UiO : Department of Informatics
University of Oslo

Less is Often More: Header Whitelisting as Semantic Gap Mitigation in HTTP-based Software Systems

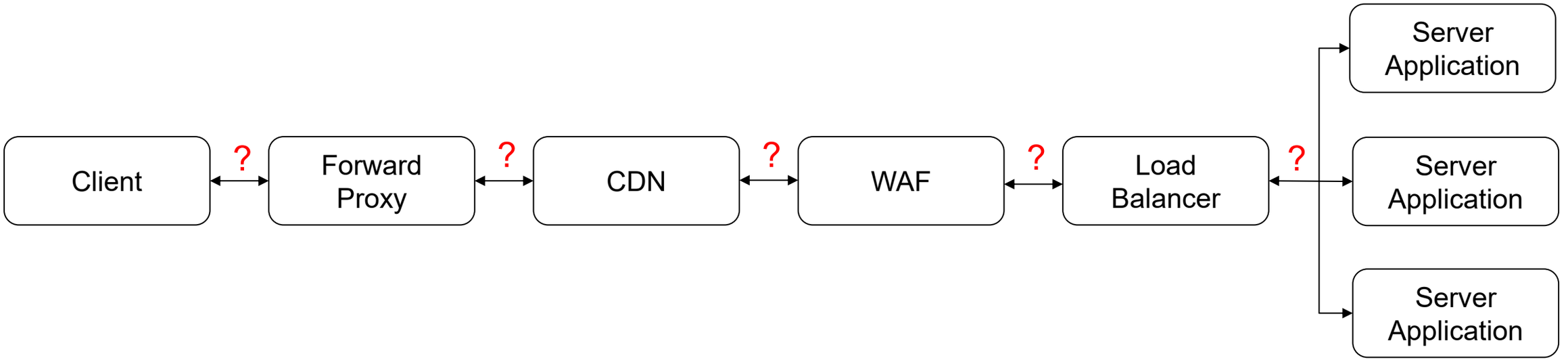
Andre Büttner*, Hoai Viet Nguyen†, Nils Gruschka* and Luigi Lo Iacono†

* University of Oslo † Hochschule Bonn-Rhein-Sieg

ifip SEC2021 OSLO NORWAY 23 June 2021

Motivation

- High complexity of the web due to intermediaries
  - E.g. caches, web application firewalls (WAFs) and load balancers
  - Every entity may have its own HTTP implementation/library



- Attacks

Attack	Embodiment
Response Splitting [Klein, 2004]	URL
Request Smuggling [Linhart *et al.*, 2005], [Kettle, 2019]	Header
Host-of-Trouble (HoT) [Chen *et al.*, 2016]	Header
Cache-Poisoned Denial of Service (CPDoS) [Nguyen *et al.*, 2019]	Header
Hop-by-Hop [Davison, 2019]	Header
Web Cache Deception [Gil, 2017], [Mirheidari *et al.*, 2020]	URL

Agenda

- Semantic Gaps in HTTP Message Processing
- Header Whitelisting
- Evaluation
- Discussion
- Conclusion

Semantic Gaps in HTTP Message Processing

- Definition:
  Inconsistent processing of HTTP messages inside a pipeline between the actual application logic and the intermediaries.

- Root causes:
  - Ambiguities within the HTTP standard
    - E.g.: duplicate header fields, no header size limit, non-standardized header fields
  - Improper implementations
    - E.g.: incorrect parsing of HTTP messages
  - Different HTTP versions used
    - HTTP/1.1: RFC 2616, RFC 7230
    - HTTP/1.1, HTTP/2, HTTP/3

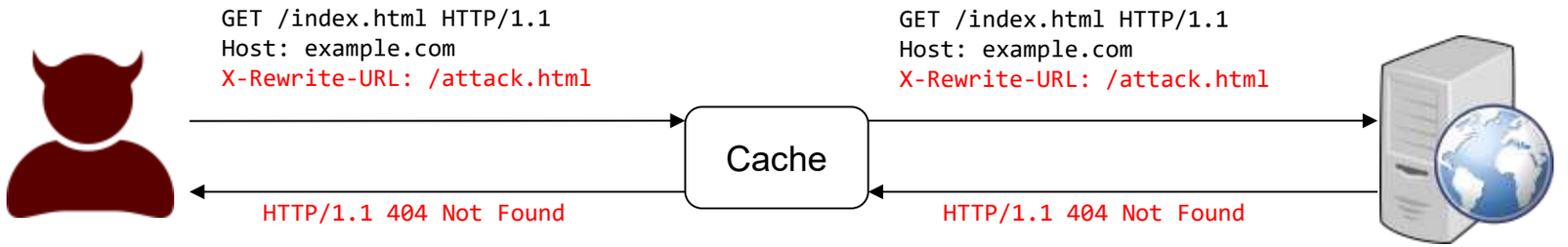
- Mitigation
  - Proposed countermeasures usually address only one of these attacks
  - WAFs have their own drawbacks and do also not prevent all attacks
  - A more holistic view is required to reduce the attack surface (cf. [Mirheidari *et al.*, 2020])

