IDOR via Predictable Username Generation

The target exposes an unauthenticated API endpoint, `GET /api/userInfo/showUserDetails/{username}`, that returns full PII for any user whose username you can supply — no session, cookie, or API key required. The catch: there's no listing endpoint and no sequential numeric ID to walk.

The "ID" is a username, and the username turns out to be derived from a person's own name — first initial + surname, with a numeric suffix appended only to disambiguate people who'd otherwise share the same `initial+surname` — rather than randomly assigned. That makes the whole user base enumerable via a targeted wordlist instead of a login bypass or token leak.

Flag: `flag{9c9b920xxxxxxx}` — found on user `cgarcia`, via `GET /api/userInfo/showUserDetails/cgarcia`.

---

1. Confirming the IDOR

```
GET /api/userInfo/showUserDetails/jrodriguez75 HTTP/1.1
Host: nch40d908ea8.ctfhub.io
Accept: application/json
```


```
{
  "estado": "éxito",
  "datos": {
    "nombre_usuario": "jrodriguez75",
    "nombre": "J",
    "apellido": "Rodriguez",
    "correo_electronico": "jrodriguez75@ejemplo.com",
    "edad": 50,
    "telefono": "+34 693 741 885",
    "direccion": "Plaza del Sol 45, 28801",
    "carnet_identidad": "82980065K",
    "ciudad": "Ciudad de México"
  }
}
```

Sent with **no cookie, no Authorization header, nothing** — full name, email, age, phone, home address, national ID number, and city came back regardless. That's the whole vulnerability in one request: broken access control (missing authentication) on a route that returns PII keyed by a guessable identifier. Everything after this point is about *how guessable* that identifier actually is.

2. Reverse-engineering the username

Staring at `nombre_usuario: "jrodriguez75"` next to `apellido: "Rodriguez"` and `edad: 50`, the shape is obvious once you see it:

```
{first-initial}{surname}{2-digit-suffix}
   j          + rodriguez +    75
```

The interesting question was what `75` *means*. Two hypotheses:

1. **Disambiguation counter** — a suffix appended when two people share the same `initial+surname`, to keep their usernames unique.
2. **Birth-year suffix** — the session date was 2026-07-17, and a person aged 50 was born in either 1975 or 1976 depending on whether their birthday had passed. `75` matches 1975 cleanly.

Testing the *bare* form, `jrodriguez` with no suffix at all, before declaring the birth-year theory. It turns out to be a live, distinct account:


```
{
  "nombre_usuario": "jrodriguez",
  "nombre": "J", "apellido": "Rodriguez",
  "edad": 44,
  "telefono": "+34 740 542 917",
  "direccion": "Calle Luna 7, 28071",
  "carnet_identidad": "76641619F",
  "ciudad": "Ciudad de México"
}
```


That's a second, real "J. Rodriguez" — different age, phone, address, and national ID from `jrodriguez75` — reachable with **no suffix whatsoever**. That single data point falsifies "every account carries a birth-year suffix": if the suffix were a mandatory encoding, the bare form couldn't be a complete, valid identity on its own. What it's consistent with is

hypothesis (1): the first person registered under a given `initial+surname` gets the bare form, and later people who'd otherwise collide with that name get a numeric suffix appended to disambiguate.

2. Scaling the guess: from one surname to a wordlist

Confirming the pattern on `rodriguez` proves the vulnerability class; it doesn't harvest the user base. The natural next step was widening the search: more surnames, every possible first initial, and a numeric-suffix range wide enough to cover both the "no suffix" and "disambiguating suffix" cases, a search space in the tens of thousands, which is exactly what a purpose-built fuzzer is for.

Caido's Automate feature is that tool.
AI generates the candidate wordlist, while someone fires Caido's Automate UI to actually run it.

Wordlist construction

```
{initial}{surname}{suffix}
```

- Initials: `a` – `z` (26)
- Surnames: 11 common Spanish/Latin-American surnames — `garcia, martinez, rodriguez, lopez, gonzalez, perez, sanchez, ramirez, torres, flores, gomez`
- Suffix, three variants per (initial, surname) pair, based on what we'd actually confirmed and what was still untested:
  - No suffix (`agarcia`) — untested until explicitly requested; turned out to be the winning shape
  - Single digit (`agarcia0` – `agarcia9`)
  - Two-digit, age-plausible (`agarcia51` – `agarcia99`, `agarcia00` – `agarcia08` — modeled on ages ~18–75 as of 2026, since `jrodriguez75` was the one confirmed suffixed example, even though the suffix's exact meaning was still unconfirmed at this point)

Total: 26 × 11 × (1 + 10 + 58) = 19,734 candidates

The iteration on suffix shape mattered: the initial wordlist only covered two-digit birth-year suffixes (mirroring the one confirmed example, `jrodriguez75`), which is a natural but unverified generalization — nothing had actually confirmed that *every* account follows the exact two-digit pattern. Broadening to include no-suffix and single-digit variants before

running the fuzz is what caught the flag; a narrower, "confirm my own hypothesis" wordlist would have missed it.

3. Running it: Caido Automate

1. Took a captured `GET /api/userInfo/showUserDetails/jrodriguez75` request into Replay.
2. Marked the `jrodriguez75` segment of the path as the fuzz position.
3. Loaded `username_wordlist.txt` as a Simple List payload.
4. Ran the Automate task and filtered results by status code — `404` was the baseline noise, `200` was signal.

One row stood out: `cgarcia` (the no-suffix variant).

```
GET /api/userInfo/showUserDetails/cgarcia HTTP/1.1
Host: nch40d908ea8.ctfhub.io
```


```
{
  "estado": "éxito",
  "datos": {
    "nombre_usuario": "cgarcia",
    "nombre": "C",
    "apellido": "Garcia",
    "correo_electronico": "cgarcia@ejemplo.com",
    "edad": 21,
    "telefono": "+34 610 915 022",
    "direccion": "Plaza del Sol 28, 28773",
    "carnet_identidad": "29367700N",
    "ciudad": "Lima",
    "flag": "flag{9c9b920xxxx}"
  }
}
```


`edad: 21` with no birth-year suffix in the username lines up with the theory that suffix presence/format isn't uniform across the dataset — another reason a single rigid guess (always 2-digit birth year) would have missed this record.

4. Root cause & impact

- CWE-639 (Authorization Bypass Through User-Controlled Key) / OWASP A01:2021 – Broken Access Control: `showUserDetails` performs

no authentication or authorization check at all.

- CWE-330 / weak identifier generation: the resource identifier (username) is not a random, unguessable token — it's derived from a person's own name (first initial + surname, plus an occasional disambiguating suffix), so knowing or guessing someone's name is sufficient to compute likely valid identifiers without ever seeing the user directory.
- Impact: full PII disclosure (name, email, phone, home address, national ID number, city) for any user in the system, at scale, without credentials — a mass PII harvesting primitive, not just a one-off record leak.

8. Remediation

1. Require authentication and an authorization check (the requester should only be able to fetch their own record, or need an explicit admin role) on `showUserDetails`.
2. Stop using human-derivable values as primary lookup keys. Use random, high-entropy, non-guessable identifiers (UUIDv4, etc.) for anything reachable by external input.
3. Rate-limit and monitor for enumeration patterns (high-volume 404s against a single endpoint shape from one client is a strong enumeration signal, and this endpoint's response length/shape trivially distinguishes hits from misses even under HTTP 200-everywhere schemes like the hospital-portal lab from earlier in this engagement).
4. Return uniform 404s regardless of *why* a lookup failed (missing vs. unauthorized) so response behavior doesn't leak which usernames are syntactically valid vs. simply forbidden.