1. **How Request Smuggling Works in This Context**
   - **Inconsistent Header Handling**:
     - The front-end server (Apache Traffic Server) prioritizes the `Content-Length` header.
     - The back-end server (Nginx) prioritizes the `Transfer-Encoding` header.
   - This discrepancy allows the attacker to craft a request where:
     - The front-end server sees it as one request (ending based on `Content-Length`).
     - The back-end server interprets it as two requests due to `Transfer-Encoding: chunked`.
   - **Result**: The second part of the payload (after `0`) is treated by the back-end as an independent request, smuggling a malicious or intercepting request into the server's pipeline.

---

2. **Why Are Other Users' Requests Intercepted?**
   - When an attacker smuggles a request like:

```
POST /contact.php HTTP/1.1
Host: httprequestsmuggling.thm
username=test&query=§
```

   - The smuggled request is queued in the pipeline, creating a "split" effect:
     - The front-end sends other users' requests into the pipeline unaware of the smuggled request.
     - The back-end processes the attacker's smuggled request alongside other users' requests.
   - **Pipeline Mixing**:
     - The back-end reads the attacker's smuggled request as part of other users' sessions.
     - As a result, users' legitimate requests to `/contact.php` are inadvertently appended to the smuggled request or processed together.
   - **Intercepted Requests**:
     - When the back-end processes `/contact.php`, the attacker sees data submitted by other users (e.g., form inputs or sensitive information).

---

3. **Why Use Null Payloads?**

   - **What Are Null Payloads?**
     - Null payloads are empty variations of the crafted payload sent repeatedly during an automated attack.
     - Each "null" payload doesn't change the payload content but increases the attack's frequency and range.
   - **Purpose in This Attack**:
     - To **simulate multiple requests** being sent rapidly.
     - To **maximize the chances** of:
       * Other users submitting requests during the attack.
       * Intercepting those users' requests in the `/contact.php` endpoint.
   - **Practical Use**:
     - The attacker sends 10,000 null payloads to increase overlap with user activity on the server.
     - This high volume ensures that during the smuggling attack, at least some legitimate user requests are captured.

---

4. **Why Does This Happen?**

   - **Root Cause**:
     - Poorly configured servers handling HTTP headers inconsistently.
     - HTTP/1.1's flexibility with `Content-Length` and `Transfer-Encoding` headers, which leads to ambiguities.
   - **Exploitation**:
     - The attacker exploits these ambiguities to inject unauthorized requests into the back-end pipeline.
     - Other users' requests unintentionally interact with the smuggled payload, allowing the attacker to capture their sensitive data.

Which Protocol is Mostly Used: HTTP/1.1 or HTTP/2?

1. **Current Usage Trends**:
   - **HTTP/1.1** is still widely used because it has been the standard for decades and many legacy systems rely on it. It's common in environments where simplicity and compatibility are prioritized.
   - **HTTP/2** is increasingly popular, especially for modern applications, due to its performance benefits like multiplexing, header compression, and server push.
2. **Adoption**:

  - According to web statistics, **HTTP/2** adoption is growing, but **HTTP/1.1** still dominates in certain contexts due to backward compatibility and the slower pace of infrastructure upgrades.
  - Most browsers and CDNs (like Cloudflare) support HTTP/2 by default, but fallback to HTTP/1.1 occurs when the server or network doesn't support HTTP/2.

3. **Real-World Scenarios**:

  - HTTP/1.1 is prevalent in smaller, less modernized setups.
  - HTTP/2 is standard for high-traffic, performance-critical applications, such as streaming platforms or e-commerce sites.

Is This Attack Realistic in the Real World?

1. How This Works in Theory

- **Request Smuggling in HTTP/1.1**:
  - The attack relies on ambiguous handling of `Content-Length` and `Transfer-Encoding` headers.
  - Front-end servers (proxies) and back-end servers sometimes interpret these headers differently, leading to pipeline misalignment.
  - When this happens, attackers can manipulate the pipeline to insert their payloads or capture others' requests.

2. Real-World Feasibility

- **Is It Possible?**

  - Yes, **request smuggling has been exploited in the real world**, but it requires very specific conditions:
    * Misconfigured or outdated servers (e.g., Apache, Nginx, or load balancers).
    * Applications using HTTP/1.1 with ambiguous header handling.
    * High user activity during the attack to intercept real requests.

- **Challenges in Execution**:

  - Predicting user activity at the right moment to capture sensitive data is tricky.
  - Modern web applications often use CSRF tokens or authentication mechanisms, making it harder to exploit intercepted data.
  - HTTP/2 inherently prevents such attacks since it doesn't allow `Content-Length` and `Transfer-Encoding` headers to coexist, thus eliminating ambiguity.

- **Likelihood of Exploiting in the Wild**:

  - While **not impossible**, it's rare because:

    * Organizations increasingly use HTTP/2 or secure configurations.
    * Tools like Web Application Firewalls (WAFs) detect and block malformed headers.

Does It Make Sense?

- **Intercepting Other Users' Requests**:
  - In vulnerable environments, the attacker's smuggled request can sit in the pipeline.
  - When legitimate users make requests, their data gets appended to or processed as part of the attacker's smuggled payload.
  - However, intercepting highly sensitive information (e.g., passwords) is context-dependent and often challenging without additional vulnerabilities.
- **Modern Realities**:
  - The attack's practicality decreases in modern setups using HTTP/2 or robust security practices.
  - However, in older or poorly maintained systems, it's still a legitimate threat.

Conclusion

- **HTTP/2 vs. HTTP/1.1**:
  - HTTP/2 is more secure against this type of attack, but HTTP/1.1 remains widely used in specific setups.
- **Real-World Feasibility**:
  - The attack is theoretically possible but rare in modern environments.
  - Most real-world applications employ mitigations like strict header parsing, robust WAFs, and HTTP/2 adoption.
- **Should You Worry?**
  - Only if you're working with legacy systems or poorly configured environments. Proper security hardening minimizes this risk.