- Our solution
  - Header Whitelisting (HWL) as an effective measure to mitigate the Semantic Gap

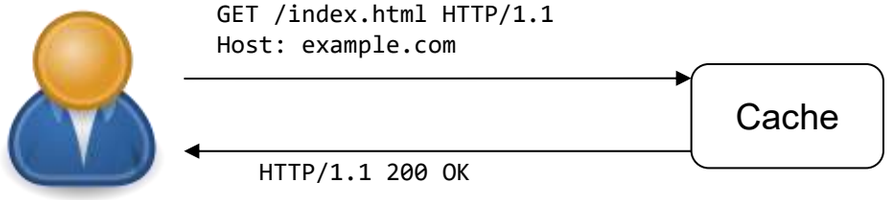
Header Whitelisting

- Concept
  - Reducing the HTTP request header to the minimum required header fields
    → Remove header fields that are not whitelisted
  - Apply approach to each component separately
    → Append header fields back to the request before forwarding to the next component
  - Enforce strictly standard compliant header parsing
    → Reject requests that include invalid syntax / meta characters

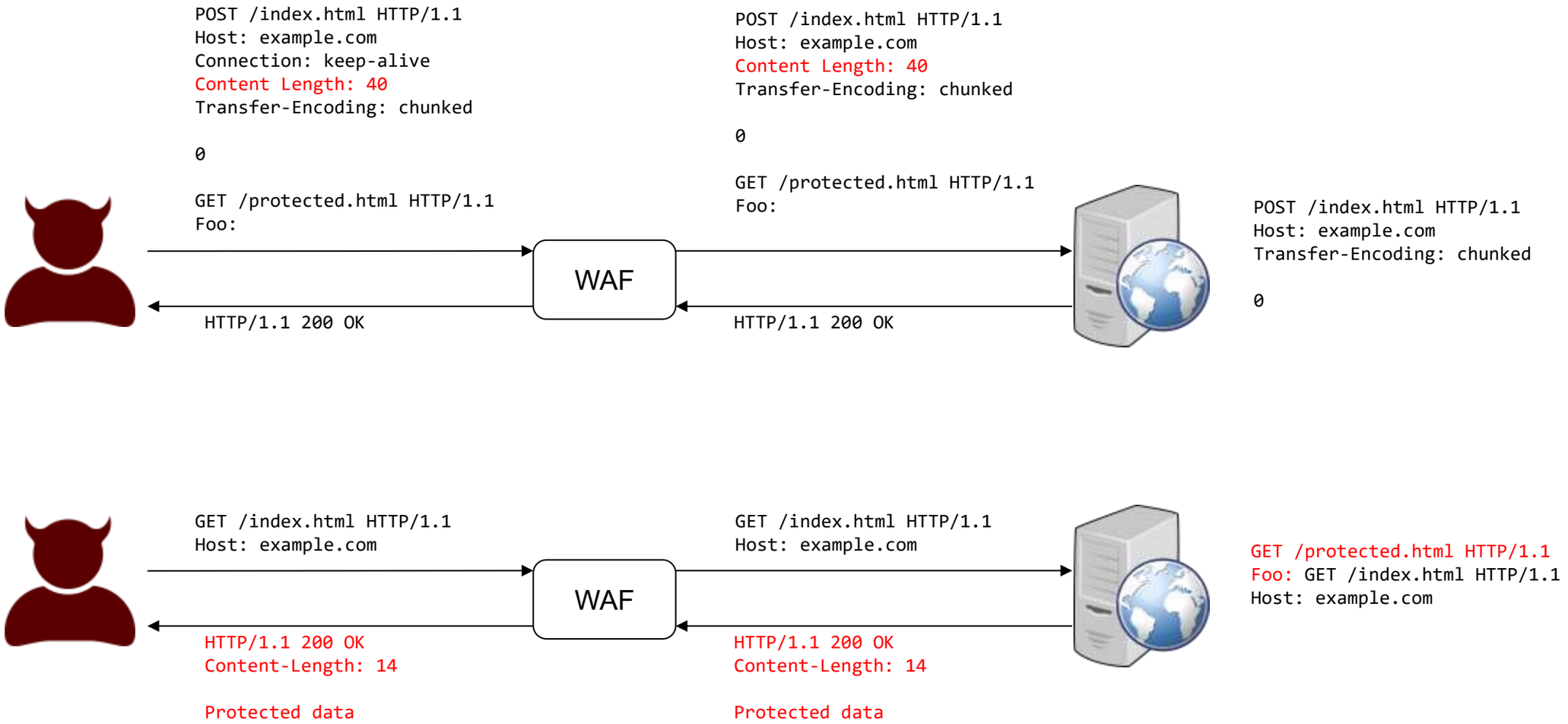
- Example 1: CPDoS



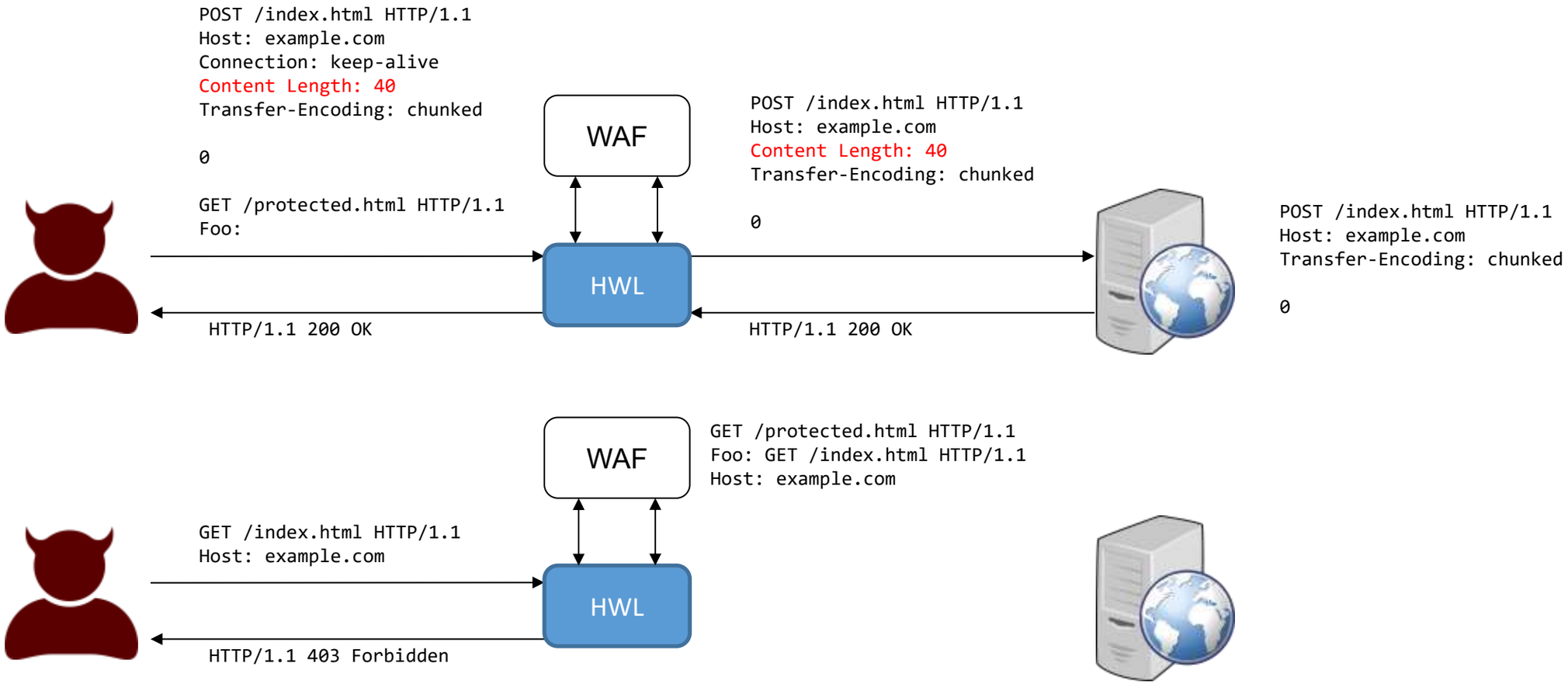
- Example 1: CPDoS with HWL



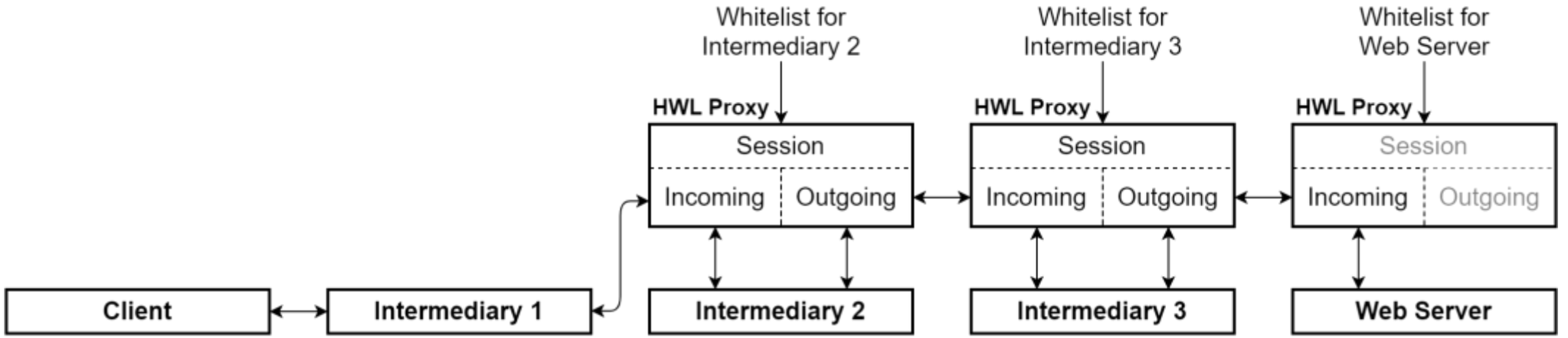
- Example 2: Request Smuggling



- Example 2: Request Smuggling with HWL

