A P4-Based Content-Aware Approach to Mitigate Slow HTTP POST Attacks

EuroP4 - December 9, 2022

Chih-Yu Hsieh,
Hong-Yen Chen,
Shan-Hsiang Shen,
Chen-Hsiang Hung,
Tsung-Nan Lin
Agenda

- Introduction
- Proposed Method
- Experiments and Results
- Conclusion
Slow HTTP DDoS Attacks

Slow HTTP DDoS attacks disturb services by occupying server threads with

- HTTP headers: slowloris / slow header
- HTTP body: slow POST / slow body / RUDY

Sending body simulates **realistic file upload**

<table>
<thead>
<tr>
<th>POST /posts HTTP/1.1</th>
<th>start-line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host: 10.0.1.1</td>
<td>headers</td>
</tr>
<tr>
<td>User-Agent: Mozilla/4.0</td>
<td></td>
</tr>
<tr>
<td>Content-Length: 7</td>
<td></td>
</tr>
<tr>
<td>Content-Type: application/x-www-form-urlencoded</td>
<td>body</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>slowloris</th>
<th>slow POST</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment</td>
<td>HTTP header</td>
</tr>
<tr>
<td>expected size</td>
<td>small</td>
</tr>
</tbody>
</table>
Challenge of Detection

● How to distinguish attackers from clients correctly in various network activities?
  ○ Viewing websites
  ○ Uploading photos / videos
  ○ Filling forms
  ○ Slow HTTP attack

● Existing works
  ○ timeout methods [1-3]
  ○ credibility method [4]

● False positives make legitimate users suffer from denial-of-service

Timeout-based Defense Mechanism

Files are corrupt because the user cannot finish uploading within the timeout.
Credibility-based Defense Mechanism

The file to upload cannot be completed within the specified number of packets

Size: 300 MB

split file into packets

credit = 100

credit = 98

credit = 0

blocked by fragmented packets

received size < 6.4 MB
Contribution

Our proposed method, RASP, is an open source\(^1\), P4-based content-aware countermeasure.

- **High accuracy**: overcome the false positive issue by utilizing HTTP information
- **Scalable** deployment by P4
  - Application-layer headers processing is distributed to switches
  - Quantifies network usage savings
- Demonstrates the ability of P4 to parse variable-length header fields

\(^1\) [https://github.com/doraeric/p4-rasp](https://github.com/doraeric/p4-rasp)
Method Overview

1. **GET requests**
   - client sends HTTP req
   - identify method
   - classify long / short-term from headers

2. **short-term**
   - finish
   - # of conn
   - timeout

3. **long-term**
   - finish
   - # of conn

1. **complete**
   - finish
   - # of conn

① client sends HTTP req
② identify method
③ classify long / short-term from headers
④ apply limitations
⑤ punish bad client

4xx (Failed)
other
response code

Server

Close new connection

Request: Initial Protection

Response: Punishment at the End

RASP overview
Initial Protection

- Limitation per client per category
  - **complete**: none
  - **short-term**: number of requests is 8, connection time < 10 seconds
  - **long-term**: number of requests is 4

- Close excess connections and keep old ones.
  
The user needs to finish old requests first.
Punishment at the End

- HTTP status code can indicate whether a request is successful.
  - 2xx: the backend processes the request without error
  - 4xx: the request failed due to client error (malformed / invalid request)
- Punishment is to decrease the number of allowed connections.

HTTP/1.1 200 OK
Server: Apache/2.4.25 (Debian)
Content-Type: text/html
...

HTTP/1.1 400 Bad Request
Server: Apache/2.4.25 (Debian)
Content-Type: text/html
...

A good HTTP response.  
A bad HTTP response.
Implementation

- Control plane
  - manage connection state

- Data plane
  - parse HTTP headers
  - manage the number of open connections with register
  - report to controller with digest messages

```c
struct headers_t {
    char_header_t[200] http_buffer;
};
```
Experiments - Simulation Scenario

We simulate different usage scenarios to verify the robustness of RASP:

1. short GET: slow client viewing websites under a slow header attack
2. long non-GET: clients uploading several photos under a slow POST attack
3. short non-GET: clients uploading GPS locations under a slow POST attack.

We investigate

- the number of successful requests the clients send
- reduction in network usage by adopting P4

Experiment with BMv2
1. Slow Header Attack

Short-term GET clients under slow header attack

Our proposed RASP mitigates attacks **earlier** by sending TCP RST.
2. POST Photos

Long-term non-GET clients under slow POST attacks

RASP **correctly** completes all client requests in time

Table. Received files by backend

<table>
<thead>
<tr>
<th>Method</th>
<th>receive bytes</th>
<th>complete files</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHDA</td>
<td>43.3 MB</td>
<td>0</td>
</tr>
<tr>
<td>CCSA</td>
<td>1.7 MB</td>
<td>0</td>
</tr>
<tr>
<td>RASP</td>
<td><strong>129 MB</strong></td>
<td>60</td>
</tr>
</tbody>
</table>

Graph:ulative chart showing the number of connections over time for different methods. The chart indicates:
- **Apache**: Takes more than 300 seconds.
- **SHDA**: Clients disconnected due to timeout.
- **CCSA**: Clients blocked due to low credit.
- **RASP**: Finish requests at $t = 230$. 


3. Upload GPS Locations

Short-term non-GET clients under slow POST attacks

RASP **correctly** protects clients from DDoS attacks.

Table. Received requests by backend

<table>
<thead>
<tr>
<th>Method</th>
<th># of req</th>
<th>success</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHDA</td>
<td>1782</td>
<td>99%</td>
</tr>
<tr>
<td>CCSA</td>
<td>300</td>
<td>16.7%</td>
</tr>
<tr>
<td>RASP</td>
<td>1800</td>
<td>100%</td>
</tr>
</tbody>
</table>

clients keep waiting

curve raises quickly, becomes DDoS

clients blocked due to low credit

receive all client requests
Network Usage

- Send smaller digests messages than raw packets
  - Raw: between switches and clients, including attackers
  - P4RT: digest messages between switches and controller
- Digest message (P4) compared to raw packets (OpenFlow)
  - Number of packets -> approximately 30%
  - Number of bytes -> 20%
- Larger HTTP body benefit more (exp2, 0.74% / 0.1%)
Conclusion

- We propose RASP, a defense mechanism against slow HTTP POST DDoS attacks. RASP utilizes new information from application-layer headers to implement more delicate control.

- RASP achieves more accurate detection than that in previous work under realistic simulations.

- It is implemented on the highly programmable P4, which provides potential for future development. Other plaintext-based protocols like HTTP, may be applied in similar approaches.
Thank you!

Chih-Yu Hsieh
r09921a17@ntu.edu.tw

github.com/doraeric/p4-rasp