HTTP Request Smuggling: Uma Análise Técnica da Vulnerabilidade em Aplicações Web

Leonardo Teodoro[1]

Resumo

O ataque HTTP Request Smuggling (HRS) explora inconsistências na interpretação de cabeçalhos HTTP entre servidores intermediários e servidores de aplicação, permitindo a injeção e manipulação de requisições maliciosas. Este artigo apresenta uma análise técnica aprofundada sobre o funcionamento do HRS, abordando suas variações, como CL.TE e TE.CL, além dos impactos práticos, como bypass de autenticação, envenenamento de cache e comprometimento de sessões. São também discutidas estratégias eficazes de mitigação, com foco em configurações seguras de servidores e proxies, conformidade com as RFCs do protocolo HTTP, e ferramentas de detecção específicas. A vulnerabilidade, embora pouco visível, representa um risco crítico em ambientes distribuídos, exigindo atenção constante por parte de desenvolvedores e administradores de sistemas.

Palavras-chave: Segurança Web. HTTP Request Smuggling. Proxies. Cabeçalhos HTTP. OWASP.

Abstract

The HTTP Request Smuggling (HRS) attack exploits inconsistencies in the interpretation of HTTP headers between intermediary servers and application servers, enabling the injection and manipulation of malicious requests. This paper presents a comprehensive technical analysis of HRS, covering its variants, such as CL.TE and TE.CL, as well as practical impacts, including authentication bypass, cache poisoning, and session compromise. Effective mitigation strategies are also discussed, focusing on secure configurations for servers and proxies, compliance with HTTP protocol RFCs, and specific detection tools. Although not widely known, this vulnerability poses a critical risk in distributed environments, demanding constant attention from developers and system administrators.

Keywords: Web Security. HTTP Request Smuggling. Proxies. HTTP Headers. OWASP.

[1] Especialista em Pentest. Mestrando em Ciência da Computação na UTFPR. E-mail: teodoroleo96@gmail.com

1 INTRODUÇÃO

A crescente complexidade das arquiteturas web modernas tem ampliado a superfície de ataque disponível para agentes maliciosos. Em especial, o uso de proxies reversos, balanceadores de carga e servidores intermediários em aplicações web tem tornado essas estruturas suscetíveis a vulnerabilidades menos evidentes, como o **HTTP Request Smuggling** (HRS).

O HRS é uma técnica de ataque que explora discrepâncias na interpretação dos cabeçalhos HTTP entre diferentes camadas de servidores - geralmente entre um front-end (como um proxy) e um back-end (como o servidor de aplicação). Ao manipular campos como *Content-Length* e *Transfer-Encoding*, um atacante é capaz de “contrabandear” requisições maliciosas para dentro de requisições aparentemente legítimas, contornando mecanismos de autenticação, firewalls de aplicação (WAFs) e outras barreiras de segurança (PORTSWIGGER, 2024).

Embora a técnica tenha sido documentada pela primeira vez em 2005, ela continua relevante, especialmente em infraestruturas que não seguem rigorosamente as especificações da RFC 7230 (IETF, 2014). Casos reais reportados em grandes empresas demonstram o potencial do HRS para comprometer dados sensíveis, manipular sessões e realizar escalonamentos de privilégio (BRIGHT SECURITY, 2024).

Este artigo tem como objetivo apresentar uma análise técnica e descritiva do ataque HTTP Request Smuggling, discutindo seus princípios de funcionamento, técnicas de exploração, impactos potenciais e estratégias de mitigação. Serão utilizados exemplos práticos para facilitar a compreensão da mecânica do ataque e de suas possíveis defesas.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O Protocolo HTTP e sua Arquitetura

O protocolo HTTP (Hypertext Transfer Protocol) é a base da comunicação na web, operando segundo um modelo cliente-servidor. Cada requisição HTTP é composta por uma linha de requisição, cabeçalhos e um corpo opcional, sendo que a forma como os cabeçalhos são interpretados influencia diretamente o comportamento do servidor (FIELDING et al., 1999).

A versão 1.1 do protocolo (HTTP/1.1), especificada na RFC 2616 e posteriormente atualizada pela RFC 7230, introduziu funcionalidades como conexões persistentes e suporte ao cabeçalho *Transfer-Encoding*, que permite a transmissão de dados em blocos (chunked

encoding). Essas inovações, embora benéficas para o desempenho, também geraram brechas de segurança exploráveis, como é o caso do HTTP Request Smuggling.

