2025

RedShield

Generative AI

Exposing Risky Parsing Gaps in WAFs

Executive summary

Attackers have moved beyond simple signature evasion. Many successful attacks now exploit the way different parts of a web stack disagree about the meaning of the same request. Increasingly, generative AI tools allow attackers to automate and accelerate this process - rapidly generating and testing thousands of small request variants to probe for parsing differences that a WAF might miss.

A WAF may accept a request that the application subsequently interprets very differently, which lets malicious content slip through without raising an alert. RedShield's application security service includes in-flight security patches which address this by operating in the traffic path to normalize ambiguous inputs and apply targeted fixes to known weaknesses - without touching the application's code. This means transforming incoming HTTP requests so that any unusual, inconsistent, or non-standard formatting is cleaned up and made consistent before the request reaches the application.

The result is lower residual risk, faster time to mitigation, and clearer evidence that risky behavior is prevented.

Why WAF-only defenses struggle today

A modern web request passes through a CDN or proxy, a WAF, and finally the application framework. Each layer parses and interprets that request. If those interpretations differ, an attacker can craft inputs that look harmless to the WAF but resolve into something dangerous at the application. Generative AI makes it easier for adversaries to do this inexpensively and at scale. Typical examples include mixed or malformed content types, repeated parameters in a form, or conflicting headers that cause one system to read a body while another ignores it. None of this requires exotic techniques - only careful manipulation of edge cases that arise in the gaps between components.

The problem is not a lack of rules; it is that rules attached to an approximate understanding of a request do not control the behavior that matters. You end up with tuning cycles, exceptions for legitimate traffic, and lingering audit findings while engineering teams schedule code changes.

Example - WAFFLED: bypassing WAFs via parsing discrepancies

WAF bypasses aren't a simple patch or tune issue but stem from a core gap in many reverse-proxy-based WAF architectures - restricting their utility to threat management rather than vulnerability management (finding and fixing the weaknesses that attackers exploit).

What is it

WAFFLED is a research framework (Akhavani et al., 2025) that demonstrated 1,207 successful WAF bypasses by changing only the structure of HTTP bodies (multipart/form-data, JSON, XML) rather than the attack payload. The trick is to exploit tiny differences in how WAFs and web frameworks parse the same request.

How it works

Instead of hiding a malicious string, WAFFLED tweaks things like multipart boundaries or JSON field wrappers so the WAF parses one meaning (or gives up), while the backend framework parses another and accepts the payload. Because the payload itself is unchanged, signature rules still look "right" but the request slips through.

Why it matters

In default, vendor-recommended configurations, the study found bypasses across several major WAFs. For example, Cloudflare WAF (Pro plan, Managed Rules + OWASP CRS) was bypassed for multipart, JSON and XML. The authors also observed that over 90% of tested sites accept form-encoded and multipart bodies interchangeably - exactly the kind of ambiguity these attacks rely on.

Implication for your stack

This is not about any one vendor; it's about the gap between classification and interpretation. The defense is to normalize inputs before the app sees them and make ambiguous cases explicit. RedShield's in-flight patches performs this normalization in practice, enforcing strict message framing and content types, defining clear policies for duplicates, and re-serializing malformed bodies so the WAF and the application see the same request.

WAFFLED Example - Multipart Boundary Smuggling

```
POST / HTTP/1.1
Host: victim.com

Content-Type: multipart/form-data;
 boundary=fake;boundary*0=real-;boundary*1=boundary



--fake
Content-Disposition: form-data; name="field1"
value1
--fake--



--real-boundary
Content-Disposition: form-data; name="xss"
<script>alert(1)</script>
--real-boundary--
```

The WAF sees `boundary=fake`, ignores the rest, and misses the XSS. The Backend concatenates `boundary*0` and `boundary*1` → `real-boundary`, parses the XSS, and executes it.

This is not payload obfuscation - it's parser confusion.

Example - HPP as Semantic Smuggling

In a recent pentest, Bruno Mendes [Mendes, 2025] demonstrated how HPP can bypass even strict WAF configurations on an ASP.NET application:

```
GET /search?q=1'&q=alert(1)&q='2 HTTP/1.1
```

- WAF: Scans each q parameter in isolation. Sees `alert(1)` but misses context.
- Backend (ASP.NET): Concatenates values: 1',alert(1),'2
- Reflected in JavaScript:

```
userInput = '1',alert(1),'2';
```

- → Valid, executable code via the comma operator.

