example — synthetic data, not client evidence.

Evidence for F-05 — service and build identification against the internet-facing gateway, the unsupported platform string in the response headers, and an unauthenticated diagnostic endpoint returning the internal domain and node name. Illustrative reconstruction using synthetic data on reserved example.test names and the TEST-NET address 192.0.2.80; not client evidence.

Reproduction steps

- 1 Identify the service and build on the published gateway with a version scan against the TLS service.
- 2 Request the root resource and inspect the response headers; the sample shows the platform string with its unsupported build and an X-Backend-Host header carrying the internal host name.
- 3 Request the unauthenticated diagnostic endpoint and observe the internal domain and node identifier in the response body.
- 4 Cross-reference the identified build against the vendor's support lifecycle to confirm it no longer receives security updates. No exploitation was attempted against the live gateway.

Impact

An authentication boundary that cannot be patched will eventually be breached by a vulnerability nobody has published yet, and the organisation will have no remedy available on the day it is. Because this device terminates authentication for the internal directory, a compromise there is not a perimeter incident — it is a credential incident with an internal foothold attached. The disclosed internal names shorten the reconnaissance phase of exactly that attack.

Recommendation

- 1 Migrate the gateway role to a supported, currently patched platform. Treat this as a dated project with an owner, not a backlog item.
- 2 Until migration completes, restrict administrative interfaces to known source ranges, enforce multi-factor authentication for every account that can traverse the gateway, and raise monitoring on it.
- 3 Suppress internal naming in responses: remove the backend host header, and require authentication on — or remove — the diagnostic endpoint.
- 4 Add the asset to a lifecycle inventory that records vendor end-of-support dates and raises them before they pass, then re-scan the external estate for other unsupported builds.

MEDIUM

F-06 Long-lived API token leaked via URL fragment and JavaScript global

SEVERITY	CVSS v3.1 (ILLUSTRATIVE)	AFFECTED COMPONENT	STATUS
Medium	6.4 AV:N/AC:H/PR:L/UI:R/S:U/C:H/I:H/A:N	SSO handoff / single-page app	Open

Class: Sensitive information in a URL (CWE-598) · insufficient session expiration (CWE-613) · **Reference:** OWASP ASVS V3.3 · OAuth 2.0 Security BCP (RFC 9700) · *Composite element: web application and API assessment*

Description

The single-page application receives its API access token from the SSO handoff in the URL fragment and then assigns it to a global variable on the page. Both are poor places for a bearer credential: the fragment is written into browser history and session-restore data and is visible to anything with access to the address bar or the profile on disk, while the global is readable by every script running in the page, including any third-party tag or analytics bundle.

The token's lifetime makes the exposure materially worse. The decoded claims show a validity measured in years rather than minutes, and the token continued to authenticate after the user had logged out: logout clears the client state but does not revoke the credential server-side. The rating is Medium rather than High only because a precondition is needed — the browser, the history, a shared workstation or a script in the page.

Evidence

Long-lived API token leaked via URL fragment and JS global

SYNTHETIC EXAMPLE

A token with a multi-year lifetime is exposed client-side and survives logout.

```
analyst@wkstn - node

analyst@wkstn:~$ cat captured-url.txt
Synthetic transcript — invented token and decoded claims
https://app.example.test/dashboard#access_token=eyJhbGciOi...<truncated>

analyst@wkstn:~$ node -e "console.log(window.__API_TOKEN)" # page global
eyJhbGciOi...<truncated>

analyst@wkstn:~$ node decode.js --field exp
iat: 2026-01-04   exp: 2029-01-03   (lifetime ~1095d)

analyst@wkstn:~$ curl -s -o /dev/null -w '%{http_code}\n' \
-H "Authorization: Bearer eyJhbGciOi...<truncated>" \
https://api.example.test/v3/me # replayed after logout
200

▶ The fragment/JS-exposed token has a ~3-year lifetime and still authenticates after logout.
```

i Illustrative example — synthetic data, not client evidence.

Evidence for F-06 — the access token arrives in the URL fragment, is also readable from a page global, decodes to a multi-year lifetime, and still authenticates to the API after logout. Illustrative reconstruction using synthetic data on reserved example.test names; not client evidence.

Reproduction steps

- 1 Complete the SSO handoff and capture the resulting URL; the access token is present in the fragment after the #. Read the same token from the page's JavaScript global to confirm it is also exposed to any script in the document.
- 2 Decode the token claims and compare the issued-at and expiry values; the sample shows a lifetime of roughly three years.
- 3 Log out of the application through the user interface.
- 4 Replay the captured token as a bearer credential against an API endpoint and observe 200, confirming no server-side revocation took place.