2.2 Arquiteturas Web e a Interpretação de Cabeçalhos

Sistemas modernos frequentemente utilizam arquiteturas compostas por diversos servidores em camadas. Um cliente pode se comunicar com um proxy reverso ou balanceador de carga (front-end), que então repassa a requisição ao servidor de aplicação (back-end). Essa separação de responsabilidades pode levar a divergências na interpretação de requisições HTTP, sobretudo quando os servidores envolvidos utilizam regras distintas para processar os cabeçalhos *Content-Length* e *Transfer-Encoding*.

2.3 O Ataque HTTP Request Smuggling

O HTTP Request Smuggling (HRS) ocorre quando um atacante envia uma requisição HTTP maliciosa que, devido à ambiguidade nos cabeçalhos, é interpretada de forma diferente pelos servidores front-end e back-end. Essa discrepância permite “injetar” uma requisição dentro de outra, ocultando a requisição maliciosa do primeiro servidor.

A técnica foi descrita inicialmente por Linhart et al. (2005) e ganhou maior notoriedade após ser amplamente estudada por pesquisadores da PortSwigger Web Security. Existem diferentes variantes do ataque, classificadas de acordo com os cabeçalhos manipulados:

- **CL.TE (Content-Length / Transfer-Encoding)**: O servidor front-end usa *Content-Length*, enquanto o back-end interpreta *Transfer-Encoding*.
- **TE.CL (Transfer-Encoding / Content-Length)**: O inverso do anterior; o front-end usa *Transfer-Encoding* e o back-end, *Content-Length*.
- **TE.TE (Transfer-Encoding duplo)**: Ambas as camadas interpretam *Transfer-Encoding*, mas de maneira distinta.

Essas variações aproveitam a falta de conformidade com as especificações da RFC 7230, que determina que o uso simultâneo de *Content-Length* e *Transfer-Encoding* em uma mesma requisição é inválido. No entanto, muitos servidores ainda aceitam tais requisições, tratando-as de acordo com suas próprias regras de precedência.

2.4 Consequências e Vetores de Exploração

A exploração do HRS pode permitir a realização de diversos ataques, como:

- Bypass de autenticação e WAFs;
- Fixação ou sequestro de sessão (session fixation/hijacking);
- Ataques de cross-site scripting (XSS) indireto;
- Abuso de cache e envenenamento de respostas (HTTP Response Smuggling).

A gravidade do ataque está diretamente ligada ao papel do servidor intermediário e à natureza dos dados e permissões envolvidos.

3 FUNCIONAMENTO DO ATAQUE HTTP REQUEST SMUGGLING

O ataque HTTP Request Smuggling (HRS) baseia-se na manipulação de cabeçalhos da requisição HTTP de modo a induzir os servidores de uma arquitetura em camadas a interpretarem de forma distinta os limites e o conteúdo de uma mesma requisição. Isso permite que o atacante injete comandos ou requisições inteiras “escondidas” entre requisições legítimas.

3.1 Desalinhamento entre Front-End e Back-End

Em muitas aplicações web, uma requisição enviada pelo cliente é primeiramente tratada por um proxy reverso (como NGINX ou HAProxy) antes de ser encaminhada ao servidor de aplicação (como Apache, Tomcat ou Node.js). Quando o proxy e o servidor interpretam os cabeçalhos de forma divergente - por exemplo, ao decidir qual delimitação de corpo utilizar entre *Content-Length* e *Transfer-Encoding* - o atacante pode se aproveitar dessa ambiguidade.

3.2 Variações do Ataque

As variantes mais comuns do HTTP Request Smuggling envolvem a manipulação combinada dos cabeçalhos *Content-Length* (CL) e *Transfer-Encoding* (TE):

CL.TE: O front-end utiliza *Content-Length* para interpretar a requisição, enquanto o back-end utiliza *Transfer-Encoding: chunked*.

TE.CL: O oposto; o front-end utiliza *Transfer-Encoding*, mas o back-end usa *Content-Length*.

TE.TE: Ambos os servidores interpretam *Transfer-Encoding*, mas de forma diferente (por exemplo, um ignora o valor inválido, o outro interpreta como chunked).

3.3 Execução Prática de um Ataque CL.TE

Considere o seguinte cenário: o atacante envia uma requisição HTTP contendo cabeçalhos tanto *Content-Length* quanto *Transfer-Encoding*, sendo que o proxy utiliza CL e o servidor usa TE. O atacante pode, então, inserir uma segunda requisição maliciosa no corpo da requisição original. O proxy interpreta apenas a primeira, enquanto o servidor back-end processa ambas, expondo funcionalidades ou dados restritos.