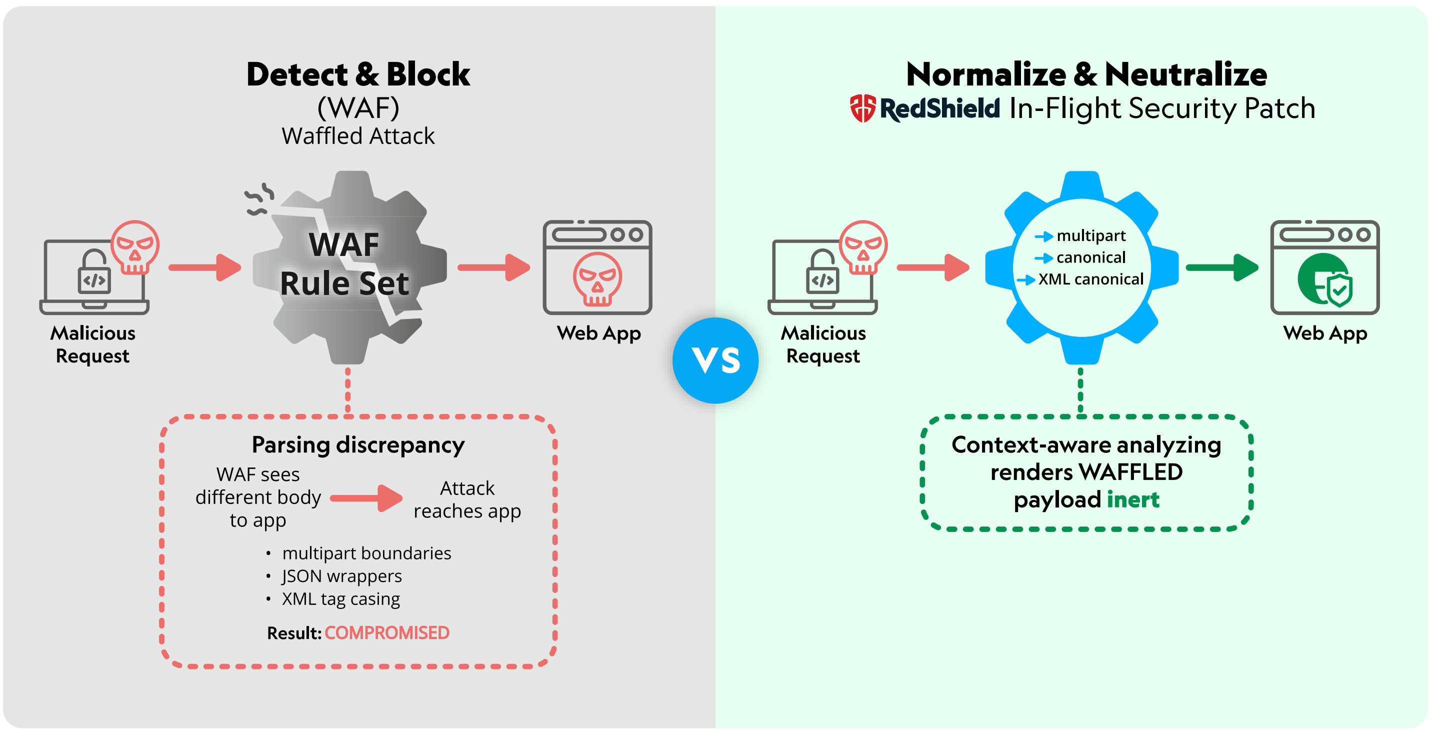
Like WAFFLED, this is not obfuscation - it's protocol-level smuggling, where the attack is assembled only at the application layer.

What RedShield does differently

RedShield adds a managed “fix” layer in the request/response path. Before traffic reaches the application, RedShield-developed in-flight security patches:

- **Normalize** inputs so that ambiguous or invalid encodings are made explicit or rejected.
- **Make intent clear** by defining policies for situations your frameworks handle in different ways (for example, what to do with duplicate parameters).
- **Apply targeted logic** to address known weaknesses on specific routes, such as enforcing strict content types, removing unexpected fields from JSON bodies, or inserting missing controls like anti-CSRF tokens.

Because changes are made in-flight, you do not need to change application code to close gaps. In-flight patches are scoped to the routes and behaviours that matter, rolled out safely, and removed once permanent fixes are in place in the applications, if you prefer.



Outcomes that matter to CISOs and CTOs

RedShield’s approach reduces exposure immediately while engineering keeps moving at its desired pace. In practice, teams see:

- Faster remediation of high-risk findings (hours or days, not multiple sprints).
- Fewer bypasses versus WAF-only because decisions are made at the same semantic level as the application.
- Lower tuning overhead and clearer run-books: policies are explicit and measured.
- Audit-ready evidence showing which routes are protected and which inputs are rejected.

A simple scenario

Consider a search form where the same field is submitted more than once. Some frameworks silently join those values; others pick the first or last. A WAF often inspects each value separately and sees nothing dangerous, while the joined value becomes harmful only inside the application. RedShield can enforce a clear policy - reject duplicates on sensitive routes, or emulate the exact behaviour your framework expects - so the risky combination never reaches the code.

How this fits into your environment

Deployment is straightforward. We start by reviewing existing findings and a small set of high-value flows. In a short observation phase we confirm where interpretation gaps occur and baseline false positives. We then deploy in-flight patches on those flows, initially in shadow or route-limited mode. Rollout is versioned and reversible. Ongoing dashboards show coverage by route, blocked attempts, and patch health, giving you a clean hand-off to engineering for permanent fixes when appropriate.

Questions to ask your team

- Do our WAFs parse requests differently to our applications?
- Which endpoints accept multiple content types or allow repeated fields that could mask intent?
- Which open findings could be mitigated today without changing application code?

Next step

Run a limited-route trial. We will normalize inputs and apply targeted patches “out of path” (no production impact) for one web application, then report the measured reduction in risk and operational effort. If it meets your bar, move the patches into the production path, and extend coverage across other high-value routes while engineering focuses on permanent fixes where they add the most value.

Learn More

For a deeper dive into the challenges of web application security and how RedShield helps organizations address them, download our whitepaper and visit us at **RedShield.co**.