Impact

A token recovered from browser history, a shared or kiosk workstation, a synchronised browser profile, a screen share or a third-party script grants full API access as that user for years, and logging out does not take it away. On a shared device the next person to sit down can retrieve it from history without any technical skill. Because there is no revocation path, the only response to a suspected leak is to rotate the signing key and disrupt every user at once.

Recommendation

- 1 Use the OAuth 2.0 authorization-code flow with PKCE so that no token is ever delivered in a URL fragment.
- 2 Keep tokens out of page globals. Hold them in memory within a closure, or in an HttpOnly, Secure, SameSite cookie issued by a server-side session broker.
- 3 Shorten access-token lifetime to minutes and use refresh tokens that are rotated on each use and bound to the client.
- 4 Revoke tokens server-side on logout, on password change and on role change, and give support staff a way to revoke an individual session.

MEDIUM

F-07 Stored spreadsheet formula injection in data export

SEVERITY	CVSS v3.1 (ILLUSTRATIVE)	AFFECTED COMPONENT	STATUS
Medium	6.4 AV:N/AC:H/PR:L/UI:R/S:U/C:H/I:H/A:N	Table / CSV export	Open

Class: Improper neutralisation of formula elements in a CSV file (CWE-1236) · **Reference:** OWASP WSTG-INPV · OWASP ASVS V5.3 · *Composite element: web application assessment*

Description

A user-controlled profile field accepts a value beginning with an equals sign and stores it verbatim. The application's export feature then writes that value into a CSV file without neutralising it, so the cell is emitted as a live formula rather than as text. When the export is opened in a spreadsheet application, the formula is evaluated on the machine of whoever opened it.

The exposure is routinely misread as cosmetic because the application itself is unaffected. The vulnerability does not execute in the web tier at all — it executes in the desktop application of an internal reviewer in operations, finance or support who opens the export as part of their job. Depending on the spreadsheet client and its configuration, formula payloads can reach external data, trigger a protocol handler or prompt the user to run a command, and the prompt arrives attached to a file the user has every reason to trust. Those conditions are common rather than guaranteed, which is why the rating is Medium.

Evidence

Stored formula injection reaches the data export

A value beginning with = is stored verbatim and exported as a live spreadsheet formula.

SYNTHETIC EXAMPLE

```
analyst@wkstn ~$ curl -s -X POST https://app.example.test/api/v3/profile \
-d 'display_name==2+5+cmd|"/c calc"!A1'
Synthetic transcript - invented record and exported file
{"status":"saved"}

analyst@wkstn ~$ curl -s https://app.example.test/api/v3/export.csv -o out.csv
analyst@wkstn ~$ cat out.csv
id,display_name,email
1,alice,alice@example.test
2,=2+5+cmd|"/c calc"!A1,bob@example.test

the stored value reaches the CSV with its leading = intact
# a spreadsheet client parses the cell as a formula, not text,
# so it executes in the reviewer's application

▶ The = value reaches the export as a live formula that runs in the reviewer's spreadsheet.
```

Illustrative example — synthetic data, not client evidence.

Evidence for F-07 — a profile value beginning with = is stored verbatim and reaches the generated CSV with its leading character intact, so a spreadsheet client parses the cell as a formula. Illustrative reconstruction using synthetic data on reserved example.test names and the placeholder users alice / bob; not client evidence.

Reproduction steps

- 1 Authenticate as an ordinary low-privilege user and set a profile field to a value whose first character is = and which contains a spreadsheet formula; the application accepts and stores it.
- 2 Trigger the data export that includes that field — typically from an administrative or reporting view used by another member of staff.

- 3 Inspect the generated file and confirm the stored value appears with its leading = intact and is neither quoted nor prefixed.
- 4 Open the file in a spreadsheet client in a controlled test environment and observe the cell being parsed as a formula rather than as text. No payload was executed against any live workstation during this assessment.

Impact

An unprivileged user can place executable content into a file that a trusted internal reviewer will open on a corporate workstation — a path from the lowest privilege level in the application to code execution inside the office network, bypassing the perimeter because the file arrives through the organisation's own reporting process. It also poisons exports shared with auditors, regulators and customers.

Recommendation

- 1 Neutralise every exported cell whose first character is =, +, -, @, a tab or a carriage return — prefix with a single quote or wrap the value so the client treats it as text.
- 2 Apply the control in the export layer, once, for every export in the product. Per-field or per-endpoint fixes reliably miss the next export someone adds.
- 3 Where the format allows it, write typed cells and declare the column as text rather than relying on the client's parsing heuristics, and add a regression test that round-trips a formula-shaped value through store and export and asserts it comes back inert.

Remediation at a glance

The same seven recommendations, reduced to the decision each one needs. Effort is our estimate of the work itself, not of the change-control around it.