Abaixo, um exemplo real de payload utilizado em um ataque do tipo CL.TE:

```
POST / HTTP/1.1
Host: vulneravel.com
Content-Length: 6
Transfer-Encoding: chunked

0

GET /admin HTTP/1.1
Host: vulneravel.com
```

Essa requisição será tratada de maneira diferente pelos servidores envolvidos. O proxy considerará apenas a parte delimitada por *Content-Length: 6*, descartando o restante. Já o servidor de aplicação reconhecerá o uso de *Transfer-Encoding* e interpretará toda a sequência, processando o *GET /admin*.

O resultado pode ser a execução de uma requisição privilegiada, como acesso não autorizado ao painel de administração, sem que os mecanismos de segurança no proxy percebam a anomalia.

4 IMPACTOS E CASOS REAIS

O ataque HTTP Request Smuggling (HRS) representa uma ameaça significativa à segurança de aplicações web, especialmente em infraestruturas compostas por múltiplos intermediários HTTP. A possibilidade de desviar, manipular ou injetar requisições de maneira encoberta impacta diretamente a integridade, a confidencialidade e a disponibilidade dos serviços.

4.1 Consequências Práticas do HRS

Os impactos mais comuns decorrentes de um ataque bem-sucedido de request smuggling incluem:

- **Bypass de autenticação e autorização**: O atacante pode acessar rotas ou funcionalidades que, em circunstâncias normais, estariam protegidas por autenticação no front-end.
- **Session hijacking e session fixation**: Ao capturar ou forçar sessões de outros usuários, o atacante pode assumir identidades dentro do sistema.
- **Envenenamento de cache (Cache Poisoning)**: Requisições maliciosas manipuladas podem comprometer o conteúdo armazenado em cache, afetando todos os usuários subsequentes.
- **Execução de comandos no back-end**: Em casos extremos, o atacante pode induzir o back-end a executar comandos não intencionais, resultando em comprometimento completo do sistema.

4.2 Casos Reais e Reconhecimento pela Indústria

Em 2020, pesquisadores da PortSwigger identificaram uma série de implementações vulneráveis em servidores amplamente utilizados, como Amazon Web Services (AWS), Apache, HAProxy e até o Cloudflare (PORTSWIGGER, 2020). Essas descobertas mostraram que mesmo empresas com forte investimento em segurança estão suscetíveis à técnica, sobretudo quando há envolvimento de múltiplas camadas HTTP.

Além disso, diversas plataformas de bug bounty - como HackerOne e Bugcrowd - registram relatórios recorrentes de vulnerabilidades explorando HRS, muitas vezes com recompensas elevadas dadas a sua criticidade (HACKERONE, 2024). Em alguns desses casos, os ataques permitiram a execução de requisições administrativas em grandes portais, extração de dados internos e interrupção parcial de serviços.

4.3 Detecção e Dificuldades Operacionais

A detecção do HRS pode ser extremamente desafiadora. Ferramentas como BurpSuite Professional possuem módulos específicos para varredura de vulnerabilidades relacionadas a request smuggling, mas a identificação requer atenção a detalhes como:

- Divergência entre resposta esperada e real;
- Requisições que geram efeitos fora da sequência normal;
- Logs no back-end com requisições inesperadas ou truncadas.

Na prática, é comum que logs de proxies e servidores não estejam alinhados, dificultando a reconstrução precisa do ataque e atrasando ações de contenção.

5 ESTRATÉGIAS DE MITIGAÇÃO E BOAS PRÁTICAS

A prevenção ao ataque HTTP Request Smuggling exige medidas em diversas camadas da aplicação, especialmente nas que envolvem proxies reversos, balanceadores de carga e servidores de aplicação. A ausência de conformidade com as RFCs do protocolo HTTP e a interpretação ambígua de cabeçalhos são os principais facilitadores dessa vulnerabilidade.

5.1 Conformidade com as Especificações HTTP

Conforme definido pela RFC 7230 (FIELDING et al., 2014), requisições HTTP não devem conter simultaneamente os cabeçalhos *Content-Length* e *Transfer-Encoding*. Implementações devem rejeitar tais requisições com códigos de erro apropriados, como *400 Bad Request*.

É recomendado que proxies e servidores web modernos sigam as diretrizes mais atualizadas e estejam configurados para aplicar essas regras de forma estrita. Em muitos casos, atualizações de software já incluem correções de comportamento em relação à interpretação dos cabeçalhos.

5.2 Configurações Seguras em Proxies e Servidores

Proxies reversos (como NGINX, HAProxy e Apache HTTPD com mod_proxy) devem ser configurados com atenção especial ao tratamento de requisições com múltiplos cabeçalhos de delimitação de corpo.