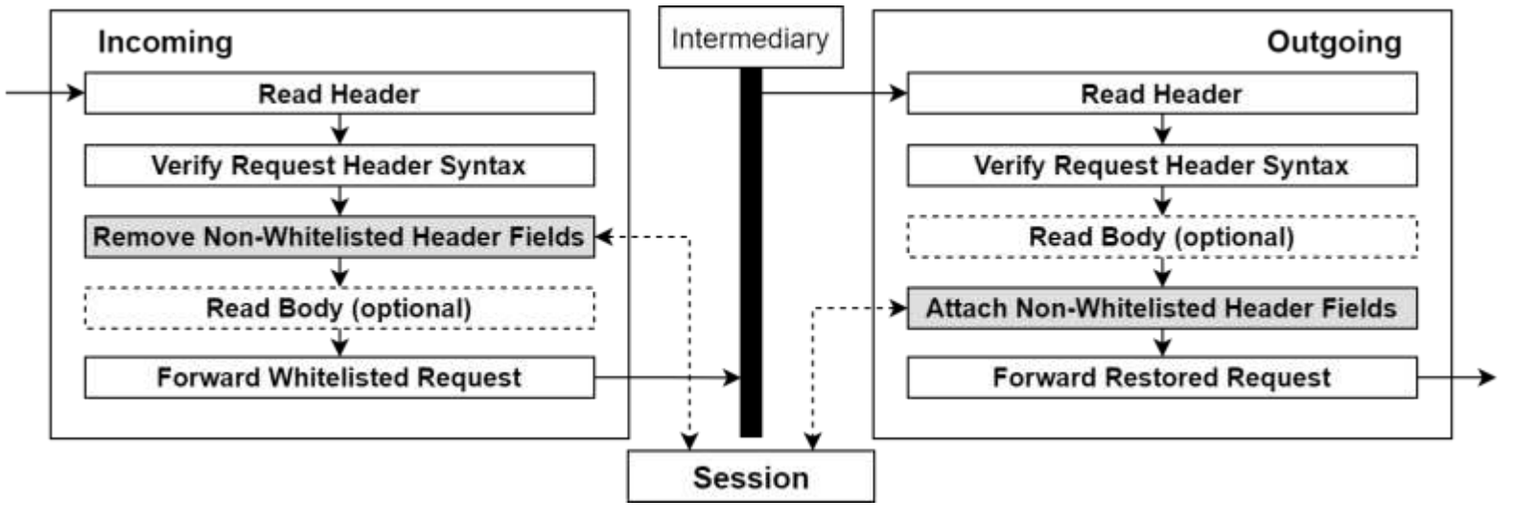


- Architecture



  - Incoming module
    - Sanitize HTTP request from non-whitelisted header fields
  - Session module
    - Assign ID to HTTP requests and store corresponding non-whitelisted header fields
  - Outgoing module
    - Attach non-whitelisted header fields back to request

- Prototype Implementation
  - HWL Proxy
  - Go Programming Language
  - Implementation overview:



- Header whitelist specified in JSON file

```
[
    {
        "key": "host"
    },
    {
        "key": "connection",
        "val": "(close|keep-alive)"
    },
    {
        "key": "content -length",
        "val": "\\d+"
    }
]
```

```
POST /index.html HTTP/1.1
Host: example.com
~~Connection: invalid~~
Content-Length: 5

abcdef
```

```
GET /index.html HTTP/1.1
Host: example.com
Connection: close
~~X-Forwarded-Host: evil.org~~
```

- Source code: https://github.com/Digital-Security-Lab/hwl-proxy

Evaluation

- Methodology
  - Recreation of attacks in a lab environment
  - Checking if attacks are prevented in case HWL is deployed

- Test Environment
  - Three virtual server instances (Ubuntu 16.04 LTS)
    - Client
    - Intermediary
    - Web Server
  - Different proxies and test server applications to recreate attack scenarios

- Test cases

ID	Attack type	Causing header	Intermediary	Web Server
TC1	Request Smuggling	Content-Length	ATS 7.1.2	NodeJS 4.1.2
TC2	Request Smuggling	Transfer-Encoding + <SP>	ATS 7.1.2	NodeJS 4.1.2
TC3	Request Smuggling	X-Rewrite-Url	NGINX 1.1.15	Symfony 3.4.0
TC4	CPDoS	X-Original-Url	Varnish 6.3.1	Symfony 3.4.0
TC5	CPDoS	X-HTTP-Method-Override	Varnish 6.3.1	Play 1.5.0
TC6	Hop-by-Hop	Connection	Varnish 3.0.0	NodeJS 4.1.2
TC7	HoT	Host	ATS 7.1.2	Rails 5.2.0

- Test results

Intermediary	Server	TC1	TC2	TC3	TC4	TC5	TC6	TC7
○	○	⊖	⊖	⊖	⊖	⊖	⊖	⊖
○	●	⊕	⊕	⊕	⊕	⊕	⊖	⊕
●	○	⊕	⊖	⊖	⊖	⊖	⊕	⊖
●	●	⊕	⊕	⊕	⊕	⊕	⊕	⊕

○ HWL disabled
● HWL enabled

⊕ attack prevented
⊖ attack succeeded

Discussion

- Strengths
  - All attacks could be prevented
  - Compatible with HTTP components
  - Zero-day exploits may be mitigated
- Limitations
  - Request URL and HTTP responses not considered
  - Evaluation only includes attacks, where an effect was expected
  - CDNs were not tested
- Vulnerabilities
  - Parsing errors may still occur
  - Vulnerable against DoS attacks
- Whitelist specification
  - Incorrect configuration can cause malfunction
  - Automatic whitelist creation
  - Default configuration should be provided
- Deployment
  - Integration into existing HTTP libraries
  - HWL as Software-as-a-Service

Conclusion

- HWL is proposed as a measure to mitigate a broad range of attacks
- We have implemented and evaluated a prototype
- The results show that attacks can be prevented effectively
- Future work:
  - Evaluate and improve performance
  - Standardize approach
  - Development of advanced features (e.g.: automatic whitelist, ACLs, etc.)

References

- Klein, Amit. *"Divide and conquer." HTTP Response Splitting, Web Cache Poisoning Attacks and Related Topics, Sanctum whitepaper* (2004).
- Linhart, C., Klein, A., Heled, R., Steve, O.: *Http request smuggling* (2005). https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf
- Kettle, J.: *Http desync attacks: Request smuggling reborn* (2019). https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn
- Chen, J., Jiang, J., Duan, H., Weaver, N., Wan, T., Paxson, V.: *Host of troubles: multiple host ambiguities in http implementations*. In: 23th ACM SIGSAC Conference on Computer and Communications Security (CCS) (2016)
- Nguyen, H.V., Lo Iacono, L., Federrath, H.: *Your cache has fallen: cache-poisoned denial-of-service attack*. In: 26th ACM Conference on Computer and Communications Security (CCS) (2019)
- Davison, N.: *Abusing http hop-by-hop request headers* (2019). https://nathandavison.com/blog/abusing-http-hop-by-hop-request-headers
- Gil, O.: *WEB CACHE DECEPTION ATTACK*. In: Blackhat USA (2017). https://blogs.akamai.com/2017/03/on-web-cache-deception-attacks.html
- Mirheidari, S.A., Arshad, S., Onarlioglu, K., Crispo, B., Kirda, E., Robertson, W.: *Cached and confused: web cache deception in the wild*. In: 29th USENIX Security Symposium (USENIX Security) (2020)

Thank you!