ID	What to do	Who owns it	Effort	When
F-01	Rotate the shipped credential and key; broker authentication server-side	Mobile and platform	Hours, then days	Today, then this cycle
F-02	Demote the application database role to least privilege	Database and platform	Hours	Today
F-03	Rotate the signing key; move it into a secrets store	Platform	Hours, then days	Today, then this cycle
F-04	Remove the account from Domain Admins; convert to a managed service account	Active Directory	Hours	This cycle
F-05	Migrate the gateway role to a supported, patched platform	Infrastructure	Weeks (project)	Dated plan now
F-06	Move to authorization code with PKCE; shorten and revoke tokens	Web and identity	Days	Next cycle
F-07	Neutralise formula characters in the export layer	Application	Hours	Next cycle

A reminder about this section. F-01 to F-07 are drawn from several different anonymised engagements and presented here as though they came from one. Every transcript, hostname, address, account and score above is synthetic and illustrative, and no single real client is described.

SECTION 7

Retest and closure

Remediation is only finished when someone has attacked the fix. The retest is included in the engagement price as standard — it is not an add-on and it is not quoted separately.

What the retest covers

When your team confirms that remediation is complete, we return and test every finding in this report again. The retest is not a documentation review: each finding is re-attacked using the same reproduction path recorded in section 6, so that a fix is confirmed by failing to work rather than by being described.

- 1 Every finding is retested, not a sample.** All seven rows in section 5 are re-examined, including the Mediums.
- 2 The fix is tested, and so is the way around it.** We check that the reproduction path now fails, and we look for the obvious bypass — a filter applied on one endpoint but not its sibling, a rotated credential still valid in a second place, a privilege reduced in one environment and not the other.
- 3 Regression within the affected component.** Where a fix changes authentication, authorisation or output handling, we re-exercise the surrounding functionality to confirm it did not open something new.
- 4 A clear per-finding outcome.** Each finding is reissued as Closed, Partially remediated or Open, with a short note explaining any status that is not Closed. A fresh issue introduced by the remediation work is reported too, though the retest is not a second full assessment of the whole scope.

What the closure letter states

After the retest we issue a signed closure letter alongside the reissued report. It is written to be shared: with customers running vendor due diligence, with auditors evidencing a control, and with a board that wants a one-page answer. It states, without requiring the reader to open the technical report:

The letter records

- Who was tested, by whom, and over which dates.
- The scope and the standards the testing followed.
- How many findings were raised, at which severities.
- How many are confirmed closed, and the date of the retest.
- Any finding that remains open, with its severity and the client's stated plan.
- The name and signature of the consultant who verified the remediation.

The letter deliberately does not record

- Reproduction steps, payloads or evidence.
- Hostnames, addresses or architectural detail.
- Anything that would help an attacker who obtained the letter.
- Any claim that the environment is now free of vulnerabilities — it says what was tested and what was fixed, and nothing more.

Why we include the retest rather than sell it. A finding that has been fixed but never verified is a plan, not a result — and in our experience roughly the difference between a report that changes an environment and a report that is filed. Including the retest removes the commercial reason to skip the only step that proves the work was worth doing.

APPENDIX A

Tooling used

The tools below supported the engagement. They are listed for transparency and for your own records; none of them produced a finding on its own.

Stage	Typical tooling	What it was used for
Reconnaissance and mapping	Network and service scanners, DNS and certificate-transparency enumeration, web crawlers and content discovery	Establishing the real attack surface: live hosts, open services, software versions, routes, parameters and anything published unintentionally.
Web and API testing	An intercepting proxy with manual repeater and fuzzing, a browser developer console, and a request-replay client	Manipulating requests by hand, comparing responses across accounts and roles, and exercising business logic that no scanner can model.
Mobile assessment	Bundle extraction and static analysis utilities, string and secret scanners, traffic interception with certificate handling	Unpacking the published bundle, recovering embedded configuration and secrets, and observing what the application actually sends.
Infrastructure and Active Directory	Directory enumeration and Kerberos utilities, SMB and LDAP clients, offline password-recovery tooling	Mapping the domain, identifying privileged and service accounts, and demonstrating credential-based escalation paths offline.
Verification and reporting	Scripted request replay, evidence capture, and an internal reporting pipeline	Reproducing each finding independently, capturing evidence, and generating this document and the machine-readable export.

On custom tooling

Where an environment does not fit an off-the-shelf tool — an unusual protocol, a bespoke authentication handshake, a volume of data that needs sifting — we write what is needed for the engagement. Any script that touches the client's environment is described in the appendix of the real report and provided on request, so that nothing that ran against your systems is opaque to you.

Tools do not produce findings. Everything in section 6 was verified by hand and, where safe, exploited to demonstrate real impact. A tool's output is a candidate for investigation, never a line in the report. Candidates that proved not to be exploitable in context were discarded rather than reported as findings.

APPENDIX B

Explicitly out of scope

What was excluded, why, and what that means for the assurance this report provides. Anything listed here was not tested, and nothing in this report should be read as a statement about it.