Exemplos de medidas específicas incluem:

- No **NGINX**: usar a diretiva *ignore_invalid_headers on* e garantir a rejeição de requisições com cabeçalhos conflitantes.
- No **HAProxy**: configurar a política *h1-use-cl 1* para forçar o uso exclusivo de *Content-Length* em conexões HTTP/1.
- No **Apache HTTPD**: evitar o uso do *mod_proxy* em conjunto com servidores que não validam corretamente os cabeçalhos.

Além disso, o uso de web application firewalls (WAFs) pode fornecer uma camada adicional de proteção, desde que estejam atualizados para detectar variações de request smuggling.

5.3 Validação de Entrada e Normalização

Aplicações devem implementar rotinas de sanitização e validação de cabeçalhos HTTP, mesmo que essa camada esteja atrás de proxies ou balanceadores. Cabeçalhos desconhecidos, repetidos ou malformados devem ser tratados com rigor.

Além disso, é importante normalizar as requisições recebidas antes de processá-las internamente. Essa prática reduz o risco de interpretações divergentes entre as camadas da arquitetura.

5.4 Testes de Segurança e Ferramentas

Ferramentas como **BurpSuite**, **Smuggler.py** e scanners automatizados podem ser utilizadas para detectar a presença de falhas relacionadas ao HRS. Testes periódicos devem ser incluídos no ciclo de desenvolvimento seguro (DevSecOps), especialmente em aplicações que utilizam infraestrutura distribuída ou serviços de terceiros.

Além disso, é importante realizar varreduras com diferentes tipos de payloads, incluindo variações que simulam ataques reais, pois muitos mecanismos de defesa falham ao detectar variantes menos comuns.

5.5 Educação da Equipe e Monitoramento Contínuo

A conscientização dos desenvolvedores e administradores de sistemas sobre o funcionamento do HRS é fundamental para reduzir a superfície de ataque. Além disso, o monitoramento contínuo de logs de acesso e comportamento anômalo pode ser essencial para detectar ataques que tenham passado despercebidos pelas camadas automatizadas de defesa.

6 CONSIDERAÇÕES FINAIS

O ataque HTTP Request Smuggling representa uma ameaça sutil, porém altamente impactante, principalmente em arquiteturas web modernas compostas por múltiplas camadas de proxies e servidores. Sua efetividade está diretamente ligada à forma como diferentes componentes da cadeia HTTP interpretam os cabeçalhos das requisições — uma falha muitas vezes imperceptível durante os testes convencionais de segurança.

Ao longo deste artigo, foi possível demonstrar o funcionamento da vulnerabilidade por meio de exemplos práticos e explorar os impactos que esse tipo de ataque pode causar, como acesso não autorizado, envenenamento de cache e comprometimento de sessões. Também foram apresentadas estratégias de mitigação que abrangem desde a conformidade com especificações HTTP até práticas seguras de configuração e validação.

Embora o HTTP Request Smuggling não seja um vetor de ataque amplamente conhecido fora dos círculos especializados em segurança ofensiva, ele permanece altamente relevante, especialmente em sistemas legados ou mal configurados. A conscientização sobre sua existência e a implementação de contramedidas apropriadas devem ser prioridades para equipes técnicas, desenvolvedores e administradores de infraestrutura.

Portanto, mais do que uma ameaça conceitual, o HTTP Request Smuggling é um reflexo das complexidades da comunicação entre sistemas distribuídos e da importância de uma abordagem de segurança proativa, contínua e colaborativa.

Referências

FIELDING, R.; RESCHKE, J. _Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing_. RFC 7230, jun. 2014. Disponível em: https://datatracker.ietf.org/doc/html/rfc7230. Acesso em: 20 abr. 2025.

HACKERONE. _Smuggling Past Proxies with HTTP Desync Attacks_. 2024. Disponível em: https://www.hackerone.com/blog/Smuggling-Past-Proxies-with-HTTP-Desync-Attacks. Acesso em: 20 abr. 2025.

PORTSWIGGER. _HTTP Request Smuggling_. 2020. Disponível em: https://portswigger.net/web-security/request-smuggling. Acesso em: 20 abr. 2025.

OWASP. _OWASP Top 10 Mobile 2024_. 2024. Disponível em: https://owasp.org/www-project-mobile-top-10/. Acesso em: 20 abr. 2025.

OWASP FOUNDATION. _OWASP Testing Guide v4_. 2014. Disponível em: https://owasp.org/www-project-web-security-testing-guide/. Acesso em: 20 abr. 2025.