Excluded	Reason	What that means for you
Denial-of-service and stress testing	Excluded by agreement. Deliberately exhausting a production service is indistinguishable from an outage.	This report says nothing about how the environment behaves under load or under a volumetric attack. Resilience testing is a separate exercise.
Physical intrusion	Not within the agreed scope for this engagement type.	Offices, data centres, badge systems and on-site network access were not assessed.
Social engineering of client staff	Excluded by agreement. Phishing and pretexting are commissioned separately so that consent and handling of staff data can be agreed on their own terms.	The human layer was not tested. Several findings here would be reached faster by an attacker who phished first.
Third-party and supplier systems	Not ours to test. Testing a system you do not own requires that owner's written authorisation.	Hosted services, payment providers and partner integrations were treated as boundaries. Where one appeared in a path, the path was followed to the boundary and stopped.
Source code review	Not commissioned for this engagement. Testing was black and grey box against running systems.	Defects visible only in source — a dormant code path, a weak algorithm never reached in the tested flow — would not surface here.
Anything not named in the signed scope	A penetration test is authorised against a specific list.	Systems discovered during testing but outside the signed scope were reported as observations and not probed.

Out of scope means untested, not secure. It is the most commonly misread line in any penetration test report. If a system does not appear in section 2, this document makes no claim about it at all — good or bad. Where an exclusion leaves a risk you care about, the debrief is the right place to scope it into the next piece of work.

APPENDIX C

How to read this report

Different readers need different parts of this document. Nobody needs all of it, and reading it front to back is usually the slowest way to act on it.

› **If you are an executive or a board member**

Read section 1 and stop. It states what was tested, how serious the result is and the three things to fix first, without jargon. Section 4 explains what the severity words mean if you want to calibrate them.

› **If you own the remediation programme**

Read sections 1, 4 and 5, then the recommendation block of each finding in section 6. Load the machine-readable export into your tracker, assign each finding an owner and a date, and look for rows that share a root cause — here, five of seven come back to credentials and privilege.

› **If you are an engineer fixing a finding**

Go straight to your finding in section 6. The reproduction steps let you see the issue before you change anything, and confirm the fix by running them again and watching them fail. The recommendation is ordered: the first item is what to do today, the rest is what makes it stay fixed.

› **If you are an auditor or a customer doing due diligence**

Read sections 2, 3 and 4 to establish what was tested and how, then section 5 for the result. After remediation, the closure letter in section 7 is the document intended for you: the outcome, without the attack detail.

› **If you are deciding whether to commission a test**

This document is the deliverable. What you would receive is this structure, filled with your systems: evidence for every finding, a reproduction path your developers can follow, a fix that fits your stack, a live debrief, and a retest.

The anatomy of a finding

Every finding in section 6 carries

- **Identifier and title** — stable, so it can be tracked to closure.
- **Severity and CVSS v3.1** — the business-risk rating, with the score and vector that informed it.
- **Affected component** — what to go and look at.
- **Finding class** — the CWE or technique reference, for your own taxonomy.
- **Description** — what was found and why it matters.

- **Evidence** — the capture that establishes the finding.
- **Reproduction** — numbered steps your engineers can follow.
- **Impact** — what it costs the business if it is used.
- **Recommendation** — a specific fix for this stack, in the order we would apply it.
- **Status** — Open at issue; updated at retest.

The measure of a report is not how much it contains. It is how much of it somebody can act on without ringing the author.

APPENDIX C · THE SHORT VERSION



PXLSec · [REQUEST A PROPOSAL](#)

This is the deliverable. Now have it written about your systems.

Everything in this sample — the evidence, the reproduction paths, the fixes, the retest — is what a PXLSec engagement produces. The only thing missing is your environment. Tell us what you want tested and a senior consultant, not a salesperson, will come back with questions, a proposed approach and a quote.

- Penetration testing — web, API, mobile, cloud, internal and external infrastructure.
- Active Directory and internal network assessment, where most internal compromises actually begin.
- Red teaming and adversary simulation, reported as a narrative of how the environment really falls.
- Compliance-driven testing for ISO 27001, SOC 2, PCI DSS, DORA and NIS2.
- Retests included as standard, because a fix that was never verified is a plan, not a result.

pxlsec.com/contact · hello@pxlsec.com

PXLSecurity LTD. · 12 Vasil Levski Blvd., 1142 Sofia, Bulgaria · VAT BG203239331 · independent since 2014

This document is a composite sample. It was assembled from several anonymised PXLSec engagements of different types and rewritten with synthetic, illustrative data. It describes no single real client. All names use the reserved example.test domain and TEST-NET addresses in 192.0.2.0/24; all accounts, keys, transcripts and CVSS scores are invented for illustration. No client name, hostname, address, credential, person or engagement date appears anywhere in it.

